Issue XI: Minimal Proving System of Henk with Countable Universes for Erlang/OTP

Maxim Sokhatsky

[1] National Technical University of Ukraine
Igor Sikorsky Kyiv Polytechnical Institute
15 березня 2026 р.

Анотація

This paper presents **Henk**, a pure type system (PTS) with a countable hierarchy of universes indexed by natural numbers. Henk is implemented as a minimal dependently typed core in Erlang and extracts to BEAM bytecode via type erasure. The type checker supports configurable predicative and impredicative modes and follows the standard PTS rules of Barendregt. Syntax is compatible with Morte, extended with explicit universe indices. We provide the complete formal specification, the reference implementation of the type checker (normalization, conversion, shifting, substitution), and the erasure/extraction pipeline. Data types are encoded via Church–Böhm–Berarducci representations; effects use recursive free monads and corecursive comonads linked to Erlang I/O primitives. Higher-level languages (Frank, Per, Christine) that extend Henk with inductive types, Σ-types and equality are outlined as future work. The core is limited to 1K lines of Erlang for the essential passes. This work builds directly on the Calculus of Constructions, AUTOMATH and the original Henk design of Meijer and Peyton Jones.

Зміст

1 Introduction

Verified software requires a trusted computational core whose meta-theory is well-understood. Henk is such a core: a pure type system whose only primitive is the dependent function type Π and whose universes are indexed by $\mathbb{N}$. It extracts directly to untyped Erlang terms that run on the BEAM virtual machine.

Henk is deliberately minimal: five AST constructors, one axiom schema, four deduction rules, and no built-in recursion or inductive types. All data (Booleans, natural numbers, lists, I/O protocols) are encoded via Church–Böhm–Berarducci representations. The implementation is written entirely in Erlang; the extracted code is plain Erlang that can be loaded into any OTP application.

Higher languages in the Groupoid Infinity family (Per, Christine) embed Henk as their core and add inductive definitions and equality reasoning on top.

1.1 Extraction to Erlang

By Curry–Howard, proofs are programs. Henk erases all type information and universes, producing a pure λ-term that is then translated to Erlang functions. The resulting code is ordinary BEAM bytecode with no runtime type checking beyond Erlang's own. Host primitives (I/O, arithmetic) are supplied at extraction time.

1.2 System Architecture

Henk consists of:

- Level 0: strongly-typed Erlang runtime (BEAM).
- Level 1: PTS Core (type checker + extractor).
- Level 2: higher languages (Henk, Frank, Christine).

All compiler passes are implemented in Erlang; the target remains Erlang.

2 Formal Definition of the Henk PTS

Henk is a Pure Type System in the sense of Barendregt. We give the precise specification that the implementation realises.

2.1 Sorts, Axioms and Rules

Sorts: $\mathcal{S} = \{\mathsf{Type}_i \mid i \in \mathbb{N}\}$.

Two configurations are supported:

Impredicative mode

$$\begin{array}{rll} \text{Axioms} & \mathsf{Type}_i : \mathsf{Type}_{i+1} & \text{for all } i, \\ \text{Rules} & (\mathsf{Type}_i, \mathsf{Type}_j, \mathsf{Type}_j) & \text{for all } i, j. \end{array}$$

Predicative mode

$$\begin{array}{rll} \text{Axioms} & \mathsf{Type}_i : \mathsf{Type}_j & \text{for all } i < j, \\ \text{Rules} & (\mathsf{Type}_i, \mathsf{Type}_j, \mathsf{Type}_{\max(i,j)}) & \text{for all } i, j. \end{array}$$

Both configurations satisfy Barendregt's criteria and are therefore strongly normalizing and consistent (Terlouw, 1993; Barendregt, 1992).

2.2 Judgments

Contexts Γ are lists of declarations $x : A$ with $A : \mathsf{Type}_k$. The typing relation $\Gamma \vdash t : T$ is defined by the standard PTS rules:

- (Sort) $\vdash \mathsf{Type}_i : \mathsf{Type}_{i+1}$
- (Var) $\Gamma, x : A \vdash x : A$ if $\Gamma \vdash A : \mathsf{Type}_k$
- (Pi) $\Gamma \vdash A : \mathsf{Type}_i \quad \Gamma, x : A \vdash B : \mathsf{Type}_j \quad \Rightarrow \quad \Gamma \vdash \Pi x : A.B : \mathsf{Type}_k$ where (i, j, k) is a rule
- (Lam) $\Gamma, x : A \vdash b : B \quad \Rightarrow \quad \Gamma \vdash \lambda x : A.b : \Pi x : A.B$
- (App) $\Gamma \vdash f : \Pi x : A.B \quad \Gamma \vdash a : A \quad \Rightarrow \quad \Gamma \vdash f\, a : B[a/x]$
- (Conv) $\Gamma \vdash a : A \quad \Gamma \vdash B : \mathsf{Type}_k \quad A \equiv B \quad \Rightarrow \quad \Gamma \vdash a : B$

Definitional equality $\equiv$ is β-convertibility obtained by full normalization (the implementation uses weak-head normalization followed by structural comparison with shifting and substitution).

Subject reduction, strong normalization and decidability of type checking follow from the meta-theory of PTS with countable universes (standard results; see Barendregt, 1992 and the references therein). The implementation realises exactly these rules; formal verification of the 60-line type-checker module is left as future work.

2.3 Abstract Syntax

$$t \quad ::= \quad x \mid \mathsf{Type}_i \mid \Pi x : A.B \mid \lambda x : A.b \mid f\, a$$

De Bruijn indices with explicit universe tags are used internally; the surface syntax accepts named variables and Morte-style notation.

Табл. 1: Compiler passes

Module	LOC	Description
om_erase	53	type erasure to untyped λ-terms
om_tok	53	hand-written lexer
om_parse	61	hand-written 2-pass parser
om_type	102	normalization, shifting, substitution, typing
om_extract	120	translation to Erlang abstract format

3 Implementation

The reference implementation consists of five modules (total $\approx$ 265 LOC):

The type-checker implements the rules above via:

- `star/1` – universe level extraction,
- `shift/3`, `subst/4` – de Bruijn operations,
- `norm/1` – weak-head normalization,
- `eq/2` – β-convertibility after normalization,
- `type/2` – the bidirectional typing algorithm.

All functions are total and terminate because of strong normalization.

4 Erasure and Extraction

Erasure removes all universe annotations and dependent Π-binders, mapping to ordinary λ-terms:

$$\begin{cases} \mathrm{erase}(\mathsf{Type}_i) = \texttt{unit}, \\ \mathrm{erase}(\Pi x : A.B) = \mathrm{erase}(B), \\ \mathrm{erase}(\lambda x : A.b) = \lambda x.\, \mathrm{erase}(b), \\ \mathrm{erase}(f\ a) = \mathrm{erase}(f)\, \mathrm{erase}(a). \end{cases}$$

The extracted term is then pretty-printed as Erlang functions using `erl_abstract_format`. Primitive operations (I/O, arithmetic) are supplied as higher-order Erlang functions at extraction time. The result runs directly on BEAM with no additional runtime overhead beyond ordinary Erlang.

Exampled the Church-encoded successor function which extracts to a plain Erlang closure that can be applied to any host-provided natural-number representation.

5 Tutorial: Encodings and Effects

All data are Church-encoded. The prelude (available in the repository) provides:

- `Bool`, `Nat`, `List`, `Maybe`, `Either`,
- Σ- and equality types via Böhm–Berarducci encodings,
- two I/O models:
  1. recursive free monad (finite scripts),
  2. corecursive comonad (infinite servers/processes).

The corecursive model directly mirrors Erlang processes: an anamorphism generates an infinite state machine whose step function is linked to `io:get_line` and `io:put_chars`.

Example extraction and execution of a corecursive echo server is shown in the repository tests; the resulting Erlang code runs indefinitely and interacts with the console exactly as a native OTP process.

6 Higher-Level Languages

Henk is the trusted core. The planned language Christine adds:

- inductive families (via fixpoints or self-types),
- Σ-types and dependent records,
- Martin-Löf identity type with eliminators,
- case analysis and recursion on inductive types.

These extensions are compiled down to Henk + host primitives, preserving the extraction pipeline.

7 Limitations

- No inductive types or fixpoints in the core (by design; they are added in higher layers).
- The type-checker correctness has not yet been mechanically verified.
- Performance benchmarks against Coq/Agda extraction or native Erlang are not included in this paper.
- Extraction yields untyped code; any safety guarantees beyond those of Erlang itself are the responsibility of the higher-level compiler.

8 Conclusion

Henk is a mathematically precise, minimal PTS implementation whose meta-theory is standard and well-understood. It provides a trustworthy extraction path from dependent types to the Erlang ecosystem. The implementation is compact, the formal rules are fully specified, and the encodings for data and effects are complete. Future work will add inductive types in Christine and formally verify the type-checker kernel.

The source code, prelude, test suite and live REPL are available at `https://github.com/groupoid/henk` and `https://henk.groupoid.space`.

Література

[1] Henk Barendregt, *Lambda Calculi with Types*, Handbook of Logic in Computer Science, Vol. 2, Oxford University Press, 1992.

[2] J. Terlouw, *Strong Normalization for the Calculus of Constructions*, Technical Report, University of Nijmegen, 1993.

[3] Simon Peyton Jones and Erik Meijer, *Henk: A Typed Intermediate Language*, Types in Compilation Workshop, 1997.

[4] Thierry Coquand and Gérard Huet, *The Calculus of Constructions*, Information and Computation 95(2), 1988.

[5] Per Martin-Löf, *Intuitionistic Type Theory*, Bibliopolis, 1984.

[6] Herman Geuvers, *Induction is not Derivable in Second-Order Dependent Type Theory*, Typed Lambda Calculi and Applications, 2001.

[7] Corrado Böhm and Alessandro Berarducci, *Automatic Synthesis of Typed Lambda-Programs on Term Algebras*, Theoretical Computer Science 39, 1985.

[8] Peng Fu and Aaron Stump, *Self Types for Dependently Typed Lambda Encodings*, RTA-TLCA 2014.