Arithmetization of Functional Program Execution via Interaction Nets in Halo 2

Anthony Hart

Orbis Labs
team@orbislabs.com

September 12th, 2022

Abstract

We sketch a method for creating a zero-knowledge proof of knowledge for the correct execution of a program within a model of higher-order, recursive, purely functional programming by leveraging Halo 2. To our knowledge, this is the first ZKP for general purpose computation based on purely functional computation. This is an attractive alternative to using a von Neumann architecture based zero-knowledge virtual machine for verified computing of functional programs, as compilation will be more direct, making it more easily verifiable and potentially more efficient.

Interaction nets are a natural setting for recursive, higher-order functional programming where all computation steps are linear and local. Interaction nets are graphs and traces for such programs are hyper-graphs. Correctness of a trace is a simple syntactic check over the structure of the trace represented as a hyper-graph. We reformulate this syntactic check as a Halo 2 circuit which is universal over all traces.

Contents

1 Introduction to Interaction Nets

Interaction nets are a model for computation introduced in [Laf89]. They are a setting for higher-order functional programming comparable to systems like the lambda calculus. They can be considered as patching some shortcomings of the lambda calculus. Consider the following pseudocode for evaluating lambda expressions

$$\begin{aligned}
\text{spine}((XY), [A...]) &:= \text{spine}(X, [\text{spine}(Y, []), A...]) \\
\text{spine}(\lambda x.X, []) &:= \lambda x.\text{spine}(X, []) \\
\text{spine}(\lambda x.X, [A, B...]) &:= \text{spine}(X[A/x], [B...]) \\
\text{spine}(x, L) &:= \text{fold}(x, L) \\
\text{fold}(X, []) &:= X \\
\text{fold}(X, [A, B...]) &:= \text{fold}((XA), [B...])
\end{aligned}$$

spine will attempt to normalize a given lambda expression by splitting off elements from its spine into a list and using those elements to eliminate lambda binders when possible. At the end, when it reaches a

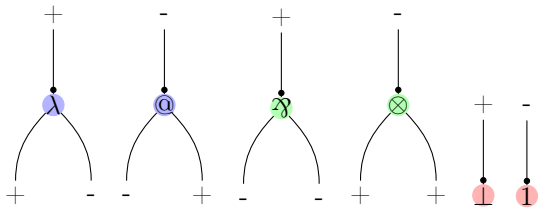

Figure 1: The six combinators of the Symmetric Interaction Combinators. Principal ports are marked with a black dot. In order, these combinators correspond to lambda binding, function application, duplication for both polarities, and deletion for both polarities.

variable, it builds that spine back up. Notice the first case of spine, where spine is called on the argument being put into the list. This call makes this an *eager* evaluator. It will attempt to normalize everything appearing in the expression. Even if this expression will later be deleted, this work will still be done. This also implies that some expressions which should have normal forms, like $(\lambda y.\lambda x.x)((\lambda x.xx)(\lambda x.xx))$, will not normalize using this program. Consider the following evaluator instead;

$$\begin{aligned}
\text{spine}((XY), [A...]) &:= \text{spine}(X, [Y, A...]) \\
\text{spine}(\lambda x.X, []) &:= \lambda x.\text{spine}(X, []) \\
\text{spine}(\lambda x.X, [A, B...]) &:= \text{spine}(X[A/x], [B...]) \\
\text{spine}(x, L) &:= \text{fold}(x, L) \\
\text{fold}(X, []) &:= X \\
\text{fold}(X, [A, B...]) &:= \text{fold}((X\ \text{spine}(A, [])), [B...])
\end{aligned}$$

Now entries put into the spine list will be left untouched, only being normalized when reconstructing the spine as part of the call to fold. This implementation will only do work which is necessary for normalization, making this a *lazy* evaluator. But this has its own shortcomings. If an expression is copied into multiple locations, it will be evaluated as many times as it's copied. This means that, while all the work it does is necessary, it may do the same work multiple times.

The conventional compromise to this problem is to implement a sharing mechanism. To be brief, whenever an expression is to be substituted into a lambda expression, it'd instead be stored onto a heap and the address of that expression would be substituted instead. Whenever an address needs to be evaluated, that address would be looked up in the heap, evaluated to head normal form, sub-expressions would be given their own addresses, and the head with sub expression addresses would be copied to wherever the original address was referenced. This prevents both doing unnecessary work and duplicating work. However, there's still much ambiguity in terms of how one should implement this optimally. One may view interaction nets as a model of computation where the sharing mechanism is integrated into its computational model.

Interaction nets can also be thought of as a graphical generalization to (arboreal) combinator calculi, such as the SKI calculus. As such, nodes and, synecdochically, the model of interaction nets itself are sometimes referred to as "interaction combinators". Like combinator calculi, there are different models of interaction nets with different sets of combinators. Each combinator in an interaction net model will be a node in a graph with some number of positively or negatively "polarized" ports. Each combinator will also have a special port, called its "principal" port. An interaction net itself will then be a graph with combinators acting as nodes such that each positive port has a unique link to a negative port. One of the most basic examples of such a calculus are the Symmetric Interaction Combinators [Maz07]. This system has six combinators, depicted in figure 1. Some presentations include an additional "root" node which allows one to complete programs with a single dangling positive port.

Each interaction calculus comes equipped with interaction rules which replace two nodes connected by their principal ports with some new sub-graph. The Symmetric Interaction Combinators has nine interaction rules, some of which are depicted in figure 2.

Combinators with positive principal ports can be viewed as being constructors and combinators with negative principal ports as being consumers. In order for evaluation to take place, a consumer must be connected to a constructor; that is, a consumer must be connected to an expression which is at least in head normal form, and all computation, such as copying or deleting, happens at these heads. It is in this sense that interaction nets have an integrated sharing mechanism, as described earlier in this section. This idea is at the root of the optimal evaluation of functional programming languages [AG98].

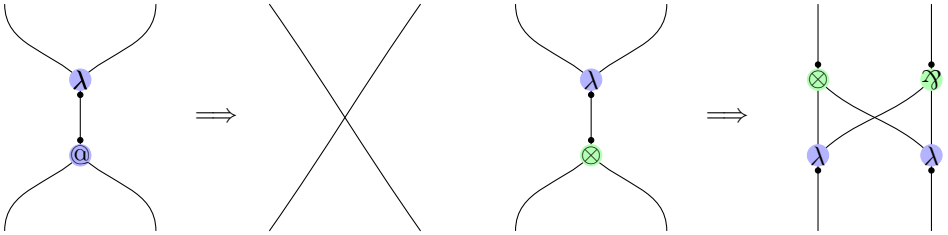

Figure 2: Two of the nine interaction rules for the Symmetric Interaction Combinators. The first performs linear β-reduction when an application is applied to a lambda binder. The second copies a lambda binder.

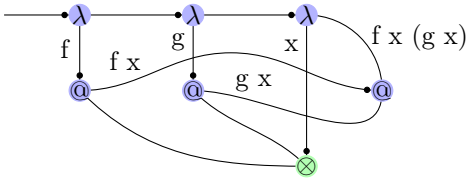

Figure 3: The S combinator, $\lambda f.\lambda g.\lambda x.fx(gx)$, translated into the Symmetric Interaction Combinators. The letters are there to aid in understanding the translation and aren't part of the net.

As a programming setting, interaction combinators are very similar to the lambda calculus. Non-principal positive ports can be thought of as variable binders while non-principal negative ports are ancillary arguments. There is a more or less direct way to translate lambda calculus expressions into Symmetric Interaction Combinator nets. An example of this is given in figure 3. As a consequence, powerful programming techniques available in that setting, such as fixed-point combinators and lambda encoding, still exist in this new setting.

Interaction combinators have some other attractive features. Consider the evaluation step

$$(\lambda x.\lambda f.fxx)T \mapsto \lambda f.fTT$$

How much work does this step take? Well, T could be arbitrarily large, so this step could take an arbitrarily large amount of work to copy T, despite being only a single step. This makes resources hard to deal with in the vanilla lambda calculus. Interaction combinators, on the other hand, don't have this problem. There are only a finite number of replacement rules, each replacing a fixed finite sub-graph with another fixed finite sub-graph. Evaluation for interaction combinators is *linear*. It is this property which makes interaction combinators a good candidate for arithmetization.

Traces can be formulated to only have a fixed, finite width. They can be seen as graphs themselves, with each rule acting as a node. Checking the correctness of these traces then becomes a context-free check on the structure of the graph. The goal of the rest of this paper is to describe this in detail and arithmetize the process in halo2 [Com21] to demonstrate practicality. This will provide an arithmetizable setting for general purpose, higher-order functional programming. Contrast this with projects like TinyRAM [CGTV20] which require the programmer to use a form of assembly code which will be more onerous for most applications.

2 Defining Interaction Nets and Traces

Interaction nets require an existing specification for the types of nodes which may appear in the graph.

***Definition* 1** *Node Context* A node context consists of

1. A finite set, $\mathcal{T}$, of node types.
2. A function $\mathcal{P}^+ : \mathcal{T} \to \mathbb{N}$ assigning a number of positive ports to each type.
3. A function $\mathcal{P}^- : \mathcal{T} \to \mathbb{N}$ assigning a number of negative ports to each type.

***Definition* 2** *Interaction Net* Given a node context $(\mathcal{T}, \mathcal{P}^+, \mathcal{P}^-)$, an interaction net over that specification consists of

1. A finite set, N, of nodes.
2. A function $\tau : \mathrm{N} \to \mathcal{T}$ assigning a type to each node.
3. The set $P^+ := \{(n, m) \in \mathrm{N} \times \mathbb{N} \mid m < \mathcal{P}^+(\tau(n))\}$ of positive ports.
4. The set $P^- := \{(n, m) \in \mathrm{N} \times \mathbb{N} \mid m < \mathcal{P}^-(\tau(n))\}$ of negative ports.
5. A bijection $w : P^+ \to P^-$ wiring each positive port onto a negative port.

The actual interaction rules specifying a step in a computation are a bit more complicated.

***Definition* 3** *Interaction Rule* Given a node context $(\mathcal{T}, \mathcal{P}^+, \mathcal{P}^-)$, an interaction rule consists of

1. Two elements I^+ and I^- taken from $\mathcal{T}$. These will be the types of the two nodes at the input to the rule. I^+ will be connected from its (principal) positive 0 port to I^-'s negative (principal) 0 port. Principal ports are not explicitly marked, but they may be inferred by their presence within an interaction rule.
2. The set $B^+ := \{n \mid 0 < n < \mathcal{P}^+(\mathrm{I}^+)\} + \{n \in \mathbb{N} \mid n < \mathcal{P}^+(\mathrm{I}^-)\}$ of positive boundary ports where $+$ denotes a disjoint union.
3. The set $B^- := \{n \mid 0 < n < \mathcal{P}^-(\mathrm{I}^-)\} + \{n \in \mathbb{N} \mid n < \mathcal{P}^-(\mathrm{I}^+)\}$ of negative boundary ports.
4. A finite set, O, of output nodes.
5. A function $o : \mathrm{O} \to \mathcal{T}$ assigning each output node a type.
6. The set $O^+ := \{(n, m) \in \mathrm{O} \times \mathbb{N} \mid m < \mathcal{P}^+(o(n))\}$ of positive output node ports.
7. The set $O^- := \{(n, m) \in \mathrm{O} \times \mathbb{N} \mid m < \mathcal{P}^-(o(n))\}$ of negative output node ports.
8. A bijection $i : B^- + O^+ \to B^+ + O^-$ wiring each positive (output or boundary) port onto a negative (output or boundary) port.

Notice that negative boundary ports may be wired to negative output ports by i, and similarly for positive boundary ports and positive output ports. This is because the boundary ports correspond to the ports facing outward on the input, and such ports are wired to ports facing the same direction on the output.

Each trace requires a background setting dictating what interactions are available.

***Definition* 4** *Trace Setting* Given a node context $(\mathcal{T}, \mathcal{P}^+, \mathcal{P}^-)$, a trace setting consists of

1. A finite set $\mathcal{R}$ of interaction rule names.
2. A map $\mathcal{D}$ mapping each $t \in \mathcal{R}$ to an interaction rule, $(\mathrm{I}^+_{\mathcal{D}(t)}, \mathrm{I}^-_{\mathcal{D}(t)}, \mathrm{O}_{\mathcal{D}(t)}, o_{\mathcal{D}(t)}, i_{\mathcal{D}(t)})$, over $(\mathcal{T}, \mathcal{P}^+, \mathcal{P}^-)$.

Finally, We will define an interaction trace.

***Definition* 5** *Interaction Trace* Given a trace setting $(\mathcal{R}, \mathcal{D})$ over the node context $(\mathcal{T}, \mathcal{P}^+, \mathcal{P}^-)$, an interaction trace consists of

1. An interaction net $(\mathrm{N}_i, \tau_i, w_i)$ specifying the starting state.
2. An interaction net $(\mathrm{N}_o, \tau_o, w_o)$ specifying the ending state.
3. A $\mathbb{n} \in \mathbb{N}$ specifying the number of interactions within the trace.
4. The set $\mathcal{S} := \{n \in \mathbb{N} \mid n < \mathbb{n}\}$ of interaction steps. This way of defining $\mathcal{S}$ allows one to enforce a canonical order.
5. A map $\mathrm{T} : \mathcal{S} \to \mathcal{R}$ assigning an interaction rule to each step.
6. For each $t \in \mathcal{T}$, the set $N^+_t := \{n \in \mathrm{N}_i \mid \tau_i(n) = t\} + \{(s, n) \in \mathcal{S} \times \mathrm{O}_{\mathcal{D}(\mathrm{T}(s))} \mid \mathrm{o}_{\mathcal{D}(\mathrm{T}(s))}(n) = t\}$ of positively polarized nodes.

7. For each $t \in \mathcal{T}$, the set $N_t^- := \{n \in \mathrm{N}_o \mid \tau_o(n) = t\} + \{s \in \mathcal{S} \mid \mathrm{I}^+_{\mathcal{D}(\mathrm{T}(s))} = t\} + \{s \in \mathcal{S} \mid \mathrm{I}^-_{\mathcal{D}(\mathrm{T}(s))} = t\}$ of negatively polarized nodes.

8. For each $t \in \mathcal{T}$, a bijection $b_t : N_t^+ \to N_t^-$.

9. A condition, to be stated at the end of this section, called "temporal consistency".

Before defining temporal consistency, a few things need to be noted.
Firstly, establish a canonical number to all the nodes. Define

$$\mathrm{level}(n) := \begin{cases} 0 & \text{if } n \in \mathrm{N}_i \\ \mathbb{n} + 1 & \text{if } n \in \mathrm{N}_o \\ i + 1 & \text{if } n \text{ is a node of interaction step } i \end{cases}$$

Extend this to a function over ports (which we'll also call "level") that assigns to each port the level of the node it belongs to.

The trace definition (ignoring temporal consistency) makes references to nodes, but not their ports. However, each node within the trace has a collection of ports. These ports are connected together in a structured way through the various bijections in the definition.

The maps b_t collectively define a bijection between all the positive and negative nodes. This follows from the observation that each interaction has only one type, and so the domain and codomain of the different bs don't overlap. This bijection can be lifted into a bijection between all the ports on the graph. Specifically, it defines a bijection between ports of the same polarization, but going from positive nodes to negative nodes of the same type.

Additionally, each interaction, as well as the input and output nets, contains a bijection among its ports through its respective wiring. Since interaction steps do not overlap on their nodes or ports, these collectively define a bijection over all the ports on the graph.

Define the set of positive ports of a trace to be the following:

$$\begin{aligned} p^+ := &\{(n, m) \in \mathrm{N}_i \times \mathbb{N} \mid m < \mathcal{P}^+(n)\} \\ &+ \{(n, m) \in \mathrm{N}_o \times \mathbb{N} \mid m < \mathcal{P}^-(n)\} \\ &+ \{((s, o), m) \in (\mathcal{S} \times \mathrm{O}_{\mathcal{D}(\mathrm{T}(s))}) \times \mathbb{N} \mid m < \mathcal{P}^+(o)\} \\ &+ \{(s, m) \in \mathcal{S} \times \mathbb{N} \mid m < \mathcal{P}^-(\mathrm{I}^+_{\mathcal{D}(\mathrm{T}(s))})\} \\ &+ \{(s, m) \in \mathcal{S} \times \mathbb{N} \mid m < \mathcal{P}^-(\mathrm{I}^-_{\mathcal{D}(\mathrm{T}(s))})\} \end{aligned}$$

Notice that the ports on negative nodes swap polarity. So the positive ports consist of all the positive ports on the positive nodes plus the negative ports on the negative nodes. Define the negative ports of a trace similarly:

$$\begin{aligned} p^- := &\{(n, m) \in \mathrm{N}_i \times \mathbb{N} \mid m < \mathcal{P}^-(n)\} \\ &+ \{(n, m) \in \mathrm{N}_o \times \mathbb{N} \mid m < \mathcal{P}^+(n)\} \\ &+ \{((s, o), m) \in (\mathcal{S} \times \mathrm{O}_{\mathcal{D}(\mathrm{T}(s))}) \times \mathbb{N} \mid m < \mathcal{P}^-(o)\} \\ &+ \{(s, m) \in \mathcal{S} \times \mathbb{N} \mid m < \mathcal{P}^+(\mathrm{I}^+_{\mathcal{D}(\mathrm{T}(s))})\} \\ &+ \{(s, m) \in \mathcal{S} \times \mathbb{N} \mid m < \mathcal{P}^+(\mathrm{I}^-_{\mathcal{D}(\mathrm{T}(s))})\} \end{aligned}$$

The negative ports consist of all the negative ports on the positive nodes plus the positive ports on the negative nodes.

Observe that both the previously mentioned bijections are from p^+ to p^-. Define the bijection generated by the bs to be

$$b^* := \{((i, a, n), (j, b, m)) \in p^+ \times p^- \mid n = m \wedge \exists t, b_t(a) = b \vee b_t(b) = a\}$$

There are also certain special links. Each interaction rule reserves the 0th positive port of I^+ and the 0th negative port of I^-. This link is part of the wiring of that interaction step. This will be called the principal link between principal ports. Define the bijection generated by the rules and starting and ending nets to be

$$r^* := w_i \cup w_o^{\cup} \cup \bigcup_{s \in \mathcal{S}} i_{\mathcal{D}(\mathrm{T}(s))} \cup \{(a,b) \in p^+ \times p^- \mid \exists s \in \mathcal{S}, a = (4,s,0) \ \wedge \ b = (3,s,0) \ \}$$

That last expression in the union chain declares the links between the principal ports. Here, 4 and 3 refer to the index within the respective coproducts defining p^- and p^+.

By composing the two bijections together, a bijection from either p^+ or p^- back onto itself is formed. Either way, this defines a permutation among the ports. This permutation decomposes into cycles of ports. These cycles form the boundary of the surfaces within the trace graph. Call the cycle containing some port p the "cycle of p".

One may recast cycles in two different ways. Cycles can be viewed as a permutation over r^*, that is, over pairs of ports linked by interactions. Define this permutation as

$$r^\circ := \{((a,b),(x,y)) \in r^* \times r^* \mid b^*(a) = y\}$$

Just as easily, cycles can be viewed as a permutation over b^*, over pairs of ports linked by the b_t maps. Define this permutation as

$$b^\circ := \{((a,b),(x,y)) \in b^* \times b^* \mid r^*(a) = y\}$$

This latter permutation will be most important for understanding the correctness proof in the next section.

We am now in a position to define temporal consistency. It is the following two requirements;

1. For all port cycles, there is at most one occurrence of a principal link, and the principal ports of this link have a level greater than or equal to the level of every other port within said cycle. In other words, no part of a cycle can appear after the principal link of that cycle within a trace.
2. A port cycle does not have a principal link if and only if it has exactly one occurrence of a link from the ending state.

From these, obtain the following important corollary

***Corollary* 1** *Monotonicity of Trace Wirings*

$\forall t \in \mathcal{R}, \forall n \in N_t^+, \mathrm{level}(n) < \mathrm{level}(b_t(n))$. That is, the node wiring within the trace must be strictly increasing with respect to node levels.

Proof. To see this, imagine a scenario where a positive node is connected to a negative node of a lower level. Such a scenario is depicted in figure 4. As explained in that figure's caption, this creates a port cycle which violates condition 1. of temporal consistency.

3 Correctness of Trace Representation

It's not obvious that the definition of trace presented here is correct. This section is dedicated to presenting a collection of theorems which clearly verify that we've represented a trace. Specifically, we will prove that a list of reasonable intermediate states between the starting and ending state can be extracted from our representation.

We need the following lemma.

***Lemma* 1** *Knitting Lemma* Given the finite sets O^+, O^-, S^+, S^-, I^+, and I^- and bijections $b_1 : O^+ + S^+ \to O^- + S^-$ and $b_2 : I^+ + S^- \to I^- + S^+$ the following hold;

1. $\forall x \in O^+, \exists n, (b_1 \circ b_2)^n(b_1(x)) \in O^- \vee (b_2 \circ b_1)^n(x) \in I^-$
2. $\forall x \in I^+, \exists n, (b_2 \circ b_1)^n(b_2(x)) \in I^- \vee (b_1 \circ b_2)^n(x) \in O^-$

Note that this statement is abusing notation to coerce an $A + B$ into either an A or a B where appropriate.

Proof. 1. By induction on $|S^+| + |S^-|$.

1. Assume $|S^+| + |S^-| = 0$. b_1 will be a bijection between O^+ and O^-. This allows us to conclude that $(b_1 \circ b_2)^0(b_1(x)) \in O^-$.

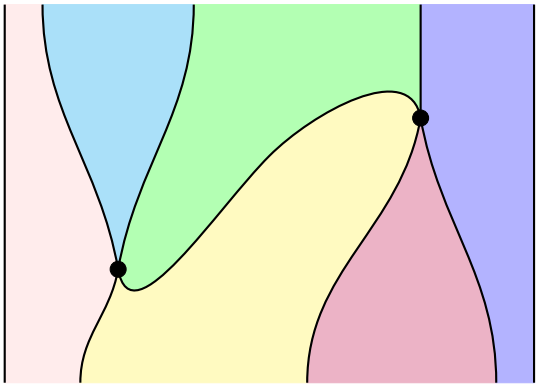

Figure 4: An impossible slice of a trace graph. The dots are interaction steps, lines are links between nodes, and colored surfaces are bound by port cycles. The level increases from bottom to top. This slice depicts a hypothetical scenario where a positive node (the left positive node of the right interaction here) is connected to a negative node (the right negative node of the left interaction here) such that the level of the negative node is less than the level of the positive node. One may notice that the port cycle bounding the yellow region will have an occurrence on the right interaction at a higher level than its principal port in the left interaction. This is impossible by temporal consistency.

2. For the inductive cases consider three mutually exclusive, exhaustive cases;

   (a) For each quadruple $(x, y, z, w) \in O^+ \times S^- \times S^+ \times (O^- + S^-)$ such that (x, y) and (z, w) are in the graph of b_1 and (y, z) is in the graph of b_2, define bijections

   $$b_1' : O^+ + \{e \in S^+ \mid e \neq z\} \to O^- + \{e \in S^- \mid e \neq y\} := (b_1 \backslash \{(x, y), (z, w)\}) \cup \{(x, w)\}$$

   $$b_2' : I^+ + \{e \in S^- \mid e \neq y\} \to I^- + \{e \in S^+ \mid e \neq z\} := b_2 \backslash \{(y, z)\}$$

   Observing that

   $$|\{e \in S^+ \mid e \neq z\}| + |\{e \in S^- \mid e \neq y\}| < |S^+| + |S^-|$$

   use the inductive hypothesis to conclude the existence of a natural number n such that $(b_1' \circ b_2')^n(b_1'(x)) \in O^- \vee (b_2' \circ b_1')^n(x) \in I^-$. From the observation that $b_1(b_2(b_1(x))) = b_1'(x)$ and otherwise that $\forall y, b_1(y) = b_1'(y)$ and $\forall y, b_2(y) = b_2'(y)$, conclude that $(b_1 \circ b_2)^{n+1}(b_1(x)) \in O^- \vee (b_2 \circ b_1)^{n+1}(x) \in I^-$.

   (b) For each triple $(x, y, z) \in O^+ \times S^- \times I^-$ such that (x, y) is in the graph of b_1 and (y, z) is in the graph of b_2, conclude that $(b_2 \circ b_1)^1(x) \in I^-$.

   (c) For each pair $(x, y) \in O^+ \times O^-$ in the graph of b_1, conclude that $(b_1 \circ b_2)^0(b_1(x))$.

2. follows from 1.; simply swap O and I, swap S^+ and S^-, and swap b_1 and b_2 in the premises. **QED.**

A corollary of Lemma 1 is that the existence of the bijections $O^+ + S^+ \to O^- + S^-$ and $I^+ + S^- \to I^- + S^+$ imply the existence of a bijection $O^+ + I^+ \to O^- + I^-$.

***Definition* 6** *Interaction Patch* Given a node context $(\mathcal{T}, \mathcal{P}^+, \mathcal{P}^-)$, an Interaction Patch consists of the following data

1. B^+, a finite set of positive boundary ports.
2. B^-, a finite set of negative boundary ports.
3. N, a finite set of nodes.
4. $\tau : N \to \mathcal{T}$, a function assigning each node a type.
5. The set $N^+ := \{(n, m) \in N \times \mathbb{N} \mid m < \mathcal{P}^+(\tau(n))\}$ of positive node ports.
6. The set $N^- := \{(n, m) \in \mathrm{O} \times \mathbb{N} \mid m < \mathcal{P}^-(\tau(n))\}$ of negative node ports.
7. $w : N^+ + B^+ \to N^- + B^-$, a bijection wiring negative ports to positive ports.

An interaction patch is essentially an incomplete piece of an interaction net. Every interaction rule defines two interaction patches; one on the input and the other on the output. Given an interaction rule $R = (\mathrm{I}^+, \mathrm{I}^-, \mathrm{O}, B^+, B^-, o, i)$, the *input patch* is the interaction patch defined by $\underline{R} = (B^-, B^+, \{\mathrm{I}^+, \mathrm{I}^-\}, \lambda x.x, f)$, where f wires each boundary port to itself and wires the 0th positive port of I^+ to the 0th negative port of I^-. For the same rule, the *output patch* is the interaction patch defined by $\overline{R} = (B^-, B^+, \mathrm{O}, o, i)$.

Also note the fact that the Knitting Lemma 1 can be used to stitch two patches together. Given patches $p = (B^+, B^-, N_p, \tau_p, w_p)$ and $q = (B^-, B^+, N_q, \tau_q, w_q)$ over the same node context, note that $w_p : N_p^+ + B^+ \to N_p^- + B^-$ and $w_q : N_q^+ + B^- \to N_q^- + B^+$. By the knitting lemma, a new bijection $w_{pq} : N_p^+ + N_q^+ \to N_p^- + N_q^-$ is defined. Then define the interaction net $(N_p + N_q, \tau_p + \tau_q, w_{pq})$. Denote this by $p \bowtie q$.

Also note that an interaction net can, itself, be viewed as an interaction patch with no boundary ports.

***Definition* 7** *Patch Homomorphism* Given two interaction patches

$$(B_1^+, B_1^-, \mathrm{N}_1, \tau_1, N_1^+, N_1^-, w_1) \text{ and } (B_2^+, B_2^-, \mathrm{N}_2, \tau_2, N_2^+, N_2^-, w_2)$$

over the same node context $(\mathcal{T}, \mathcal{P}^+, \mathcal{P}^-)$, an interaction patch homomorphism consists of

1. For all $t \in \mathcal{T}$ an injection $i_t : \{n \in \mathrm{N}_1 \mid \tau_1(n) = t\} \to \{n \in \mathrm{N}_2 \mid \tau_2(n) = t\}$
2. The injections, $i_t^+ : N_1^+ \to N_2^+$ defined by $(n, m) \mapsto (i_t(n), m)$
3. The injections, $i_t^- : N_1^- \to N_2^-$ defined by $(n, m) \mapsto (i_t(n), m)$
4. An injection $j^+ : B_1^+ \to N_2^+ + B_2^+$.
5. An injection $j^- : B_1^- \to N_2^- + B_2^-$.
6. Such that $\forall n \in N_1^+ + B_1^+, [i_t^-, j^-](w_1(n)) = w_2([i_t^+, j^+](n))$, for the type appropriate i_t, where $[f, g] : A + B \to C$ is the universal property of the coproduct on $f : A \to C$ and $g : B \to C$.

Using this, define a category of patches with patch homomorphisms as morphisms. Given the patch homomorphisms, one can compose them by composing their defining functions, and there is an obvious definition for reflexive morphisms. This leads, in particular, to a notion of *patch isomorphism*.

***Definition* 8** *Patch Isomorphism* Given two interaction patches,

$$(B_1^+, B_1^-, \mathrm{N}_1, \tau_1, N_1^+, N_1^-, w_1) \text{ and } (B_2^+, B_2^-, \mathrm{N}_2, \tau_2, N_2^+, N_2^-, w_2)$$

over the same node context $(\mathcal{T}, \mathcal{P}^+, \mathcal{P}^-)$, a patch isomorphism between them consists of

1. For all $t \in \mathcal{T}$ a bijection $b_t : \{n \in \mathrm{N}_1 \mid \tau_1(n) = t\} \to \{n \in \mathrm{N}_2 \mid \tau_2(n) = t\}$.
2. The bijections, $b_t^+ : N_1^+ \to N_2^+$ defined by $(n, m) \mapsto (b_t(n), m)$
3. The bijections, $b_t^- : N_1^- \to N_2^-$ defined by $(n, m) \mapsto (b_t(n), m)$
4. A bijection $j^+ : B_1^+ \to B_2^+$.
5. A bijection $j^- : B_1^- \to B_2^-$.
6. Such that $\forall p \in N_1^+ + B_1^+, [b_t^-, j^-](w_1(p)) = w_2([b_t^+, j^+](p))$, for the type appropriate b_t.

Patch isomorphisms act as the isomorphisms in our aforementioned category of patches. Restricting to patches with no boundary ports, obtain a notion of *interaction net isomorphism*.

***Definition* 9** *Patch Complement* Given the interaction patch,

$$(B_1^+, B_1^-, \mathrm{N}_1, \tau_1, N_1^+, N_1^-, w_1)$$

and interaction net

$$(\mathrm{N}_2, \tau_2, N_2^+, N_2^-, w_2)$$

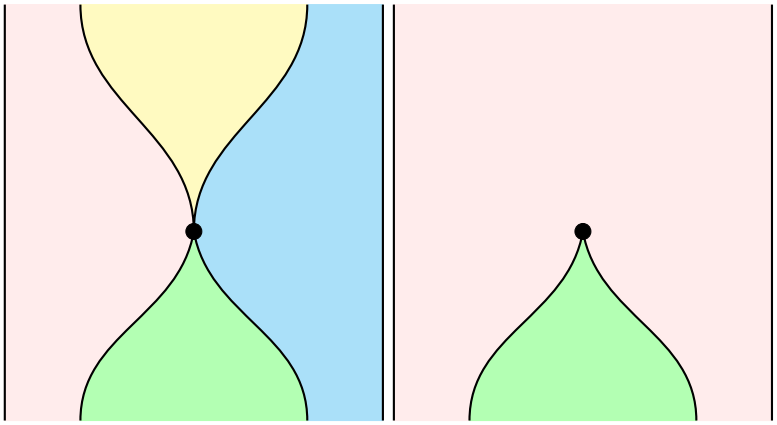

Figure 5: Two possible slices of a valid trace graph. The center dot is an interaction, lines are links between nodes, and colored surfaces are bound by port cycles. The level increases from bottom to top. The left graph demonstrates that interactions eliminate and can create surfaces/port cycles. The right demonstrates that port cycles can increase then decrease in level at an interaction.

over the same node context $(\mathcal{T}, \mathcal{P}^+, \mathcal{P}^-)$ and a patch homomorphism

$$h = (\{i_t\}_{t\in\mathcal{T}}, \{i_t^+\}_{t\in\mathcal{T}}, \{i_t^-\}_{t\in\mathcal{T}}, j^+, j^-)$$

the patch complement, denoted $x\backslash h$, is the interaction patch defined by

1. $B^+ := B^-$
2. $B^- := B^+$
3. $N := \mathrm{N}_2 \backslash \bigcup_{t\in\mathcal{T}} \mathrm{Im}(i_t)$
4. $\tau := \{(n,t) \in \tau_2 \mid \forall t \in \mathcal{T}, n \notin \mathrm{Im}(i_t)\}$
5. $N^+ := \mathrm{N}_2^+ \backslash \bigcup_{t\in\mathcal{T}} \mathrm{Im}(i_t^+)$
6. $N^- := \mathrm{N}_2^- \backslash \bigcup_{t\in\mathcal{T}} \mathrm{Im}(i_t^-)$
7. $$\begin{array}{rl} w: & \mathrm{N}_2^+ \backslash \bigcup_{t\in\mathcal{T}} \mathrm{Im}(i_t^+) + B^- \to \mathrm{N}_2^- \backslash \bigcup_{t\in\mathcal{T}} \mathrm{Im}(i_t^-) + B^+ \\ := & (w_2 \backslash \bigcup_{t\in\mathcal{T}} \{(a,b) \mid a \in \mathrm{Im}([i_t^+, j^+]) \vee b \in \mathrm{Im}([i_t^-, j^-])\}) \ \cup j^- \cup (j^+)^\cup \end{array}$$

This allows us to create a hole within a net using a homomorphism. Note that stitching a patch with a complement generated with a homomorphism out of that same patch yields the original net the complement was taken from. That is, for any net n and homomorphism from the patch p onto n, h, n is isomorphic to $p \bowtie (n\backslash h)$.

For any trace, one can take a slice out of the trace at any point between any two interactions. This gives, for all $i \in \{0, ..., \mathbb{n}\}$, the following set of slice node links;

$$L_i := \{(t, a, b) \in \mathcal{R} \times N_t^+ \times N_t^- \mid b_t(a) = b \wedge \mathrm{lvl}(a) \leq i \wedge \mathrm{lvl}(b) > i\}$$

further identify the set of port links determined by a slice. Define the following notations:

$$L_i^+ := \{(p, n) \in L_i \times \mathbb{N} \mid n < \mathcal{P}^+(\pi_1(p))\}$$
$$L_i^- := \{(p, n) \in L_i \times \mathbb{N} \mid n < \mathcal{P}^-(\pi_1(p))\}$$

denoting the positive and negative ports associated to the nodes within L_i. Note that the ports on any L_i will simply be the ports associated to the type of the nodes on either end. These ports are exactly the elements in b^* restricted to layer i. For any $((t, a, b), n)$ in either L_i^+ or L_i^-, $\mathrm{lvl}((t, a, b), n) = (\mathrm{lvl}(a), \mathrm{lvl}(b))$.

With this in hand, prove the following lemma:

Lemma 2 *Slice Wiring* For all k, Given a pair $(a, b) \in L_k^+$, there exists a pair $(x, y) \in L_k^-$ such that there exists a $n \in \mathbb{N}$ such that $(b^\circ)^n(a, b) = (x, y)$ and, for all $0 < m < n$, there exists some $l < k$ such that $(b^\circ)^m(a, b) \in L_l^+ \cup L_l^-$. That is, for any link in L_k^+, one can follow b° through lower Ls until a L_k^- is reached.

Proof. By induction on the level.

Start with T_0 and a pair $(a, b) \in L_0^+$. Note that $\text{lvl}(a) \leq 0$ implies that a is a port on the starting net. This implies that $r^*(a) = w_i(a)$ will be another port of opposite polarity, also on the starting net. This implies that $b^\circ((a, b))$ will be another pair with its second element, $r^*(a)$, having a level less than or equal to 0. The first element of this pair, $b_t(r^*(a))$, must satisfy $\text{lvl}(b_t(r^*(a))) > 0$, which holds as a direct consequence of the monotonicity of trace wiring. This gives a pair $b^\circ((a, b)) = (b_t(r^*(a)), r^*(a)) \in L_0^-$. So, $n = 1$ for the base case. Furthermore, note that b° produces an isomorphic wiring to w_i, in this case.

If we am at a T_{k+1} then we need to determine an element of L_{k+1}^- for each L_{k+1}^+. There are two main scenarios to start with. For any $(a, b) \in L_{k+1}^+$, we know that $\text{lvl}(a) \leq k + 1$.

1. If $\text{lvl}(a) = k + 1$, then it is an output port of interaction k. Follow the output wiring of k to get either a negative port within the output of the same level (analogous to following the boundary of the yellow surface in the left graph in figure 5), or it may produce a boundary port (analogous to following the inner boundary of either the blue or red surfaces *down* in the left graph in figure 5). In the case of an interior port we're done and our $n = 1$ since r^* will reproduce the output wiring of k in this case. In the case of a boundary port, we must traverse back down the trace in the hopes of returning to L_{k+1}. Put a pin in this.

2. If $\text{lvl}(a) < k + 1$ then $\text{lvl}(a) \leq k$ and so $(a, b) \in L_k^+$. In this case, we can use the inductive hypothesis to conclude that, after some m applications of b°, we will get another pair $(x, y) \in L_k^-$. We know that $\text{lvl}(y) > k$. If $\text{lvl}(y) > k + 1$, we've arrived in L_{k+1}^- and am done. If, however, $\text{lvl}(y) = k + 1$, then we will be arriving at a port on interaction k. A priori, this could be a principal port (analogous to traveling along the boundary of the green surface in either graph in figure 5) or a boundary port. However, we know that (a, b) is part of the same cycle as (x, y). Since b has a higher level than $k + 1$, condition 1. of temporal consistency prevents us from arriving at a principal port. Therefore, y must be a boundary port. This immediately implies the existence of a wiring,

   $$w'_k : \{(a, b) \ \in L_{k+1}^+ \mid \text{lvl}(a) < k + 1\} + B_k^+ \to \{(a, b) \ \in L_{k+1}^- \mid \text{lvl}(a) < k + 1\} + B_k^-$$

   from the inductive hypothesis, where B_k^+ and B_k^- are the boundary ports for interaction k.

   If we've arrived at a boundary port, we can then use the output wiring of interaction k to get either an interior port within the output at the correct level (analogous to following the inner boundary of either the blue or red surfaces *up* in the left graph in figure 5), or we will get a boundary port (analogous to following the interior boundary of the red surface in the right graph of figure 5). In the case of an interior port, we're done and our $n = m + 1$. In the case of a boundary port, we must traverse back down the trace in the hopes of returning to L_{k+1}.

If we end up traversing back down the graph from interaction k, nothing prevents us from returning to interaction k along another one of its boundary ports. We must prove this ends at some point. We may observe that the elements of (a, b) of either L_{k+1}^+ or L_{k+1}^- attached to interaction k are exactly those satisfying $\text{lvl}(a) = k + 1$. We can therefore recast the output wiring of interaction k as a bijection

$$i'_k : \{(a, b) \ \in L_{k+1}^+ \mid \text{lvl}(a) = k + 1\} + B_k^- \to \{(a, b) \ \in L_{k+1}^- \mid \text{lvl}(a) = k + 1\} + B_k^+$$

By the knitting lemma, we can turn w'_k and i'_k into a bijection

$$\begin{aligned} &\{(a, b) \ \in L_{k+1}^+ \mid \text{lvl}(a) < k + 1\} + \{(a, b) \ \in L_{k+1}^+ \mid \text{lvl}(a) = k + 1\} \\ &\to \{(a, b) \ \in L_{k+1}^- \mid \text{lvl}(a) < k + 1\} + \{(a, b) \ \in L_{k+1}^- \mid \text{lvl}(a) = k + 1\} \\ &\equiv \{(a, b) \ \in L_{k+1}^+ \mid \text{lvl}(a) \leq k + 1\} \to \{(a, b) \ \in L_{k+1}^- \mid \text{lvl}(a) \leq k + 1\} \\ &\equiv L_{k+1}^+ \to L_{k+1}^- \end{aligned}$$

Which is exactly the wiring we were looking for. By summing up the ns associated to each path through the trace used by the bijection defined by the knitting lemma, we get the expected n. **QED.**

And, finally, we can state and prove the full correctness theorem for interaction traces.

***Theorem* 1** *Trace Correctness* For any interaction trace

$$(A, B, \mathbb{n}, \mathcal{S}, \mathrm{T}, \{N_t^+\}_{t\in\mathcal{R}}, \{N_t^-\}_{t\in\mathcal{R}}, \{b_t\}_{t\in\mathcal{R}}).$$

for all $i \in \{0, ..., \mathbb{n}+1\}$, there exists an interaction net, X_i, such that:

1. The nodes of X_i are L_i.
2. X_0 is isomorphic to A.
3. For all $i \in \{0, ..., \mathbb{n}-1\}$, there is a homomorphism, h_i, from $\underline{\mathrm{T}(i)}$ into X_i.
4. For all $i \in \{0, ..., \mathbb{n}-1\}$, X_{i+1} is isomorphic to $\overline{\mathrm{T}(i)} \bowtie (X_i \backslash h_i)$. That is, X_i and X_{i+1} are the same everywhere except the patch altered by interaction rule i.
5. $X_{\mathbb{n}+1}$ is isomorphic to B.

This correctness condition essentially states that the trace representation can be automatically foliated into a sensible history of intermediate interaction nets such that the initial and final nets are what one expects.

Proof.

Using the slice wiring, obtain the links between the ports of L_i, and therefore obtain a wiring for each layer. The wire is essentially determined by following the permutation cycle down the trace until we pop up back in L_i. Lemma 2 is merely the assertion that this process always succeeds. And, furthermore, by following the cycle in the other direction, obtain the necessary condition that this wiring is a bijection.

From Lemma 2, establish all the conditions of the correctness criterion.

1. follows trivially from the base case of the slice wiring.

2. and 4. follow from essentially the same reasoning; since the wiring is determined by a partial port cycle, and temporal correctness guarantees that principal links and ending net links each complete a unique port cycle, get a one-to-one correspondence between them which determine the desired homomorphism and isomorphism, respectively. This was already demonstrated in the principal case as part of the inductive step.

3. follows from the observation that the call to the knitting lemma in the inductive step implies the same bijection used in the definition of patch stitching. That is, the inductive hypothesis essentially defines X_{k+1} to be $\overline{\mathrm{T}(k)} \bowtie (X_k \backslash h_k)$, making that condition trivial.

QED.

4 Halo 2

This section will briefly describe arithmetic circuits in Halo 2. The point of an arithmetic circuit is to define a relation over matrices of finite field elements. A circuit consists of the following;

1. $\mathbb{F}$ — A finite field.
2. $\vec{c}$ — A vector of column types, where the column types are taken from the set $\{F, A, I\} \times \{E, N\}$.

   These are used to mark a column as, F-fixed (independent of user input), A-advice (dependent on user input), I-instance (encoding the statement we're arithmetizing), E-equality constrainable, and N-not equality constrainable.
3. d — An integer denoting the polynomial degree.
4. $\vec{p}$ — A vector of polynomials of degree $\leq d$ with formal variables in $[\,|\vec{c}|\,] \times \mathbb{Z}$. These act as polynomial constraints over columns and relative row references within a matrix.
5. $\vec{l}$ — A vector of "lookup constraints", where a lookup constraint is a pair of same-length vectors,

   $\vec{x}$ — A vector of polynomials of degree $\leq d$ with formal variables in $[\,|\vec{c}|\,] \times \mathbb{Z}$.

   $\vec{a}$ — A vector of column indexes, that is, a vector of values taken from $[\,|\vec{c}|\,]$.

These are used to check that all values specified by the $\vec{x}$s are within the columns specified by $\vec{a}$.

6. r — A positive integer denoting the number of rows.
7. $\vec{e}$ — A collection of vectors taken from $([\,|\vec{c}|\,] \times [r])^2$, that is, each entry is a pair of coordinates which will be asserted to be equal.
8. X — A $u \times r$ matrix of $\mathbb{F}$s, where u is the number of fixed columns. These act as the table of values for each fixed column.

Each Halo 2 circuit defines a relation over a $v \times r$ matrix of $\mathbb{F}$s, where v is the number of *instance* columns. It denotes the assertion that, for a given $Y \in \mathbb{F}^{v \times r}$, there exists a matrix $Z \in \mathbb{F}^{w \times [r]}$, where w is the number of *advice* columns, such that the matrix $M = [XYZ] \in \mathbb{F}^{[\,|\vec{c}|\,] \times r}$ satisfies the following constraints:

1. For all i, j, $X_{i,j} = M_{i,j}$; that is the values of the fixed columns stay fixed.
2. Each polynomial in $\vec{p}$ is 0 when evaluated on points within M.
3. For each lookup constraint, the output values of the polynomials in the $\vec{x}$s when evaluated on points taken from M are present in the columns of M specified by the $\vec{a}$s.
4. Each column mentioned in the $\vec{e}$s has the E/equality constrainable type.
5. Every pair of values taken from $\vec{e}$ within M are equal.

See section 2 of [Tho22] for a more detailed account of this definition.

This paper will not stick exclusively to the tools provided by this definition. It is, by this point, somewhat of a rote procedure to convert boolean constraints into arithmetic constraints. See, for example, sections 5 and 7 of [TH22]. Additionally, there is much tedious bureaucracy in accounting for the fact that every column must have the same row number, among other requirements for real world circuits. For the sake of a clearer presentation, we will take for granted that an interested party will be able to fill in these missing details.

5 Arithmetization of Traces

An interaction trace is parameterized by a node context and trace setting. This can be incorporated as a variety of different polynomial constraints. Give arbitrary, numerical names to the various types, nodes, and rules.

1. Denote by $p0^+(x)$ a polynomial which is 0 iff the type named by x has no positive ports. In the case of symmetric interaction combinators, for example, this holds for root (0) and deletion (5) nodes, which implies the definition;

$$p0^+(x) := (x - 0)(x - 5) \tag{1}$$

   in that case. Denote by $p0^-(x)$ a polynomial which is 0 iff the type named by x has no negative ports. Define it analogously to $p0^+(x)$.

2. Denote by $\mathcal{T}_{MAX}$ the maximum value for a node type. In the case of symmetric interaction combinators, this is 6, since the 7 node types will be labeled 0 through 6.

   Define the polynomial $pT(x)$ which will be 0 iff x is a valid type. Define this as

$$pT(x) := \prod_{i=0}^{\mathcal{T}_{MAX}} (x - i) \tag{2}$$

3. Denote by $\mathcal{P}^+_{t,MAX}$ the maximum value of a positive port for type t. Denote by $\mathcal{P}^-_{t,MAX}$ the maximum value of a negative port for type t. This value does not exist for types without positive (respectively, negative) ports.

   Furthermore, define $\mathcal{P}^+_{MAX}(t)$ to be the interpolating polynomial which maps t onto $\mathcal{P}^+_{t,MAX}$. Similarly for $\mathcal{P}^-_{MAX}(t)$.

The interaction rules can be incorporated similarly.

1. A polynomial $pR0^+(x)$ which will be 0 iff x names a rule with no output nodes. This will be defined in the same way as $p0^+(x)$, as the product of all expressions $(x - i)$ where i names a rule with no output ports.

2. Define $\mathcal{R}_{MAX}$ to be the maximum value representing a rule name. For the symmetric interaction case, there are 9 rules labeled 0 through 8, so $\mathcal{R}_{MAX}$ will be 8. Also define $pR(x)$ to be a polynomial which is 0 iff x is a valid rule name. This can be defined as

$$pR(x) := \prod_{i=0}^{\mathcal{R}_{MAX}} (x - i) \tag{3}$$

3. Furthermore, define the maps from rules to input node types. i^+ and i^- will be the interpolating polynomials which map each rule onto the number corresponding to the node type of I^+ or, respectively, I^-.

4. Also have, for each rule r, the number $RO_{r,MAX}$ indicating the max value for an output node, if it exists. For example, if r has 4 output nodes, they will be labeled 0 through 3, so $RO_{r,MAX}$ will be 3. Furthermore, $RO_{MAX}(r)$ will be the interpolating polynomial mapping rule r onto $RO_{r,MAX}$.

5. Define a bivariate polynomial which will map each pair of rule and output node to a type in accordance with o. Call this interpolating polynomial po.

6. Lastly, for each rule r, have a six variable polynomial, pw_r. The arguments are, in order,

   (a) The polarity of the input node (0 for positive/output, 1 for negative/input).

   (b) The input node number.

   (c) The input port number.

   (d) The polarity of the output node.

   (e) The output node number.

   (f) The output port number.

   pw_r will be 0 iff the three numbers defining the input node are mapped to the three numbers defining the output node by the wiring of the rule. As an example, the wiring for the second rule in figure 2 corresponds to the following table of arguments.

$$\begin{array}{cccccc} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 2 & 0 \\ 0 & 2 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 2 & 1 & 0 & 1 & 1 \\ 0 & 3 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 3 & 1 & 0 & 1 & 0 \end{array}$$

   The node names are arbitrary with 0 being the top-left, 1 the top-right, 2 the bottom left, and 3 the bottom-right. The ports are named starting at 0, clockwise starting at the principal port. Both positive and negative port namings start at 0. Each argument can be given a parabola;

$$(x_1 - a)^2 + (x_2 - b)^2 + (x_3 - c)^2 + (x_4 - d)^2 + (x_5 - e)^2 + (x_6 - f)^2$$

   which will be 0 iff $(x_1, x_2, x_3, x_4, x_5, x_6) = (a, b, c, d, e, f)$. pw_r will be the product of all these parabolas for each argument in the argument table for rule r. Note that the table of arguments does **not** contain the principal link of that rule, only the information from the output wiring.

we will need the following fixed column to aid in the later arithmetization;

$\vec{\iota}$ — A column which starts at 0 and contains all 1s elsewhere.

Throughout the rest of this section, k will always be used to denote a row index, and this value will be universally quantified for every constraint.

A collection of instance columns to encode the starting net is needed. First, have two columns encoding the types of each node in the net

1. $S\vec{N}$ — An enumeration of each node in the starting net.
2. $S\vec{\tau}$ — An assignment of a type to each node in the net.

As a running example, we'll describe part of the arithmetization for evaluating a garbage collector being applied to the identity function. The starting net will have four nodes, 0-root, 1-application, 2-lambda, and 3-the garbage collector.

$S\vec{N}$	$S\vec{\tau}$
0	0
1	2
2	1
3	6
3	6
⋮	⋮
3	6

The main restriction is that every same entry in $S\vec{N}$ is given the same value in $S\vec{\tau}$. Enforce this via the constraints;

1. $\iota_{k+1} = 0 \vee S\vec{N}_k = S\vec{N}_{k+1} \vee S\vec{N}_k + 1 = S\vec{N}_{k+1}$
2. $S\vec{N}_k = S\vec{N}_{k+1} \to S\vec{\tau}_k = S\vec{\tau}_{k+1}$

These conditions cause these columns to define a partial function. Denote by $S\tau(x) = y$ the lookup constraint $(x, y) \in (S\vec{N}, S\vec{\tau})$.

Additionally, ensure that each entry in $S\vec{\tau}$ is a valid type.

$$pT(S\tau_k) = 0$$

Further columns for the internal bijection are needed.

1. $S\vec{T}^+$ — Types for entries in $S\vec{N}^+$.
2. $S\vec{N}^+$ — Elements from $S\vec{N}$ to be mapped out of.
3. $S\vec{P}^+$ — A positive port of the node in $S\vec{N}^+$.
4. $S\vec{N}^-$ — Elements from $S\vec{N}$ to be mapped out of.
5. $S\vec{P}^-$ — A negative port of the node in $S\vec{N}^-$.
6. $S\vec{T}^*$ — Types for entries in $S\vec{N}^*$.
7. $S\vec{N}^*, S\vec{P}^*$ — The contents of $(S\vec{N}^-, S\vec{P}^-)$ put in lexicographic order to aid in bijectivity checking.

The starting net table for our running example looks like

$S\vec{T}^+$	$S\vec{N}^+$	$S\vec{P}^+$	$S\vec{N}^-$	$S\vec{P}^-$	$S\vec{T}^*$	$S\vec{N}^*$	$S\vec{P}^*$
2	1	0	0	0	0	0	0
1	2	0	1	0	1	1	0
1	2	1	2	0	0	1	1
6	3	0	1	1	1	2	0
6	3	0	1	1	1	2	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
6	3	0	1	1	1	2	0

Check that the types are all correct.

$$S\tau(SN_k^+) = ST_k^+$$

$$S\tau(SN_k^*) = ST_k^*$$

Verify completeness. Firstly, verify that every node with a positive port does, in fact, appear in $S\vec{N}^+$.

$$(SN_k, S\tau_k) \in (S\vec{N}^+, S\vec{T}^+) \leftrightarrow p0^+(S\tau_k) \neq 0$$

Verify that every node with a negative port does, in fact, appear in $S\vec{N}^*$.

$$(SN_k, S\tau_k) \in (S\vec{N}^*, S\vec{T}^*) \leftrightarrow p0^-(S\tau_k) \neq 0$$

Next, verify that every port is covered uniquely by the table. Start by ensuring that $S\vec{N}^+$ and $S\vec{N}^*$ are ordered.

$$\iota_{k+1} = 0 \vee SN_k^+ \leq SN_{k+1}^+$$
$$\iota_{k+1} = 0 \vee SN_k^* \leq SN_{k+1}^*$$

Now that the nodes are ordered, ensure that all ports are present. Verify that the first port for each node is always 0 and the last port is always a max.

$$SP_0^+ = 0$$
$$SN_k^+ \neq SN_{k+1}^+ \rightarrow (SP_k^+ = \mathcal{P}_{MAX}^+(ST_k^+) \wedge SP_{k+1}^+ = 0)$$

$$SP_0^* = 0$$
$$SN_k^* \neq SN_{k+1}^* \rightarrow (SP_k^* = \mathcal{P}_{MAX}^-(ST_k^*) \wedge SP_{k+1}^* = 0)$$

Furthermore, the port number should keep incrementing so long as it hasn't reached a max.

$$SN_k^+ = SN_{k+1}^+ \rightarrow SP_k^+ + 1 = SP_{k+1}^+$$
$$SN_k^* = SN_{k+1}^* \rightarrow SP_k^* + 1 = SP_{k+1}^*$$

Lastly, check that $(S\vec{N}^*, S\vec{P}^*)$ is a permutation of $(S\vec{N}^-, S\vec{P}^-)$. Verify this with the simple lookup constraint;

$$(SN_k^*, SP_k^*) \in (S\vec{N}^-, S\vec{P}^-) \tag{4}$$

Every entry in $(S\vec{N}^*, S\vec{P}^*)$ being unique (ignoring ending repetitions), along with the fact that $(S\vec{N}^-, S\vec{P}^-)$ has the same number of entries (prior to ending repetitions), imply that $(S\vec{N}^*, S\vec{P}^*)$ must fill all slots available in $(S\vec{N}^-, S\vec{P}^-)$. The other direction of the inclusion is implied by the previous checks.

A collection of instance columns to encode the ending net are needed. First, have two columns encoding the types of each node in the net

1. $E\vec{N}$ — An enumeration of each node in the ending net.
2. $E\vec{\tau}$ — An assignment of a type to each node in the net.

and columns to encode the bijection of the ending net are needed

1. $E\vec{T}^+$ — Types for entries in $E\vec{N}^+$.
2. $E\vec{N}^+$ — Elements from $E\vec{N}$ to be mapped out of.
3. $E\vec{P}^+$ — A positive port of the node in $E\vec{N}^+$.
4. $E\vec{N}^-$ — Elements from $E\vec{N}$ to be mapped out of.

5. $E\vec{P}^-$ — A negative port of the node in $E\vec{N}^-$.

6. $E\vec{T}^*$ — Types for entries in $E\vec{N}^*$.

7. $E\vec{N}^*, E\vec{P}^*$ — The contents of $(E\vec{N}^-, E\vec{P}^-)$ put in lexicographic order to aid in bijectivity checking.

These will be subject to identical constraints to those for the starting net columns.

A collection of advice columns to encode the trace information is needed. Start with two columns, assigning interaction types to each step.

1. $\vec{\pi}$ — A column containing, at each row, the total number of steps plus 1.

2. $\vec{s}$ — An enumeration of each interaction step in the trace.

3. $\vec{r}$ — An assignment of a rule of interaction to each step.

Have a constraint asserting that $\vec{\pi}$ is, in fact, a constant.

$$\pi_k = \pi_{k+1}$$

$(\vec{s}, \vec{r})$ will be functional in the same sense as $(S\vec{N}, S\vec{\tau})$, and so will satisfy some of the same constraints.

$$\vec{s}_k = \vec{s}_{k+1} \vee \vec{s}_k + 1 = \vec{s}_{k+1}$$
$$\vec{s}_k = \vec{s}_{k+1} \to \vec{r}_k = \vec{r}_{k+1}$$
$$s_k = \pi_k \to s_{k+1} = \pi_{k+1}$$

by $r(x) = y$ we mean the lookup constraint $(x, y) \in (\vec{s}, \vec{r})$.

Also ensure that each entry in $\vec{r}$ is a valid rule.

$$pR(r_k) = 0$$

Columns to dictate the node wiring within the trace are needed:

1. $T\vec{R}^+$ — Precomputed rule for entries in $T\vec{S}^+$.

2. $T\vec{S}^+$ — The step of the nodes in $T\vec{N}^+$

3. $T\vec{N}T$ — Precomputed types of $T\vec{N}^+$ and $T\vec{N}^-$.

4. $T\vec{N}^+$ — Output/starting nodes of the trace.

5. $T\vec{R}^-$ — Precomputed rule for entries in $T\vec{S}^-$.

6. $T\vec{S}^-$ — The step of the nodes in $T\vec{N}^-$

7. $T\vec{N}^-$ — Input/ending nodes of the trace.

8. $T\vec{R}^*$ — Precomputed rule for entries in $T\vec{S}^*$.

9. $T\vec{S}^*, T\vec{N}^*$ — The contents of $(T\vec{S}^-, T\vec{N}^-)$ put in lexicographic order to aid in bijectivity checking.

Many of the checks on these columns will be very similar to those for the starting and ending nets.

Check that the rules are all correct. Note that step 0 will refer to the starting net and step $\pi + 1$ will refer to the ending net, so the numbers appearing in the TS columns will be the level, not the step number. The rule numbers for the starting and ending nets will be $\mathcal{R}_{MAX} + 1$ and $\mathcal{R}_{MAX} + 2$, respectively.

$$TS_k^+ = 0 \to TR_k^+ = \mathcal{R}_{MAX} + 1$$
$$TS_k^+ = \pi_k \to TR_k^+ = \mathcal{R}_{MAX} + 2$$
$$(TS_k^+ \neq 0 \wedge TS_k^+ \neq \pi_k) \to r(TS_k^+ - 1) = TR_k^+$$

$$TS_k^- = 0 \to TR_k^- = \mathcal{R}_{MAX} + 1$$
$$TS_k^- = \mathbb{n}_k \to TR_k^- = \mathcal{R}_{MAX} + 2$$
$$(TS_k^- \neq 0 \wedge TS_k^- \neq \mathbb{n}_k) \to r(TS_k^- - 1) = TR_k^-$$

$$TS_k^* = 0 \to TR_k^* = \mathcal{R}_{MAX} + 1$$
$$TS_k^* = \mathbb{n}_k \to TR_k^* = \mathcal{R}_{MAX} + 2$$
$$(TS_k^* \neq 0 \wedge TS_k^* \neq \mathbb{n}_k) \to r(TS_k^* - 1) = TR_k^*$$

Check that the input and output types match on each node.

$$(TS_k^+ \neq 0 \wedge TS_k^- \neq \mathbb{n}_k \wedge TN_k^- = 0) \to (i^+(TN_k^-) = TNT_k \wedge po(TR_k^+, TN_k^+) = TNT_k)$$
$$(TS_k^+ \neq 0 \wedge TS_k^- \neq \mathbb{n}_k \wedge TN_k^- = 1) \to (i^-(TN_k^-) = TNT_k \wedge po(TR_k^+, TN_k^+) = TNT_k)$$
$$(TS_k^+ = 0 \wedge TS_k^- \neq \mathbb{n}_k \wedge TN_k^- = 0) \to (i^+(TN_k^-) = TNT_k \wedge S\tau(TN_k^+) = TNT_k)$$
$$(TS_k^+ = 0 \wedge TS_k^- \neq \mathbb{n}_k \wedge TN_k^- = 1) \to (i^-(TN_k^-) = TNT_k \wedge S\tau(TN_k^+) = TNT_k)$$
$$(TS_k^+ \neq 0 \wedge TS_k^- = \mathbb{n}_k) \to (E\tau(TN_k^-) = TNT_k \wedge po(TR_k^+, TN_k^+) = TNT_k)$$
$$(TS_k^+ = 0 \wedge TS_k^- = \mathbb{n}_k) \to (E\tau(TN_k^-) = TNT_k \wedge S\tau(TN_k^+) = TNT_k)$$

Verify completeness. Firstly, verify that every step with output ports does, in fact, appear in $T\vec{S}^+$.

$$(s_k, r_k) \in (T\vec{S}^+, T\vec{R}^+) \leftrightarrow (pR0^+(r_k) \neq 0)$$

Every step will appear in $T\vec{S}^-$ and $T\vec{S}^*$ since every rule has an input. A special check isn't needed for this; instead, this follows from the ordering constraints. These say that

$$\iota_{k+1} = 0 \vee TS_k^+ \leq TS_{k+1}^+$$

$$TS_0^* = 1$$
$$\iota_{k+1} = 0 \vee TS_k^* = TS_{k+1}^* \vee TS_k^* + 1 = TS_{k+1}^*$$
$$\mathbb{n}_k \in T\vec{S}^*$$
$$(TS_k^* = \mathbb{n}_k \wedge \iota_{k+1} \neq 0) \to (TS_{k+1}^* = \mathbb{n}_{k+1})$$

Now that the steps are ordered and present, ensure that all nodes are present. Do this by first verifying that the first node for each step is always 0 and the last port is always a max.

$$(TS_k^+ \neq TS_{k+1}^+ \wedge TS_k^+ \neq 0) \to (TN_k^+ = \mathcal{R}_{MAX}^+(TR_k^+) \wedge TN_{k+1}^+ = 0)$$

$$TN_0^* = 0$$
$$(TS_k^* \neq TS_{k+1}^* \wedge TS_k^+ \neq \mathbb{n}_k) \to (TN_k^* = 1 \wedge TN_{k+1}^* = 0)$$

Furthermore, the port number should keep incrementing so long as it hasn't reached a max.

$$(TS_k^+ = TS_{k+1}^+ \wedge \iota_{k+1} \neq 0) \to TN_k^+ + 1 = TN_{k+1}^+$$
$$(TS_k^* = TS_{k+1}^* \wedge \iota_{k+1} \neq 0) \to TN_k^* + 1 = TN_{k+1}^*$$

Verify that all the nodes in the starting net appear in $T\vec{N}^+$.

$$(0, SN_k) \in (T\vec{S}^+, T\vec{N}^+)$$

Verify that all the nodes in the ending net appear in $T\vec{N}^*$.

$$(\mathbb{n}_k, EN_k) \in (T\vec{S}^*, T\vec{N}^*)$$

Lastly, check that $(S\vec{N}^*, S\vec{P}^*)$ is a permutation of $(S\vec{N}^-, S\vec{P}^-)$. Given the previous checks, verify this with the simple lookup constraint;

$$(SN_k^*, SP_k^*) \in (S\vec{N}^-, S\vec{P}^-) \tag{5}$$

The very last thing needed is a group of columns enumerating and explicating the port structure of the trace. The following columns are needed:

1. $\vec{\mathcal{S}}, \vec{\mathcal{G}}, \vec{\mathcal{N}}, \vec{\mathcal{L}}, \vec{\mathcal{P}}$ — Columns containing the ports of the trace in lexicographic order by: step ($\vec{\mathcal{S}}$), node polarity ($\vec{\mathcal{G}}$), node ($\vec{\mathcal{N}}$), port polarity ($\vec{\mathcal{L}}$), then port ($\vec{\mathcal{P}}$).
2. $\vec{\mathcal{N}T}$ — The precomputed node types for $\vec{\mathcal{N}}$.
3. $\vec{\mathcal{S}R}$ — The precomputed rules for the interaction step.
4. $\vec{!}$ — An ordered list giving names to each port cycle.
5. $\vec{F}$ — An ordered list of flags swapping between 0 and 1 and starting at 0.
6. $\vec{\mathcal{S}}!, \vec{\mathcal{G}}!, \vec{\mathcal{N}}!, \vec{\mathcal{L}}!, \vec{\mathcal{P}}!, \vec{\mathcal{S}R}!$ — The ports within the trace listed according to their order within a port cycle, starting at the negative (principal or output) port and ending at the positive.
7. $\vec{\mathcal{S}D}, \vec{\mathcal{N}D}, \vec{\mathcal{L}D}, \vec{\mathcal{P}D}$ — Columns indicating the negative (principal or output) port at the beginning of each cycle.

The number of ports in the trace will dominate the number of rows in the circuit.

For our running example, the full trace will start with the identity function applied to a garbage collector. This will then be evaluated to a garbage collector connected to a root node in a single step. This trace will have two port cycles. The full trace will look like;

$\vec{\mathcal{S}}$	$\vec{\mathcal{G}}$	$\vec{\mathcal{N}}$	$\vec{\mathcal{L}}$	$\vec{\mathcal{P}}$	$\vec{\mathcal{N}T}$	$\vec{\mathcal{S}R}$	$\vec{!}$	$\vec{F}$	$\vec{\mathcal{S}}!$	$\vec{\mathcal{G}}!$	$\vec{\mathcal{N}}!$	$\vec{\mathcal{L}}!$	$\vec{\mathcal{P}}!$	$\vec{\mathcal{S}R}!$	$\vec{\mathcal{S}D}$	$\vec{\mathcal{N}D}$	$\vec{\mathcal{L}D}$	$\vec{\mathcal{P}D}$
0	0	0	1	0	0	9	0	0	2	1	0	1	0	10	2	0	1	0
0	0	1	0	0	2	9	0	1	0	0	0	1	0	9	2	0	1	0
0	0	1	1	0	2	9	0	0	0	0	1	0	0	9	2	0	1	0
0	0	1	1	1	2	9	0	1	1	1	1	0	0	0	2	0	1	0
0	0	2	0	0	1	9	0	0	1	1	0	1	0	0	2	0	1	0
0	0	2	0	1	1	9	0	1	0	0	2	1	0	9	2	0	1	0
0	0	2	1	0	1	9	0	0	0	0	2	0	1	9	2	0	1	0
0	0	3	0	0	5	9	0	1	1	1	0	0	1	0	2	0	1	0
1	1	0	0	0	1	0	0	0	1	1	1	1	1	0	2	0	1	0
1	1	0	0	1	1	0	0	1	0	0	1	1	1	9	2	0	1	0
1	1	0	1	0	1	0	0	0	0	0	3	0	0	9	2	0	1	0
1	1	1	0	0	2	0	0	1	2	1	1	0	0	10	2	0	1	0
1	1	1	1	0	2	0	1	0	1	1	1	1	0	0	1	1	1	0
1	1	1	1	1	2	0	1	1	0	0	1	1	0	9	1	1	1	0
2	1	0	1	0	0	10	1	0	0	0	2	0	0	9	1	1	1	0
2	1	1	0	0	5	10	1	1	1	1	0	0	0	0	1	1	1	0

Ensure that the rules are correct.

$$(\mathcal{S}_k \neq 0 \wedge \mathcal{S}_k \neq \mathbb{n}_k) \rightarrow (\mathcal{S}_k - 1, \mathcal{S}R_k) \in (\vec{s}, \vec{r})$$
$$\mathcal{S}_k = 0 \rightarrow \mathcal{S}R_k = \mathcal{R}_{MAX} + 1$$
$$\mathcal{S}_k = \mathbb{n}_k \rightarrow \mathcal{S}R_k = \mathcal{R}_{MAX} + 2$$

Ensure that the node types are correct.

$$\mathcal{S}_k = 0 \rightarrow (\mathcal{N}_k, \mathcal{N}T_k) \in (S\vec{N}, S\vec{\tau})$$
$$\mathcal{S}_k = \mathbb{n}_k \rightarrow (\mathcal{N}_k, \mathcal{N}T_k) \in (E\vec{N}, E\vec{\tau})$$
$$(\mathcal{S}_k \neq 0 \wedge \mathcal{S}_k \neq \mathbb{n}_k \wedge \mathcal{G}_k = 0 \wedge \mathcal{N}_k = 0) \rightarrow i^+(\mathcal{S}_k) = \mathcal{N}T_k$$

$$(\mathcal{S}_k \neq 0 \wedge \mathcal{S}_k \neq \mathbb{n}_k \wedge \mathcal{G}_k = 0 \wedge \mathcal{N}_k = 1) \rightarrow i^-(\mathcal{S}_k) = \mathcal{N}T_k$$
$$(\mathcal{S}_k \neq 0 \wedge \mathcal{S}_k \neq \mathbb{n}_k \wedge \mathcal{G}_k = 1) \rightarrow po(\mathcal{S}R_k, \mathcal{N}_k) = \mathcal{N}T_k$$

Constraints to show that $\vec{\mathcal{S}}, \vec{\mathcal{N}}, \vec{\mathcal{P}}$ are a complete enumeration are also required.
The step column must start at 0 and periodically increment to cover all steps.

$$\mathcal{S}_0 = 0$$
$$\iota_{k+1} = 0 \vee \mathcal{S}_k = \mathcal{S}_{k+1} \vee \mathcal{S}_k + 1 = \mathcal{S}_{k+1}$$
$$\mathbb{n}_k \in \mathcal{S}$$
$$(S_k = \mathbb{n}_k \wedge \iota_{k+1} \neq 0) \rightarrow S_{k+1} = \mathbb{n}_{k+1}$$

With the steps now listed in order, verify that the port polarities are complete. Positive ports will be denoted with a 0 and negative ports with a 1. The port polarities will be listed with negative ports first, since all interaction rules have negative ports.

$$\mathcal{S}_k = 0 \rightarrow \mathcal{G}_k = 0$$
$$\mathcal{S}_k = \mathbb{n}_k \rightarrow \mathcal{G}_k = 1$$
$$\mathcal{S}_k = \mathcal{S}_{k+1} \rightarrow (\mathcal{G}_k = \mathcal{G}_{k+1} \vee \mathcal{G}_k = \mathcal{G}_{k+1} + 1)$$
$$(\mathcal{S}_k \neq \mathcal{S}_{k+1} \wedge \mathcal{S}_k \neq 0 \wedge \mathcal{S}_k \neq \mathbb{n}_k \wedge pR0^+(\mathcal{S}R_k - 1) = 0) \rightarrow (\mathcal{G}_k = 1 \wedge \mathcal{G}_{k+1} = 1)$$
$$(\mathcal{S}_k \neq \mathcal{S}_{k+1} \wedge \mathcal{S}_k \neq 0 \wedge \mathcal{S}_k \neq \mathbb{n}_k \wedge pR0^+(\mathcal{S}R_k - 1) \neq 0) \rightarrow (\mathcal{G}_k = 0 \wedge \mathcal{G}_{k+1} = 1)$$

These verify that the node polarities are completely enumerated. Now verify that all and only the correct nodes appear in $\mathcal{N}$.

$$(0, SN_k) \in (\vec{\mathcal{S}}, \vec{\mathcal{N}})$$
$$(\mathbb{n}_k, EN_k) \in (\vec{\mathcal{S}}, \vec{\mathcal{N}})$$
$$(\mathcal{S}_k = \mathcal{S}_{k+1} \wedge \mathcal{G}_k = \mathcal{G}_{k+1}) \rightarrow (\mathcal{N}_k = \mathcal{N}_{k+1} \vee \mathcal{N}_k + 1 = \mathcal{N}_{k+1})$$
$$(\mathcal{S}_k = \mathcal{S}_{k+1} \wedge \mathcal{G}_k + 1 = \mathcal{G}_{k+1}) \rightarrow (\mathcal{N}_k = 1 \wedge \mathcal{N}_{k+1} = 0)$$
$$(\mathcal{S}_k \neq \mathcal{S}_{k+1} \wedge \mathcal{S}_k \neq 0) \rightarrow (\mathcal{N}_k = RO_{MAX}(\mathcal{S}R_k - 1) \wedge \mathcal{N}_{k+1} = 0)$$

At this point, verifying completeness can get a bit hairy. Verify that two subsequent nodes are different via the following relation;

$$dN_k := \mathcal{S}_k \neq \mathcal{S}_{k+1} \vee \mathcal{G}_k \neq \mathcal{G}_{k+1} \vee \mathcal{N}_k \neq \mathcal{N}_{k+1}$$

With this, check port polarity completeness with

$$\neg dN_k \rightarrow \mathcal{L}_k = \mathcal{L}_{k+1} \vee \mathcal{L}_k + 1 = \mathcal{L}_{k+1}$$
$$(dN_k \wedge p0^-(\mathcal{N}T_k) \neq 0 \wedge p0^+(\mathcal{N}T_{k+1}) \neq 0) \rightarrow (\mathcal{L}_k = 1 \wedge \mathcal{L}_{k+1} = 0)$$
$$(dN_k \wedge p0^-(\mathcal{N}T_k) = 0 \wedge p0^+(\mathcal{N}T_{k+1}) \neq 0) \rightarrow (\mathcal{L}_k = 0 \wedge \mathcal{L}_{k+1} = 0)$$
$$(dN_k \wedge p0^-(\mathcal{N}T_k) \neq 0 \wedge p0^+(\mathcal{N}T_{k+1}) = 0) \rightarrow (\mathcal{L}_k = 1 \wedge \mathcal{L}_{k+1} = 1)$$
$$(dN_k \wedge p0^-(\mathcal{N}T_k) = 0 \wedge p0^+(\mathcal{N}T_{k+1}) = 0) \rightarrow (\mathcal{L}_k = 0 \wedge \mathcal{L}_{k+1} = 1)$$

This check relies fundamentally on the cyclic nature of checks, which guarantees that the 0th row will be set to 0 without having to say this explicitly.

Now that all the port polarities are present, verify the presence of all the ports via

$$(dN_k \wedge \mathcal{L}_k = 0) \rightarrow (\mathcal{P}_k = \mathcal{P}^+_{MAX}(\mathcal{N}T_k) \wedge \mathcal{P}_{k+1} = 0)$$
$$(dN_k \wedge \mathcal{L}_k = 1) \rightarrow (\mathcal{P}_k = \mathcal{P}^-_{MAX}(\mathcal{N}T_k) \wedge \mathcal{P}_{k+1} = 0)$$
$$(\neg dN_k \wedge \mathcal{L}_k + 1 = \mathcal{L}_{k+1}) \rightarrow (\mathcal{P}_k = \mathcal{P}^+_{MAX}(\mathcal{N}T_k) \wedge \mathcal{P}_{k+1} = 0)$$
$$(\neg dN_k \wedge \mathcal{L}_k = \mathcal{L}_{k+1}) \rightarrow \mathcal{P}_k + 1 = \mathcal{P}_{k+1}$$

These, again, rely on the cyclic nature of checks. This completes all the completeness checks.

Now put together the constraints necessary to check temporal consistency. Firstly, check that the columns containing the ports in cycle order are a permutation of all ports in the trace.

$$(\mathcal{S}_k, \mathcal{G}_k, \mathcal{N}_k, \mathcal{L}_k, \mathcal{P}_k, \mathcal{SR}_k) \in (\vec{\mathcal{S}}!, \vec{\mathcal{G}}!, \vec{\mathcal{N}}!, \vec{\mathcal{L}}!, \vec{\mathcal{P}}!, \vec{\mathcal{SR}}!)$$

The flag column should swap back and forth between 0 and 1.

$$F_0 = 0$$
$$1 - F_k = F_{k+1}$$

Incidentally, this implies that the total number of ports is even. And, in fact, the total ports in each port cycle is even, so the first flag of each cycle will be 0 and the last flag will be 1.

The port cycles themselves should be uniquely named by $\vec{!}$, but have no structure beyond that.

$$!_0 = 0$$
$$\iota_{k+1} = 0 \vee !_k = !_{k+1} \vee !_k + 1 = !_{k+1}$$

To check each port cycle, use one of the two bijections used to define the port cycle. The flag dictates which bijection is being used.

If the flag is 0 then follow the trace wiring between different steps.

$$(!_k = !_{k+1} \wedge F_k = 0 \wedge \mathcal{G}!_k = 0) \to (\mathcal{S}!_k, \mathcal{N}!_k, \mathcal{S}!_{k+1}, \mathcal{N}!_{k+1}) \in (T\vec{S}^+, T\vec{N}^+, T\vec{S}^-, S\vec{N}^-)$$
$$(!_k = !_{k+1} \wedge F_k = 0 \wedge \mathcal{G}!_k = 1) \to (\mathcal{S}!_{k+1}, \mathcal{N}!_{k+1}, \mathcal{S}!_k, \mathcal{N}!_k) \in (T\vec{S}^+, T\vec{N}^+, T\vec{S}^-, S\vec{N}^-)$$
$$(!_k = !_{k+1} \wedge F_k = 0) \to (\mathcal{L}!_k = \mathcal{L}!_{k+1} \wedge \mathcal{P}!_k = \mathcal{P}!_{k+1})$$

When the flag is 1 follow the internal wiring of either the rules or the starting net.

The step shouldn't change.

$$(!_k = !_{k+1} \wedge F_k = 1) \to \mathcal{S}!_k = \mathcal{S}!_{k+1}$$

For the starting net:

$$(!_k = !_{k+1} \wedge F_k = 1 \wedge \mathcal{S}!_k = 0 \wedge \mathcal{L}!_k = 0) \to (\mathcal{N}!_k, \mathcal{P}!_k, \mathcal{N}!_{k+1}, \mathcal{P}!_{k+1}) \in (S\vec{N}^+, S\vec{P}^+, S\vec{N}^-, S\vec{P}^-)$$

$$(!_k = !_{k+1} \wedge F_k = 1 \wedge \mathcal{S}!_k = 0 \wedge \mathcal{L}!_k = 1) \to (\mathcal{N}!_{k+1}, \mathcal{P}!_{k+1}, \mathcal{N}!_k, \mathcal{P}!_k) \in (S\vec{N}^+, S\vec{P}^+, S\vec{N}^-, S\vec{P}^-)$$

For each rule r, have two constraints for the interactions

$$(!_k = !_{k+1} \wedge F_k = 1 \wedge \mathcal{S}!_k \neq 0 \wedge \mathcal{S}!_k \neq \mathbb{n}_k \wedge ((\mathcal{G}!_k = 0 \wedge \mathcal{L}!_k = 0) \vee (\mathcal{G}!_k = 1 \wedge \mathcal{L}!_k = 1)) \wedge \mathcal{SR}!_k = r) \\ \to pw_r(\mathcal{G}!_k, \mathcal{N}!_k, \mathcal{P}!_k, \mathcal{G}!_{k+1}, \mathcal{N}!_{k+1}, \mathcal{P}!_{k+1}) = 0$$

$$(!_k = !_{k+1} \wedge F_k = 1 \wedge \mathcal{S}!_k \neq 0 \wedge \mathcal{S}!_k \neq \mathbb{n}_k \wedge ((\mathcal{G}!_k = 0 \wedge \mathcal{L}!_k = 1) \vee (\mathcal{G}!_k = 1 \wedge \mathcal{L}!_k = 0)) \wedge \mathcal{SR}!_k = r) \\ \to pw_r(\mathcal{G}!_{k+1}, \mathcal{N}!_{k+1}, \mathcal{P}!_{k+1}, \mathcal{G}!_k, \mathcal{N}!_k, \mathcal{P}!_k) = 0$$

Now check that the ends of the cycles match. The ends of each cycle should be output and principal ports.

$$!_k \neq !_{k+1} \leftrightarrow ((\mathcal{S}!_k = \mathbb{n}_k \vee (\mathcal{G}!_k = 1 \wedge \mathcal{P}!_k = 0)) \wedge (\mathcal{S}!_{k+1} = \mathbb{n}_k \vee (\mathcal{G}!_{k+1} = 1 \wedge \mathcal{P}!_{k+1} = 0)))$$

This condition forces the port cycle to change any time either an output or principal port is encountered. Also, ensure that the starting port is copied over to the D columns;

$$!_k \neq !_{k+1} \to (\mathcal{S}D_{k+1} = \mathcal{S}!_{k+1} \wedge \mathcal{N}D_{k+1} = \mathcal{N}!_{k+1} \wedge \mathcal{L}D_{k+1} = \mathcal{L}!_{k+1} \wedge \mathcal{P}D_{k+1} = \mathcal{P}!_{k+1})$$
$$!_k = !_{k+1} \to (\mathcal{S}D_{k+1} = \mathcal{S}D_k \wedge \mathcal{N}D_{k+1} = \mathcal{N}D_k \wedge \mathcal{L}D_{k+1} = \mathcal{L}D_k \wedge \mathcal{P}D_{k+1} = \mathcal{P}D_k)$$

Ensure that the port cycle reaches a maximum at its ends;

$$\mathcal{S}!_k \leq \mathcal{S}D_k$$

And the very last thing to be checked is that the ends of the same port cycle connect along principal or output ports.

In the case of output ports:

$$(!_k \neq !_{k+1} \wedge \mathcal{S}D_k = \mathbb{n}_k) \rightarrow (\mathcal{S}!_k = \mathbb{n}_k \wedge (\mathcal{N}!_k, \mathcal{P}!_k, \mathcal{N}D_k, \mathcal{P}D_k) \in (EN^+, EP^+, EN^-, EP^-))$$

In the case of principal ports:

$$!_k \neq !_{k+1} \rightarrow (\mathcal{S}!_k = \mathcal{S}D_k \wedge \mathcal{P}!_k = 0 \wedge \mathcal{P}D_k = 0 \wedge \mathcal{L}!_k = 0 \wedge \mathcal{L}D_k = 1 \wedge \mathcal{G}!_k = 1)$$

That completes all the constraints necessary to arithmetize interaction net traces.

6 Conclusion and Future Work

We have presented a description of interaction nets and have described how to arithmetize them in Halo 2.

While we've sketched out a design, this has yet to be implemented. Implementing it would be an obvious next step. Other practical considerations, such as efficiency and optimizations, must be examined to assess real world utility. Additionally, a formalization effort would be useful to confirm the correctness of the construction presented here.

Additional variations of this construction could easily be made. Instead of making the computational rules part of the polynomial constraints, one could make them instance columns, allowing for a circuit which is universal over all interaction combinator calculi.

An additional variant might fix the ending net and/or part of the starting net, allowing one to arithmetize bespoke statements along the lines of "X is an input which causes a fixed f to output a fixed Y".

Additionally, more useful variants of interaction combinators often include custom operations for performing arithmetic and boolean operations. This may be required for real-world usage.

Interaction nets are a special case of a string diagram [Sel10]. These have been used in many applications such as physics [Eas22], linear algebra [Zan18], boolean unsatisfiability [Zan], and many, many more. A theory of relational universal algebra based on string diagrams was presented in [BPS17].

Instead of arithmetizing statements as assertions about the execution of programs, one might instead arithmetize statements and proofs inside diagrammatic relational calculi.

These are not so different from interaction combinators that new arithmetization techniques would be required. The main difference is that inputs to replacement rules are allowed to be more sophisticated. When evaluating graphical calculi, an instance of the subgraph isomorphism problem must be solved in order to find a subgraph that can be replaced. This can make evaluation nontrivial. For interaction nets, this check becomes a trivial check that two principal ports are connected. This makes interaction combinators more practical as a model for programming. However, we are verifying traces after the fact, so there is no real benefit from that in this setting. Generalizing to more generic forms of graphical calculi seems desirable.

Most graphical calculi of interest are essentially syntaxes for particular traced symmetric monoidal categories, where the nodes act as atomic morphisms. Weakening the replacement rules, these become *freely-generated* 2-categories, where traces/derivations become 2-cells. Creating a scheme for arithmetizing 2-cells in free traced symmetric monoidal 2-categories should capture most calculi of interest in a single construction.

Alternatively, we could return to the precursor of interaction combinators; proof nets [Laf95]. Such nets are conceptually similar to sequent calculi, but rendered as a graph instead of a tree. Additionally, proof nets are syntactically invariant to things like premise permutation, which require additional bureaucracy in the sequent calculus. Some correctness criteria for proof nets are graphical reduction procedures comparable to those for interaction combinators [Str06]. Recent work on transcendental syntax may aid in creating generic descriptions of these ideas, which would be a good target for arithmetization [ES22]. Existing work on combinatorial proofs seem like ready-made targets for arithmetization in this vein [HSW21], though the field is still primitive, so generalizing and modifying them may be necessary for real world applications.

References

[AG98] Andrea Asperti and Stefano Guerrini. *The optimal implementation of functional programming languages*, volume 45. Cambridge University Press, 1998.

[BPS17] Filippo Bonchi, Dusko Pavlovic, and Pawel Sobocinski. Functorial semantics for relational theories. *arXiv preprint arXiv:1711.08699*, 2017.

[CGTV20] Eli Ben-Sasson Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. Tinyram architecture specification v2. 000. 2020.

[Com21] The Electric Coin Company. *The halo2 Book*. 2021.

[Eas22] Richard East. *Physique formelle de spin diagrammatique*. PhD thesis, Université Grenoble Alpes [2020-....], 2022.

[ES22] Boris Eng and Thomas Seiller. Multiplicative linear logic from a resolution-based tile system. *arXiv preprint arXiv:2207.08465*, 2022.

[HSW21] Dominic JD Hughes, Lutz Straßburger, and Jui-Hsuan Wu. Combinatorial proofs and decomposition theorems for first-order logic. In *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–13. IEEE, 2021.

[Laf89] Yves Lafont. Interaction nets. In *Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 95–108, 1989.

[Laf95] Yves Lafont. From proof nets to interaction nets. *London Mathematical Society Lecture Note Series*, pages 225–248, 1995.

[Maz07] Damiano Mazza. A denotational semantics for the symmetric interaction combinators. *Mathematical Structures in Computer Science*, 17(3):527–562, 2007.

[Sel10] Peter Selinger. A survey of graphical languages for monoidal categories. In *New structures for physics*, pages 289–355. Springer, 2010.

[Str06] Lutz Straßburger. Proof nets and the identity of proofs. *arXiv preprint cs/0610123*, 2006.

[TH22] Morgan Thomas and Anthony Hart. Arithmetization of Σ_1^1 relations with polynomial bounds in halo 2. *Cryptology ePrint Archive*, 2022.

[Tho22] Morgan Thomas. Arithmetization of Σ_1^1 relations in halo 2. *Cryptology ePrint Archive*, 2022.

[Zan] Tao Gu Robin Piedeleu Fabio Zanasi. A complete diagrammatic calculus for boolean satisfiability.

[Zan18] Fabio Zanasi. Interacting hopf algebras: the theory of linear systems. *arXiv preprint arXiv:1805.03032*, 2018.