

Lingenic-Text: A Formally Verified Unicode 17.0 Text Processing Library in SPARK/Ada

Danslav Slavenskoj
Lingenic LLC
danslav@lingenic.com

Introduction

The processing of Unicode text is among the most foundational operations in modern computing, yet the algorithms that govern it—segmentation, normalization, bidirectional layout, collation—are specified across more than a dozen Unicode Technical Annexes and Reports, each encoding rules of considerable complexity. Implementations of these algorithms in widely used libraries have historically been written in memory-unsafe languages without formal guarantees, relying on testing alone to establish correctness. The question of whether a Unicode text processing library can be not merely tested but *proved* correct—with machine-checked guarantees of both the absence of runtime errors and functional conformance to the Unicode Standard—has not, to the authors’ knowledge, been addressed prior to this work.

Lingenic-Text is a complete implementation of the Unicode 17.0 text processing stack, written in SPARK/Ada (Ada 2022) and formally verified with GNATprove. The library comprises approximately 30,700 lines of Ada source across 53 files, implementing fourteen distinct modules: UTF-8 encoding and decoding (RFC 3629), grapheme cluster segmentation, word segmentation, and sentence segmentation (UAX #29), line breaking (UAX #14), normalization to all four forms (UAX #15), case mapping including full multi-character mappings and context-sensitive rules (Unicode §3.13), collation with DUCET support (UTS #10), the full Unicode Bidirectional Algorithm including bracket pair resolution (UAX #9), East Asian width determination (UAX #11), emoji classification and property lookup (UTS #51), identifier detection (UAX #31), and internationalized domain name processing with Punycode (UTS #46, RFC 3492). Every verification condition—9,214 in total, spanning runtime checks, functional contracts, assertions, termination, initialization, and data dependencies—is discharged by the prover at Level 4. No `pragma Assume` appears anywhere in the codebase. Conformance testing against Unicode Consortium test suites and reference data passes all 504,634 test cases.

Architecture and Verification Approach

The verification architecture factors into two links of different strength. The first link is a formal proof: for every subprogram in the library, a ghost specification encodes the intended behavior as pure expression functions or recursive ghost functions, and GNATprove proves that the implementation satisfies this specification for all possible inputs. This link is machine-checked and universal. Ghost code in SPARK is erased entirely at compile time, imposing zero runtime cost. The second link is conformance testing against the Unicode Consortium test suites—`GraphemeBreakTest.txt`, `NormalizationTest.txt`, `BidiCharacterTest.txt`, and others—which validates that the ghost specifications themselves faithfully encode the rules of the Unicode Standard. This link is empirical: it is validation by examples, and its strength is bounded by the coverage of the test suites. The end-to-end guarantee is therefore proved (implementation $\models$ specification) $\wedge$ tested(specification $\approx$ standard). Along the implementation-correctness axis,

the guarantee is a proof; along the standard-conformance axis, it is only as strong as the test suite. Since the Unicode Standard is a natural-language document, the conformance boundary cannot be eliminated by formal methods alone, but the test suites are the Consortium’s own conformance instruments, and the library passes all 504,634 cases.

Two principal proof patterns emerge across the library’s modules. In the first, used by the segmentation algorithms, the Unicode rules are encoded as a recursive ghost function with a `Subprogram_Variant` annotation proving termination. The implementation is a forward state machine realized as a loop, whose invariant asserts equivalence with the recursive specification at every iteration. The postcondition of the public subprogram then states that its output equals the value of the recursive ghost function applied to the input. In the second pattern, used by normalization and case mapping, a generic text transformation framework carries a ghost predicate (`Partial_Valid`) as its loop invariant. Each callback’s postcondition preserves this invariant, and a finishing postcondition bridges from the partial invariant to the full output specification. This generic is instantiated by each module with its own callback and specification, yielding proved correctness without duplicating the proof scaffolding.

All property lookups—script, general category, grapheme break property, word break, sentence break, line break, East Asian width, Bidi class, joining type, and others—are implemented as flat arrays indexed directly by codepoint, giving $O(1)$ access with no dynamic allocation, no hash tables, and no trees. The Unicode Character Database files are read from disk at initialization by a proved UCD parser, whose postcondition guarantees that every codepoint’s property value in the populated table matches the value specified by a recursive ghost function encoding a model of the UAX #44 property file format. The fidelity of that model to the actual UAX #44 text is, like the algorithm specifications, established by test rather than by proof. This design permits updating to a new Unicode version by replacing the data files in the `ucd/` directory, without modifying any source code.

Scope and Capabilities

The library provides a complete C API as a static library with 53 exported functions, enabling integration with C, C++, and any language supporting C foreign function interfaces. The C binding is a thin validation layer: every entry point checks its arguments against the precondition of the proved SPARK subprogram it wraps, returning an error code on violation, so that the machine-checked postconditions of the core apply to every successful call through the C interface.

Among the more complex modules, the Bidirectional Algorithm implementation handles the full rule set of UAX #9, including explicit embeddings, overrides, and isolates, isolating run sequence resolution, and bracket pair matching under rule N0 with the BD16 algorithm. The reordering procedure produces a proved permutation of the input. The collation module implements UTS #10 with both Non-Ignorable and Shifted variable weighting, contraction handling, and implicit weight computation for CJK Unified Ideographs, Tangut, Nushu, and Khitan Small Script. The IDNA module implements the full UTS #46 processing pipeline with Punycode encoding (RFC 3492), ContextJ validation (RFC 5892), Bidi domain name rules (RFC 5893), and DNS length checks.

The library enforces several invariants by construction. No heap allocation occurs; all buffers are bounded arrays with every index proved in range, eliminating buffer overflows as a class of defect. No runtime exceptions are raised; all error conditions are communicated through status codes. Runtime checks are suppressed in the compiled binary (`-gnatp`) because GNATprove has already proved their absence. The sole code outside SPARK verification is the file I/O routine that reads UCD data from disk; every other subprogram is machine-checked.

Availability

Lingenic-Text version 1.1.0 implements Unicode Standard 17.0. The source code, comprising all SPARK/Ada sources, the C binding, and conformance test programs, is available under the Lingenic Source-Available License v2.3. Production use requires a separate license from Lingenic LLC. The Unicode Character Database files included in the distribution are © Unicode, Inc. and are distributed under the Unicode License V3.