The Arend theorem prover

Valery Isaev Fedor Part Sergey Sinchuk

This talk is an exposition of the Arend Interactive Theorem Prover (ITP) [1] developed at JetBrains Research. The development of Arend started in 2015 at JetBrains Research with the aim of building a modern ITP with a *native* support for HoTT/UF [7]. Its three major components are the Arend language, which based on a variant of MLTT with a native support for HoTT/UF, a tooling for it provided by Arend plugin for IntelliJ IDEA and the standard library arend-lib which focuses on formalization of constructive mathematics, synthetic homotopy theory and computer science.

Type theory. At the core of the type theory of Arend is the *homotopy type theory with interval type* (HoTT-I) formulated by Valery Isaev in 2014 [2]. HoTT-I is a minor, more extensional, modification of MLTT obtained by introducing a primitive type $\mathbf{I}$ for the interval and defining the equality type (the path type) in terms of $\mathbf{I}$. Function extensionality is derivable and is fully computational in this theory. This setup gives rise to a variant of cubical syntax: n-dimensional homotopies in a type A are naturally represented in HoTT-I as functions $\mathbf{I}^n \to A$. The univalence is added to HoTT-I as an axiom with some built-in computational rules.

The native support for HoTT/UF in the type theory of Arend is further amplified by the following extensions of HoTT-I:

1. Inductive types with conditions. Using the primitive $\mathbf{I}$, a path constructor can be defined simply as a constructor $\mathsf{pcon}\,(i : \mathbf{I})$ with an interval type parameter, or multiple parameters for higher-dimensional homotopies. The only difference with an ordinary constructor is that there could be *conditions* that dictate how pcon evaluates at the ends of the interval to other constructors: $\mathsf{pcon\,left} \Rightarrow \mathsf{con}_1$, $\mathsf{pcon\,right} \Rightarrow \mathsf{con}_2$. In particular, it implies $con_1 = con_2$ but does not lead to contradictions as in the case of standard inductive types in MLTT, thanks to a modified elimination principle that requires the preservation of conditions. This provides support for higher inductive types and, in particular, quotients with computational behavior [9].

2. Universes Prop and Set matching h-propositions and h-sets. Arend incorporates an *impredicative* universe Prop that captures the logic of h-propositions and a predicative hierarchy of universes $\mathsf{Set}\,0 \subset \mathsf{Set}\,1 \subset \ldots$ for h-sets. The universes Prop and Set provide a *syntactic* framework that facilitates working within the set theory domain, akin to the elementary topos of h-sets, and distinctively delineate the set-level aspect of the type theory from the higher levels, ensuring that set-theoretic formalization is confined strictly to the subtheory prescribed by these universes.

3. Polymorphism for homotopy levels. Arend supports a mechanism for polymorphism on the homotopy level: a single polymorphic definition can be instantiated to multiple versions for universes Prop, $\mathsf{Set}\,n$ and universes of all types $\mathsf{Type}\,n$.

The above outlines the core fragment of Arend's type theory as it relates to HoTT/UF. Its key properties include:

Computational behavior. Unlike MLTT augmented merely with axioms for univalence and higher inductive types, Arend's type theory includes a number of computational reduction rules for involved terms, which prove crucial for practical usage. However, it lacks sufficient computational rules to render the theory fully computational: the MLTT property that every closed term of type Nat evaluates to a canonical number does not hold in Arend. The core theory of Arend represents one of the earliest and simplest forms of *cubical type theory* [6, 8], in which the interval still functions as a type. More advanced, fully computational cubical type theories have since evolved, in which univalence is derivable. However, these approaches are considerably more complex, featuring two-level theories where the interval $\mathbf{I}$ is *not a type*. While the development of such fully computational theories is of theoretical interest, the complexity of two-level frameworks do not currently justify their use in practical applications. The type theory of Arend could potentially be adjusted to enhance its computational aspects, but practical experiences in formalization have not demonstrated a significant need for such enhancements.

Constructivity. The type theory of Arend is inherently constructive. In its core theory, the law of excluded middle does not hold, and the axiom of choice is valid only in the form of *unique choice*. While the axioms of classical logic can be consistently added and utilized in libraries, the main Arend library, arend-lib, is dedicated to constructive mathematics, and no standard metas (tactics) utilize any classical axioms.

Finally, the type theory of Arend has the following extensions which are of great importance for formalization and yet are not present in Coq, Lean or Agda:

1. Records with partial implementations and anonymous extensions. Type classes and records in Arend support multiple inheritance with *partial implementations*: if D extends C an arbitrary subset of fields of C can be implemented in D. Namely, for a field f of C its extension D may contain an expression $f \Rightarrow a$ specifying a value a for f. To create an object of D, only unimplemented fields need to be specified. This erases the difference between parameters and fields of classes and records and allows for extensive flexibility in constructing hierarchies of definitions. Moreover, a record type, where a subset of fields of C get implemented, can be created on the fly in Arend as an *anonymous extension* $C\{f_1 \Rightarrow a_1, \ldots, f_k \Rightarrow a_k\}$.

2. Array types. Arend has the type of arrays $\mathsf{Array}\, A$, where $A : \mathsf{Type}$, which subsumes the type of lists of elements of type A, the type of vectors of elements of type A of fixed length n and the type $\mathsf{Fin}\, n \to A$ of functions from a finite set of cardinality n to A. The type Array is a record with fields $A : \mathsf{Type}$, $\mathsf{len} : \mathsf{Nat}$ and $\mathsf{at} : \mathsf{Fin}\, \mathsf{len} \to A$. The anonymous extension $\mathsf{Array}\, A$ is the type of arrays of various lengths with elements of type A and the extension $\mathsf{Array}\, A\, n$ is the type of arrays of fixed length n. The key property of the type of arrays is a computational form of extensionality: if two arrays of constant length are elementwise computationally equal then they are computationally equal.

IntelliJ IDEA plugin. The easiest way to use Arend is through the tooling provided by the Arend plugin for IntelliJ IDEA. IntelliJ IDEA, developed by JetBrains, is a versatile IDE with intelligent code assistance, robust refactoring tools, and seamless integration with languages like Java, Kotlin, and Python. It has a variety of built-in tools for version control, debugging and testing. Some of the features of Arend plugin:

1. Incremental type checking. Under normal circumstances the Arend plugin uses the so-called "on-the-fly" (or smart) mode of type checking. In this mode, the type checker operates in a background thread, automatically reprocessing each code modification made by the user.

2. Editor features. The Arend editor supports a variety of features that greatly simplify working with Arend code such as: auto-completion of identifiers; auto-fixes for Arend errors; code auto-adjustments such as adding missing clauses in pattern matching; refactorings such as handling consequences of changing signatures of definitions or moving definitions; navigation tools such as Go to Declaration, Find Usages or Proof Search; quick documentation popups supporting LaTeX; parameter hint tooltip which can be invoked in application expressions and so on.

The arend-lib library. The main Arend library arend-lib can be divided into three parts which are developed constructively (without the excluded middle or the axiom of choice):

1. Constructive mathematics. This is the main part of the library. The mathematical landscape in the constructive setting is richer than in the classical setting since classically equivalent definitions often become inequivalent constructively.

This part contains the following: schemes via locally ringed locales; PID domains and the proof that they are 1-dimensional Smith domains; splitting fields of polynomials and algebraic closure for countable, decidable fields; connection between zero-dimensional and integral extensions; matrices over commutative rings, determinants, characteristic polynomials, Cayley-Hamilton theorem; linear algebra over Smith domains; integral ring extensions; polynomials over one or several variables; Nakayama's lemma; derivative over topological rings; directed limits for sequences and functions; series and power series; natural, integer, rational, real and complex numbers and various structures on them; categories, functors, adjoint functors, Kan extensions, (co)limits; elementary topoi and Grothendieck topoi; topological spaces, locales, uniform spaces, completion of spaces.

The main references are the books [5, 4] and the paper on constructive complete spaces by Isaev [3].

2. Synthetic homotopy theory. The following has been formalized synthetically that is under types as homotopy types viewpoint: Eckmann-Hilton argument; $K_1(G)$; Hopf fibration; localization of universes and modalities; Generalized Blakers-Massey theorem.

3. Computer science. Currently this part consists of formalization of high-order term rewriting systems. The planned future formalizations include fragments of computational complexity theory.

References

[1] Arend site. https://arend-lang.github.io.

[2] Valery Isaev. Models of homotopy type theory with an interval type, 2020. Available at: https://arxiv.org/abs/2004.14195.

[3] Valery Isaev. A constructive approach to complete spaces, 2024. Available at: https://arxiv.org/abs/2401.12345.

[4] H. Lombardi and C. Quitté. *Commutative Algebra: Constructive Methods: Finite Projective Modules*. Algebra and Applications. Springer Netherlands, 2015.

[5] R. Mines, F. Richman, and W. Ruitenburg. *A Course in Constructive Algebra*. 3Island Press, 1987.

[6] Anders Mörtberg. Cubical methods in homotopy type theory and univalent foundations. *Mathematical Structures in Computer Science*, 31(10):1147–1184, 2021.

[7] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. https://homotopytypetheory.org/book, Institute for Advanced Study, 2013.

[8] Andrea Vezzosi, Anders Mörtberg, and Andreas Abel. Cubical agda: a dependently typed programming language with univalence and higher inductive types. *Proc. ACM Program. Lang.*, 2019.

[9] Tesla Zhang and Valery Isaev. (co)condition hits the path, 2024. Available at: url=https://arxiv.org/abs/2405.12994.