nC

LANGUAGE DOCUMENTATION

nC Compiler Architecture and Bootstrap

Compiler contracts, memory, ABI, materialization and the scope of native V81 reconstruction.

Bootstrap edition V81
24 October 2026
github.com/Arizt00/nc

01 Contracts between levels
02 Parser AST semantics and IR
03 Machine code and control flow
04 Memory and representation
05 Materialization ABI and linking
06 What bootstrap V81 demonstrates
07 Reproducing and reviewing evidence
08 Releases and contributions

Contracts between levels

nC exposes several ways to express a program within one semantic model. Ordinary source, the foundational root, machine code and primitives have distinct responsibilities. An operation must retain the information required as it crosses these levels.

`tools/ncc/nc/src/` contains the native compiler written in nC. `crates/` contains the retained Rust implementation and development tools. `runtime/nc/` contains runtime bodies and providers. Build tools and contracts under `bootstrap/` connect inputs and generations.

Phase	Responsibility
Lexer and parser	Tokenization and syntactic structure
AST	Identity of declarations, types, relationships and surfaces
Semantic analysis	Symbol resolution and validation of types, arity and control flow
IR	Representation of the analyzed program for lowering and transformations
MCT and backend	Materialization according to target layout, ISA and ABI
Linking	Symbol resolution, relocations and image dependencies
Execution	Verification of results and observable effects

This is a responsibility map. Some surfaces have their own entry points and validation paths. A change must identify the affected route and the information it preserves.

Design documentation specifies the contract. Version evidence records results for particular inputs, compiler artifacts and environments. Both must be traceable to implementation and fixtures.

Parser AST semantics and IR

The parser recognizes compact imports such as `use <io:output>;`, named containers, sections and grouped functions. Material V2 tests in `crates/ncc-parser/src/lib.rs` cover this surface. A historical test name does not override the grammar that the test actually exercises.

The AST preserves the entities needed by analysis: places, functions, types, visibility, flow, layout contracts and material surfaces. Replacing an existing typed representation with an unstructured string defers errors to later phases.

Semantic analysis checks relationships accepted by the language. Signatures require consistent arity and types; references must resolve to the intended symbols; control operations must satisfy their context. Established diagnostics are observable tool behavior.

The IR carries analyzed meaning into later phases. An AST node alone is insufficient: lowering and transformations must preserve its semantics. Tests should inspect the properties that could be lost at those boundaries.

A surface change should include positive and negative fixtures, plus an IR observation when translation can alter the contract. Native-generation changes also require artifact checks and execution where the environment supports it.

Tests should cover destination resolution, signature compatibility and evaluation order using expectations independent of the implementation. Catalog consistency tests should compare declared symbol sets and reject duplicates rather than relying only on a fixed count.

Machine code and control flow

`[@machine]` represents instructions with typed structure. Operands, registers, labels and terminators must remain coherent from parsing through IR. Extending materialization must preserve those relationships.

A label identifies a position or block in the supported model. A branch connects control-flow paths. Conditional transfers preserve the meaning of the flags or operations they consume. Native emission must retain the dependency that made the branch valid in the earlier representation.

Terminators delimit what can execute after a block. An instruction placed after a final needs defined reachability. Accepting a block without its required terminator shifts a control-flow inconsistency into the backend.

`checked_add_overflow_machine.n` checks an explicit overflow branch. It does not imply automatic overflow protection for every arithmetic operation in the language.

Object `.nm` modules use the machine CLI. In retained Stage0, `machine emit-ir` is the correct entry point for `object_elf64.nm`; general `emit-ir` is not interchangeable with it.

An ISA extension requires operations, operand types, restrictions, emission rules and tests. A target name or instruction table is not evidence of execution on hardware. Physical support requires separate evidence for that profile.

Memory and representation

Ordinary source retains a manual/raw contract. The compiler does not introduce ownership, general bounds checks, automatic pointer-lifetime validation or hidden release. Type properties must be distinguished from memory-safety guarantees.

Representation covers size, alignment, offsets, padding and ABI. An aggregate may have a valid in-memory layout while requiring additional rules for argument passing or return values. Data layout and calling convention must be checked together.

V81 materializes alignments above 16 bytes on the documented native Windows and Linux/WSL x86-64 profiles. Evidence covers powers of two from 32 bytes to 1 MiB. COFF section alignment is limited to 8192 bytes in this work; direct PE and ELF paths support larger alignments within the documented profile.

A request that the format cannot represent must be rejected. Silent truncation changes the actual address while reporting apparent success. Material verification checks ranges, powers of two and offsets before consuming the artifact.

The stack is also a material constraint. Large Windows tests declare a 16 MiB reserve; Linux ABI tests adjust the stack limit according to the reproduction recipe. The emitter touches stack pages in the documented profile. This does not add an implicit heap or remove system limits.

Layout checks across ten profiles are representation calculations. V81 physical execution evidence applies to the stated x86-64 hosts. These establish different kinds of support and should be reported separately.

Materialization ABI and linking

MCT participates in object and image generation. The backend translates program decisions into target-compatible bytes and metadata. Linking must resolve symbols and relocations without accidentally relying on retained artifacts from an earlier build.

V81 changes native sources including `native_body.n`, `mct.n`, `mct_object.n`, `mct_link.n` and `mct_image_elf.n`. Their responsibilities include stack intervals, temporaries, arguments, returns, symbol alignment and alignment against final virtual addresses.

ABI details remain significant when the source is unchanged. Aggregates may travel by reference on Win64 or through a realigned argument area on SysV. Validation includes direct and indirect calls, aggregate returns, nesting and recursion.

The linked runtime must match the symbols emitted by the compiler. Successful analysis does not resolve a missing export. Provider identity, dependencies and library search rules are part of the reproduction.

Caches contain derived materials. Cache identity must change when the emitter contract changes. V81 rejects corrupt materials even after their manifest digests are recomputed: a matching digest does not validate an altered artifact's internal structure.

Retain source, IR or material, object, image and process output separately when investigating regressions. Printed output alone can conceal differences in layout, linkage and sections that surface in another program.

What bootstrap V81 demonstrates

The V81 dossier records four final reconstructions: B2 and C2 on Windows, and B2 and C2 on Linux/WSL. Each rebuilds 74 units in a new directory using `project objects --no-incremental`, without LTO or retained project objects.

The generations also rebuild their providers: 36 on Windows and 35 on Linux. Compiler B2 and C2 are byte-identical within each host. Their recorded digests are:

```text
Windows x86-64
4ff2300d30fbe99a4c541481e0fab9df284d77c32d1df082084440b9718854c9

Linux x86-64
e3d6fc744f0768d19a3f44ca93eddcb90cc3f89caee536f68f8f5ce486eae1e7
```

Identity is established within each operating system. A Windows PE and a Linux ELF are not claimed to have identical bytes. Manifests associate each digest with its artifact.

Stage0 remains an AST oracle for that validation corpus, rather than the constructor of B and C. Global Stage3 promotion is explicitly open, and the installed toolchain is retained.

The dossier reports 147 complete programs and 1472 checks per run, 38 runtime contracts per generation and host, and layout checks across ten profiles. These are retained V81 results. Publication smoke tests are recorded separately and do not replace full reconstruction.

Support expands through evidence for missing routes: inputs, emitted artifacts, dependencies and execution. Compatibility claims must match the scope of those checks.

Reproducing and reviewing evidence

`bootstrap/frontend-v81-build.json` fixes reconstruction sources. Builders verify SHA-256 during copying and reject destinations containing `.ncc`. This distinguishes a clean generation from reuse of previous objects.

The sequence is to build B with the seed, build its runtime, build C with B and its providers, then repeat the checks on C. `docs/NCC_MODULES_V81_REPRO.md` records complete builder commands and arguments.

```text
seed -> compiler B -> runtime B
compiler B + runtime B -> compiler C -> runtime C
artifact comparison and contract execution
```

Reports retain expected output, exit codes, objects used and exact differences. Failed attempts remain separate from final runs. A successful retry should not erase the cause of the previous failure.

Adversarial tests establish rejection of altered inputs. Recorded cases cover invalid alignments, inconsistent offsets and modified images. An image rejected by validation must not be executed as a substitute for resolving the rejection.

The documented unwind tests apply real Windows APIs to metadata and a test context. They do not inject a real exception at every prologue instruction. That limitation belongs to the technical claim.

Reproduction can require system tools, stack settings and external runners. Record these conditions before comparing a local result with the dossier.

Releases and contributions

A release needs an executable entry path: purpose, requirements, a program, expected output and an issue channel. Packages identify their edition, target, hashes and included runtime.

Development caches, local reports, temporary files and build outputs do not belong in the source distribution. Public export uses an explicit file selection and a hash manifest. Native kits are distributed as Release assets.

Bootstrap releases remain prereleases with explicit limits. Test badges require executed checks. Download links must resolve, and performance claims require reproducible measurements.

A technical contribution explains the problem, affected contract, before-and-after behavior and validation. Parser, AST, semantics and IR changes must preserve continuity. ABI and memory work needs boundary cases and rejection tests. Documentation changes require working links and accurate examples.

Technical references

- `AGENTS.md` and `docs/syntax.md`.
- `crates/ncc-parser/src/lib.rs` and `crates/ncc-stage0-bridge/build.rs`.
- `docs/NCC_MODULES_V81_DOSSIER.md`.
- `docs/NCC_MODULES_V81_ABI_LAYOUT.md` and `docs/NCC_MODULES_V81_REPRO.md`.
- `docs/NCC_MODULES_V81_ATTEMPTS.md`.
- `bootstrap/frontend-v81-build.json` and `bootstrap/frontend-v81-contracts.json`.

These references define this edition's scope. Subsequent work must update contracts, tests and public documentation together.