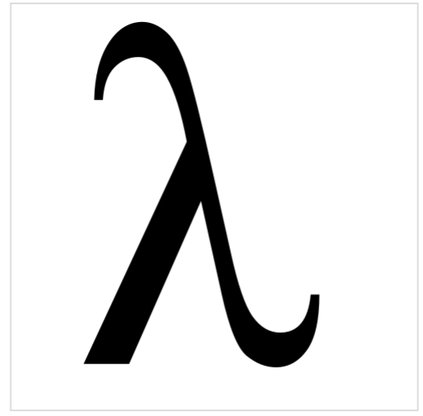
WIKIPEDIA
The Free Encyclopedia

Programming language theory

Programming language theory (**PLT**) is a branch of computer science that deals with the design, implementation, analysis, characterization, and classification of formal languages known as programming languages. Programming language theory is closely related to other fields including linguistics, mathematics, and software engineering.



The lowercase Greek letter λ (lambda) is an unofficial symbol of the field of programming-language theory. This usage derives from the lambda calculus, a model of computation introduced by Alonzo Church in the 1930s and widely used by programming-language researchers. It graces the cover of the classic text *Structure and Interpretation of Computer Programs*,[1] and the title of the so-called Lambda Papers of 1975 to 1980, written by Gerald Jay Sussman and Guy L. Steele Jr., the developers of the Scheme language.

History

In some ways, the history of programming language theory predates even the development of programming languages. The lambda calculus, developed by Alonzo Church and Stephen Cole Kleene in the 1930s, is considered by some to be the world's first programming language, even though it was intended to *model* computation rather than being a means for programmers to *describe* algorithms to a computer system. Many modern functional programming languages have been described as providing a "thin veneer" over the lambda calculus,[2] and many are described easily in terms of it.

The first programming language to be invented was Plankalkül, which was designed by Konrad Zuse in the 1940s, but not publicly known until 1972, and not implemented until 1998. The first widely known and successful high-level programming language was FORTRAN (for Formula Translation), developed from 1954 to 1957 by a team of IBM researchers led by John Backus. The success of FORTRAN led to the formation of a committee of scientists to develop a "universal" computer language; the result of their effort was ALGOL 58. Separately, John McCarthy of Massachusetts Institute of Technology (MIT) developed Lisp, the first language with origins in academia to be successful. With the success of these initial efforts, programming languages became an active topic of research in the 1960s and beyond.

Timeline

Some other key events in the history of programming language theory since then:

1950s

- Noam Chomsky developed the Chomsky hierarchy in the field of linguistics, a discovery which has directly impacted programming language theory and other branches of computer science.

1960s

- In 1962, the Simula language was developed by Ole-Johan Dahl and Kristen Nygaard; it is widely considered to be the first example of an object-oriented programming language; Simula also introduced the concept of coroutines.
- In 1964, Peter Landin is the first to realize Church's lambda calculus can be used to model programming languages. He introduces the SECD machine which "interprets" lambda expressions.
- In 1965, Landin introduces the J operator, essentially a form of continuation.
- In 1966, Landin introduces ISWIM, an abstract computer programming language in his article *The Next 700 Programming Languages*. It is influential in the design of languages leading to the Haskell language.
- In 1966, Corrado Böhm introduced the language CUCH (Curry-Church).[3]
- In 1967, Christopher Strachey publishes his influential set of lecture notes *Fundamental Concepts in Programming Languages*, introducing the terminology *R-values, L-values*, *parametric polymorphism*, and *ad hoc polymorphism*.
- In 1969, J. Roger Hindley publishes *The Principal Type-Scheme of an Object in Combinatory Logic*, later generalized into the Hindley–Milner type inference algorithm.
- In 1969, Tony Hoare introduces the Hoare logic, a form of axiomatic semantics.
- In 1969, William Alvin Howard observed that a "high-level" proof system, referred to as natural deduction, can be directly interpreted in its intuitionistic version as a typed variant of the model of computation known as lambda calculus. This became known as the Curry–Howard correspondence.

1970s

- In 1970, Dana Scott first publishes his work on denotational semantics.
- In 1972, logic programming and Prolog were developed thus allowing computer programs to be expressed as mathematical logic.
- A team of scientists at Xerox PARC led by Alan Kay develop Smalltalk, an object-oriented language widely known for its innovative development environment.
- In 1974, John C. Reynolds discovers System F. It had already been discovered in 1971 by the mathematical logician Jean-Yves Girard.
- From 1975, Gerald Jay Sussman and Guy Steele develop the Scheme language, a Lisp dialect incorporating lexical scoping, a unified namespace, and elements from the actor model including first-class continuations.
- Backus, at the 1977 Turing Award lecture, assailed the current state of industrial languages and proposed a new class of programming languages now known as function-level programming languages.
- In 1977, Gordon Plotkin introduces Programming Computable Functions, an abstract typed functional language.
- In 1978, Robin Milner introduces the Hindley–Milner type system inference algorithm for ML language. Type theory became applied as a discipline to programming languages, this application has led to great advances in type theory over the years.

1980s

- In 1981, Gordon Plotkin publishes his paper on structured operational semantics.
- In 1988, Gilles Kahn published his paper on natural semantics.
- There emerged process calculi, such as the Calculus of Communicating Systems of Robin Milner, and the Communicating sequential processes model of C. A. R. Hoare, as well as similar models of concurrency such as the actor model of Carl Hewitt.
- In 1985, the release of Miranda sparks an academic interest in lazy-evaluated purely functional programming languages. A committee was formed to define an open standard resulting in the release of the Haskell 1.0 standard in 1990.
- Bertrand Meyer created the methodology *design by contract* and incorporated it into the Eiffel language.

1990s

- Gregor Kiczales, Jim Des Rivieres and Daniel G. Bobrow published the book *The Art of the Metaobject Protocol*.
- Eugenio Moggi and Philip Wadler introduced the use of monads for structuring programs written in functional programming languages.

Sub-disciplines and related fields

There are several fields of study that either lie within programming language theory, or which have a profound influence on it; many of these have considerable overlap. In addition, PLT makes use of many other branches of mathematics, including computability theory, category theory, and set theory.

Formal semantics

Formal semantics is the formal specification of the behaviour of computer programs and programming languages. Three common approaches to describe the semantics or "meaning" of a computer program are denotational semantics, operational semantics and axiomatic semantics.

Type theory

Type theory is the study of type systems; which are "a tractable syntactic method for proving the absence of certain program behaviors by classifying phrases according to the kinds of values they compute".[4] Many programming languages are distinguished by the characteristics of their type systems.

Program analysis and transformation

Program analysis is the general problem of examining a program and determining key characteristics (such as the absence of classes of program errors). Program transformation is the process of transforming a program in one form (language) to another form.

Comparative programming language analysis

Comparative programming language analysis seeks to classify languages into different types based on their characteristics; broad categories of languages are often known as programming paradigms.

Generic and metaprogramming

Metaprogramming is the generation of higher-order programs which, when executed, produce programs (possibly in a different language, or in a subset of the original language) as a result.

Domain-specific languages

Domain-specific languages are those constructed to efficiently solve problems in a given domain, or part of such.

Compiler construction

Compiler theory is the theory of writing *compilers* (or more generally, *translators*); programs that translate a program written in one language into another form. The actions of a compiler are traditionally broken up into *syntax analysis* (scanning and parsing), *semantic analysis* (determining what a program should do), *optimization* (improving the performance of a program as indicated by some metric; typically execution speed) and *code generation* (generation and output of an equivalent program in some target language; often the instruction set architecture of a central processing unit (CPU)).

Run-time systems

Run-time systems refer to the development of programming language runtime environments and their components, including virtual machines, garbage collection, and foreign function interfaces.

Journals, publications, and conferences

Conferences are the primary venue for presenting research in programming languages. The most well known conferences include the *Symposium on Principles of Programming Languages* (POPL), *Programming Language Design and Implementation* (PLDI), the *International Conference on Functional Programming* (ICFP), the international conference on *Object-Oriented Programming, Systems, Languages & Applications* (OOPSLA) and the *International Conference on Architectural Support for Programming Languages and Operating Systems* (ASPLOS).

Notable journals that publish PLT research include the *ACM Transactions on Programming Languages and Systems* (TOPLAS), *Journal of Functional Programming* (JFP), *Journal of Functional and Logic Programming*, and *Higher-Order and Symbolic Computation*.

See also

- SIGPLAN
- Very high-level programming language

References

1. Abelson, Harold; Sussman, Gerald Jay; Sussman, Julie (1996). *Structure and Interpretation of Computer Programs* (https://www.worldcat.org/oclc/34576857) (2nd ed.). Cambridge, Massachusetts: MIT Press. ISBN 0-262-01153-0. OCLC 34576857 (https://search.worldcat.org/oclc/34576857).
2. "Models Of Computation" (http://www.c2.com/cgi/wiki?ModelsOfComputation). *wiki.c2.com*. December 3, 2014. Archived (https://web.archive.org/web/20201130055927/http://wiki.c2.com/?ModelsOfComputation) from the original on November 30, 2020.
3. C. Böhm and W. Gross (1996). Introduction to the CUCH. In E. R. Caianiello (ed.), *Automata Theory*, p. 35–64.
4. Benjamin C. Pierce. 2002. Types and Programming Languages (https://books.google.com/books?id=ti6zoAC9Ph8C&dq=%22Types+and+Programming+Languages%22&pg=PR13). MIT Press, Cambridge, Massachusetts, USA.

Further reading

- Abadi, Martín and Cardelli, Luca. *A Theory of Objects*. Springer-Verlag.
- Michael J. C. Gordon. *Programming Language Theory and Its Implementation*. Prentice Hall.
- Gunter, Carl and Mitchell, John C. (eds.). *Theoretical Aspects of Object Oriented Programming Languages: Types, Semantics, and Language Design*. MIT Press.
- Harper, Robert. *Practical Foundations for Programming Languages (https://web.archive.org/web/20070627041059/https://www.cs.cmu.edu/~rwh/plbook/book.pdf)*. Draft version.
- Knuth, Donald E. (2003). *Selected Papers on Computer Languages (http://www-cs-faculty.stanford.edu/~knuth/cl.html)*. Stanford, California: Center for the Study of Language and Information.
- Mitchell, John C. *Foundations for Programming Languages*.
- Mitchell, John C. *Introduction to Programming Language Theory*.
- O'Hearn, Peter. W. and Tennent, Robert. D. (1997). *ALGOL-like Languages (https://web.archive.org/web/20110719175135/http://www.eecs.qmul.ac.uk/~ohearn/Algol/algol.html)*. Progress in Theoretical Computer Science. Birkhauser, Boston.
- Pierce, Benjamin C. (2002). *Types and Programming Languages (http://www.cis.upenn.edu/~bcpierce/tapl/main.html)*. MIT Press.
- Pierce, Benjamin C. *Advanced Topics in Types and Programming Languages*.
- Pierce, Benjamin C. *et al.* (2010). *Software Foundations (http://www.cis.upenn.edu/~bcpierce/sf/)*.

External links

- Lambda the Ultimate (http://lambda-the-ultimate.org/policies#Purpose), a community weblog for professional discussion and repository of documents on programming language theory.
- Great Works in Programming Languages (http://www.cis.upenn.edu/~bcpierce/courses/670Fall04/GreatWorksInPL.shtml). Collected by Benjamin C. Pierce (University of Pennsylvania).
- Classic Papers in Programming Languages and Logic (https://www.cs.cmu.edu/~crary/819-f09/). Collected by Karl Crary (Carnegie Mellon University).
- Programming Language Research (https://www.cs.cmu.edu/afs/cs.cmu.edu/user/mleone/web/language-research.html). Directory by Mark Leone.
- λ-Calculus: Then & Now (http://turing100.acm.org/lambda_calculus_timeline.pdf) by Dana S. Scott for the ACM Turing Centenary Celebration
- Grand Challenges in Programming Languages (http://plgrand.blogspot.com/). Panel session at POPL 2009.

Retrieved from "https://en.wikipedia.org/w/index.php?title=Programming_language_theory&oldid=1286649926"