Anoma Research Topics | TECHNICAL REPORT

Nock for Functional Programmers

Lukasz Czajka[a]

[a]Heliax AG

* *E-Mail: lukasz@heliax.dev*

Abstract

We present Nock using mainstream programming language theory notations and terminology. The intention is to make Nock more accessible and to outline its use as a compilation target for a high-level functional language.

Keywords: Nock ; combinators ; big-step operational semantics ;

(Received: December 17, 2024; Version: December 18, 2024)

Contents

1. Introduction

Nock [noca] is a Turing-complete low-level combinator language that is a compilation target for the Hoon [hoo] programming language. Both Hoon and Nock are part of the Urbit [urb] project. The purpose of this note is to present Nock in a more accessible manner, using notation and terminology familiar to functional programmers and type theorists. This makes the specification of Nock more precise, as in fact some statements in [nocb] are slightly ambiguous.

Recently, Nock has been utilized as a target for the Juvix [Hel24] programming language to enable transparent execution in Anoma [GYB23]. The version of Nock presented in this report is used internally by the Juvix compiler.

2. Nock definition

Definition 1. *An* atom $a, b, c \in \mathcal{A}$ *is a natural number* $n \in \mathbb{N}$.

Terms $t, s, r \in \mathcal{T}$ *are defined inductively.*

- *An atom is a term.*
- *A* cell $[tt']$ *is a term for any terms* t, t'.

Nock terms are essentially S-expressions over numeric atoms. For $n > 2$, we use the notation $[t_1 t_2 \ldots t_n]$ to mean $[t_1[t_2[\ldots[t_{n-1}t_n]\ldots]]]$. In original Nock terminology [nocb], terms are called "nouns".

In Nockma (Anoma extension of Nock) the atoms are elements of integer rings ($\mathbb{Z}_n$) or finite fields ($\mathbb{F}_n$).

Definition 2. *A* position ρ *is a sequence of letters* L, R. *The empty position is denoted by* ϵ.

The subterm *of* t *at position* ρ, *denoted* $t_{|\rho}$, *is defined as expected:* $t_{|\epsilon} = t$, $[t_1 t_2]_{|L\rho} = t_{1|\rho}$, $[t_1 t_2]_{|R\rho} = t_{2|\rho}$. *Similarly,* $t_1\{t_2\}_\rho$ *is defined as the term obtained from* t_1 *by replacing the subterm at position* ρ *with* t_2.

A position ρ *is encoded by a natural number* $\varphi(\rho)$ *as follows:*

- $\varphi(\epsilon) = 1$,
- $\varphi(\rho L) = 2 * \varphi(\rho)$,
- $\varphi(\rho R) = 2 * \varphi(\rho) + 1$.

In Nock, a position ρ is represented by its natural number encoding $\varphi(\rho)$. For readability, we use ρ and $\varphi(\rho)$ interchangeably.

Definition 3. *A* Nock program *is any* Nock *term.*

Combinators *are the numbers defined below. These are used as instruction opcodes in the* Nock *programs. We name them for readability.*

- @ = 0 *is the addressing combinator used to extract a subterm at a given position. In original* Nock *terminology,* @ *is called "slot".*
- Quote = 1 *is a quoting combinator – its quoted argument is not evaluated but returned directly. It can be also seen as analogous to the* K *combinator from Combinatory Logic [HS08] implementing the constant function.*
- Apply = 2 *is an application combinator – it evaluates its arguments and then evaluates the second result in the context of the first. It can also be seen as analogous to the* S *combinator from Combinatory Logic, but with the order of arguments reversed.*
- IsCell = 3 *checks if a term is a cell or an atom.*
- Inc = 4 *increments a natural number atom.*

- Eq = 5 *tests terms for equality.*
- If = 6 *is an if-combinator. In* Nock, 0 *denotes truth and* 1 *denotes falsity.*
- Seq = 7 *implements sequencing of operations. It can also be seen as analogous to the* B *combinator from Combinatory Logic, only with argument order reversed.*
- Push = 8 *implements "stack push",*
- Call = 9 *is a method call combinator,*
- Replace = 10 *is a replacement combinator – it replaces a subterm at a given position.*
- Hint = 11 *is a hint combinator – it provides a more efficient (possibly built-in) implementation for a given* Nock *program.*

In [nocb] the Nock evaluation function is defined by rewriting rules. We present an equivalent definition in the style of big-step operational semantics (see e.g. [NK14]).

Definition 4. *The relation $s * t \Rightarrow t'$ is defined by the rules in Figure 1. If $s * t \Rightarrow t'$ then t* evaluates to *t'* on *s. In the figure, t, s are always terms, a is an atom, n is a natural number, ρ is a position (encoded by a natural number), etc.*

One can check that $s * t \Rightarrow t'$ is equivalent to $*[st] = t'$, where $*[st]$ is the Nock interpreter function from [nocb] (assuming an innermost reduction strategy in [nocb]). In original Nock terminology, s is the "subject", t is the "formula" and t' is the "product".

Informally, the relation $s * t \Rightarrow t'$ can be understood as evaluating t with data (stack, environment) s, or as evaluating the application of t to s with result t'. The second interpretation reveals an analogy between Nock combinators Quote, Apply, Seq and combinators K, S, B from Combinatory Logic [HS08].

If for given t, s, no finite derivation of $s * t \Rightarrow t'$ can be constructed for any t', then the evaluation is undefined – it results in an error (or non-termination).

The evaluation relation is deterministic (by induction on derivations), i.e., if $s * t \Rightarrow t'$ and $s * t \Rightarrow t''$ then $t' = t''$. Hence, it defines a partial function from $\mathcal{T} \times \mathcal{T}$ to $\mathcal{T}$.

2.1. Remarks

- The combinators If, Seq, Push, Call, Replace, Hint are not strictly necessary – they can be defined using the other combinators. See [nocb].

$$\frac{s * [t_1 t_2] \Rightarrow t' \quad s * t_3 \Rightarrow t''}{s * [[t_1 t_2] t_3] \Rightarrow [t' t'']}$$

$$\frac{}{s * [@\rho] \Rightarrow s_{|\rho}} \quad \frac{}{s * [\text{Quote } t] \Rightarrow t} \quad \frac{s * t_1 \Rightarrow t_1' \quad s * t_2 \Rightarrow t_2' \quad t_1' * t_2' \Rightarrow t'}{s * [\text{Apply } t_1\ t_2] \Rightarrow t'}$$

$$\frac{s * t \Rightarrow [t_1' t_2']}{s * [\text{IsCell } t] \Rightarrow 0} \quad \frac{s * t \Rightarrow a}{s * [\text{IsCell } t] \Rightarrow 1} \quad \frac{s * t \Rightarrow n}{s * [\text{Inc } t] \Rightarrow n + 1}$$

$$\frac{s * t_1 \Rightarrow t \quad s * t_2 \Rightarrow t}{s * [\text{Eq } t_1\ t_2] \Rightarrow 0} \quad \frac{s * t_1 \Rightarrow t_1' \quad s * t_2 \Rightarrow t_2' \quad t_1' \neq t_2'}{s * [\text{Eq } t_1\ t_2] \Rightarrow 1}$$

$$\frac{s * t_0 \Rightarrow 0 \quad s * t_1 \Rightarrow t_1'}{s * [\text{If } t_0\ t_1\ t_2] \Rightarrow t_1'} \quad \frac{s * t_0 \Rightarrow 1 \quad s * t_2 \Rightarrow t_2'}{s * [\text{If } t_0\ t_1\ t_2] \Rightarrow t_2'}$$

$$\frac{s * t_1 \Rightarrow t_1' \quad t_1' * t_2 \Rightarrow t'}{s * [\text{Seq } t_1\ t_2] \Rightarrow t'} \quad \frac{s * t_1 \Rightarrow t_1' \quad [t_1' s] * t_2 \Rightarrow t'}{s * [\text{Push } t_1\ t_2] \Rightarrow t'} \quad \frac{s * t \Rightarrow t' \quad t' * t'_{|\rho} \Rightarrow t''}{s * [\text{Call } \rho\ t] \Rightarrow t''}$$

$$\frac{s * t_1 \Rightarrow t_1' \quad s * t_2 \Rightarrow t_2'}{s * [\text{Replace } [\rho t_1]\ t_2] \Rightarrow t_2'\{t_1'\}_\rho} \quad \frac{s * t_2 \Rightarrow t_2' \quad s * t_3 \Rightarrow t_3'}{s * [\text{Hint } [t_1 t_2]\ t_2] \Rightarrow t_3'} \quad \frac{s * t \Rightarrow t'}{s * [\text{Hint } a\ t] \Rightarrow t'}$$

Figure 1. Nock evaluation semantics

- In the rule for e.g. Inc, in the premise $s * t \Rightarrow n$ the term n is required to be a numeric atom. If the evaluation of t on s yields a cell, then the rule cannot be applied and the evaluation of $[\text{Inc } t]$ on s is undefined. Similar remarks apply to all other rules, with t, s being arbitrary terms, a an atom, ρ a numeric representation of a position, etc.

- In the rule for @, if ρ is not a valid position of s, then the rule cannot be applied and the evaluation of $[@\rho]$ on s is undefined. Similarly with the rules for Replace and Call.

- In the first rule for the hint combinator Hint, the result t_2' of evaluating t_2 is not used. However, the presence of $s * t_2 \Rightarrow t_2'$ as a premise ensures that the evaluation of $[\text{Hint } [t_1 t_2]\ t_3]$ on s is defined only if the evaluation of t_2 on s is defined.

- In $[\text{Hint } t_1\ t_2]$, the term t_1 is a (more efficient) hint (a built-in if t_1 is an atom) that has the same semantics as t_2.

3. Compilation to Nock

In the Juvix compiler, the Nock code is generated from the JuvixTree intermediate representation [Cza24, Section 4.2]. The evaluation argument s contains a stack of locally let-bound values, the argument values, a function library pointer, etc. The generated Nock code performs "low-level" manipulations on s, extracting the required values, and updating the stack. This is feasible thanks to the addressing @ (slot) and replacement Replace combinators, which allow "random access" into s. Function calls are implemented with Call. Let-expressions are implemented with Push.

Acknowledgements

The present report is only a reformulation of the Nock specification using terminology from mainstream programming language theory. The generation of Nock code from JuvixTree mentioned in the last section was implemented by Paul Cadman and Jan Mas Rovira.

References

Cza24. Lukasz Czajka. Compiling Juvix to Cairo assembly. *Anoma Research Topics*, Sep 2024. doi:10.5281/zenodo.13739343. (cit. on p. 5.)

GYB23. Christopher Goes, Awa Sun Yin, and Adrian Brink. Anoma: a unified architecture for full-stack decentralised applications. *Anoma Research Topics*, Aug 2023. URL: https://doi.org/10.5281/zenodo.8279841, doi:10.5281/zenodo.8279842. (cit. on p. 1.)

Hel24. Heliax AG. Juvix Compiler, 2024. URL: https://github.com/anoma/juvix/. (cit. on p. 1.)

hoo. Hoon. https://docs.urbit.org/language/hoon/. (cit. on p. 1.)

HS08. J. R. Hindley and J. P. Seldin. *Lambda-Calculus and Combinators: An Introduction*. Cambridge University Press, 2 edition, 2008. (cit. on pp. 2 and 3.)

NK14. T. Nipkow and G. Klein. *Concrete Semantics with Isabelle/HOL*. Springer, 2014. (cit. on p. 3.)

noca. Nock. https://docs.urbit.org/language/nock/. (cit. on p. 1.)

nocb. Nock Definition. https://docs.urbit.org/language/nock/reference/definition. (cit. on pp. 1, 2, and 3.)

urb. Urbit. https://docs.urbit.org/. (cit. on p. 1.)