Workgroup: Network Working Group
Published: 3 June 2025
Intended Status: Informational
Expires: 5 December 2025
Author: F. (Insert-Author-Name-here;-bug-707)
(Insert Author affiliation here)

Specification of the Plonkish Relation

Abstract

An arithmetisation is a language that a proof system uses to express statements. A circuit is a program in this language. The associated computation has been computed correctly if and only if all of the constraints in the circuit are satisified.

The primary purpose of this document is to specify a particular arithmetisation: the "Plonkish" arithmetisation used in the Halo 2 proving system.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This document describes the general Plonkish relation used in zero-knowledge proof systems. It is based on ideas in [Thomas22] and is intended to be read alongside implementation-focused material.

2. Dependencies and Notation

Plonkish arithmetization depends on a scalar field over a prime modulus `p`. We represent this field as the object `Fp`. We denote the additive identity by `0` and the multiplicative identity by `1`. Integers, taken modulo the field modulus `p`, are called scalars; arithmetic operations on scalars are implicitly performed modulo `p`. We denote the sum, difference, and product of two scalars using the `+`, `-`, and `*` operators, respectively.

The notation `a..b` means the sequence of integers from `a` (inclusive) to `b` (exclusive) in ascending order. `[a, b)` means the corresponding set of integers.

The length of a sequence `S`, or the number of elements in a set `S`, is written `#S`.

`{ f(e) : e in S }` means the set of evaluations of `f` on the set `S`.

`[f(e) : e <- a..b]` means the sequence of evaluations of `f` on `a..b`.

`[A_e]_e` means the sequence of A_e for some implicitly defined sequence of indices `e`.

When `f` is a function that takes a tuple as argument, we will allow `f((i, j))` to be written as `f[i, j]`.

The terminology used here is intended to be consistent with [ZKProofCommunityReference]. We diverge from this terminology as follows: * We refer to the public inputs to the circuit as an "instance vector". The entries of this vector are called "instance variables" in the Community Reference.

3. The General Plonkish Relation `R_plonkish`

The general relation `R_plonkish` contains pairs of `(x, w)` where: * the instance `x` consists of the parameters of the proof system, the circuit `C`, and the public inputs to the circuit (i.e. the instance vector). * the witness `w` consists of the matrix of values provided by the prover. In this model it consists of the (potentially private) prover inputs to the circuit, and any intermediate values (including fixed values) that are not inputs to the circuit but are required in order to satisfy it.

We say that a `x` is a *valid* instance whenever there exists some witness `w` such that `(x, w) in R_plonkish` holds. The Plonkish language `L_plonkish` contains all valid instances.

A circuit-specific relation is a specialization of `R_plonkish` to a particular circuit.

If the proof system is knowledge sound, then the prover must have knowledge of the witness in order to construct a valid proof. If it is also zero knowledge, then witness entries can be private, and an honestly generated proof leaks no information about the private inputs to the circuit beyond the fact that it was obtained with knowledge of some satisfying witness.

3.1. Instances

The relation `R_plonkish` takes instances of the following form:

Instance element	Description
`Fp`	A prime field.
`C`	The circuit.
`phi`	The instance vector `phi : Fp^(C.t)` (where `t` is the instance vector length defined below).

Table 1

The circuit `C : AbstractCircuit_Fp` in turn has the following form:

Circuit element	Description	Used in
`t`	Length of the instance vector.	
`n > 0`	Number of rows for the witness matrix.	
`m > 0`	Number of columns for the witness matrix.	
`≡`	An equivalence relation on `[0,m) x [0,n)`, indicating which witness entries are equal to each other.	Copy constraints (Section 3.4.2)
`S`	A set `S ⊆ ([0,m) x [0,n)) x [0,t)`, indicating which witness entries are equal to instance vector entries.	Copy constraints (Section 3.4.2)
`m_f <= m`	Number of columns that are fixed.	Fixed constraints (Section 3.4.1)
`f`	The fixed content of the first `m_f` columns, `f : Fp^(m_f x n)`.	Fixed constraints (Section 3.4.1)
`p_u`	Custom multivariate polynomials `p_u : Fp^m -> Fp`.	Custom constraints (Section 3.4.3)
`CUS_u`	Sets `CUS_u ⊆ [0,n)`, indicating rows on which the custom polynomials `p_u` are constrained to evaluate to `0`.	Custom constraints (Section 3.4.3)
`L_v`	Number of table columns in the lookup table with index `v, TAB_v`.	Lookup constraints (Section 3.4.4)
`TAB_v`	Lookup tables `TAB_v ⊆ Fp^{L_v}`, each with a number of tuples in `Fp^{L_v}`.	Lookup constraints (Section 3.4.4)
`q_{v,s}`	Scaling multivariate polynomials `q_{v,s} : Fp^m -> Fp` for `s in 0..L_v`.	Lookup constraints (Section 3.4.4)
`LOOK_v`	Sets `LOOK_v ⊆ [0,n)`, indicating rows on which the scaling polynomials `q_{v,s}` evaluate to some tuple in `TAB_v`.	Lookup constraints (Section 3.4.4)

Table 2

3.2. Witnesses

The relation `R_plonkish` takes witnesses of the following form:

Witness element	Description
`w`	The witness matrix `w : Fp^(m x n)`.

Table 3

Define `w_j` as the row vector `[ w[i, j] : i <- 0..m ]`.

3.3. Definition of the relation

Given the above definitions, the relation `R_plonkish` corresponds to a set of (instance, witness) pairs:

- `x`:
  - `Fp`
  - `C`:
    - `t`, `n`, `m`, `≡`, `S`, `m_f`, `f`
    - `[ (p_u, CUS_u) ]_u`
    - `[ (L_v, TAB_v, [q_{v,s}]_s, LOOK_v) ]_v`
  - `phi`
- `w`

such that:

Domains	Constraints
`w : Fp^(m × n)`, `f : Fp^(m_f × n)`	`i in [0, m_f), j in [0, n) => w[i, j] = f[i, j]`
`S ⊆ ([0, m) x [0, n)) x [0, t)`, `phi : Fp^t`	`((i, j), k) in S => w[i, j] = phi[k]`
`≡` `⊆ ([0, m) x [0, n)) x ([0, m) × [0, n))`	`(i, j) ≡ (k, l) => w[i, j] = w[k, l]`
`CUS_u ⊆ [0, n)`, `p_u : Fp^m -> Fp`	`j in CUS_u => p_u(w_j) = 0`
`LOOK_v ⊆ [0, n)`, `q_{v,s} : Fp^m -> Fp`, `TAB_v ⊆ Fp^{L_v}`	`j in LOOK_v => [ q_{v,s}(w_j) : s <- 0..L_v ] in TAB_v`

Table 4

In this model, a circuit-specific relation `R_{Fp, C}` for a field `Fp` and circuit `C` is the relation `R_plonkish` restricted to `( (Fp, C, phi : Fp^C.t), w : Fp^(C.m × C.n))`

3.4. Conditions satisfied by statements in `R_plonkish`

There are four types of constraints that a Plonkish statement `(x, w) in R_plonkish` must satisfy: * Fixed constraints * Copy constraints * Custom constraints * Lookup constraints

3.4.1. Fixed constraints

The first `m_f` columns of `w` are fixed to the columns of `f`.

3.4.2. Copy constraints

Copy constraints enforce that entries in the witness matrix are equal to each other, or that an instance entry is equal to a witness entry.

Copy Constraints	Description
`((i,j),k) in S => w[i, j] = phi[k]`	The `(i,j)` advice entry is equal to the `k` instance entry for all `((i,j),k) in S`.
`(i,j) ≡ (k,l) => w[i, j] = w[k, l]`	`≡` is an equivalence relation indicating which witness entries are constrained to be equal.

Table 5

By convention, when fixed abstract cells have the same value, we consider them to be equivalent under `≡`. That is, if `i < m_f && k < m_f && f[i, j] = f[k, l]` then `(i, j) ≡ (k, l)`.

This has no direct effect on the relation, but it will simplify expressing an optimization.

3.4.3. Custom constraints

Plonkish also allows custom constraints between the witness matrix entries. In the abstract model we are defining, a custom constraint applies only within a single row of the witness matrix, for the rows that are selected for that constraint.

In some systems using Plonkish, custom constraints are referred to as "gates".

Custom constraints enforce that witness entries within a row satisfy some multivariate polynomial. Here `p_u` could indicate any case that can be generated using a combination of multiplications and additions.

Custom Constraints	Description
`j in CUS_u => p_u(w_j) = 0`	`u` is the index of a custom constraint. `j` ranges over the set of rows `CUS_u` for which the custom constraint is switched on.

Table 6

Here `p_u : Fp^m -> Fp` is an arbitrary multivariate polynomial:

> Given η symbols `X_0, ..., X_{η-1}` called indeterminates, a multivariate polynomial `P` in these indeterminates with coefficients in `Fp` is a finite linear combination:
>
> `P(X_0, ..., X_{η-1}) = Σ_{z=0}^{ν-1} (c_z · Π_{b=0}^{η-1} X_b^{α_{z,b}})`
>
> where `c_z in Fp`, `c_z neq 0`, and `ν` and `α_{z,b}` are positive integers.

3.4.4. Lookup constraints

Lookup constraints enforce that some polynomial function of the witness entries on a row are contained in some table.

The sizes of tables are not limited at this layer. A realization of a proving system using Plonkish arithmetization may limit the supported size of tables, possibly depending on `n`, or it may have some way to compile larger tables.

In this specification, we only support fixed lookup tables determined in advance. This could be generalized to support dynamic tables determined by part of the witness matrix.

Lookup Constraints	Description
`j in LOOK_v => [ q_{v,s}(w_j) : s <- 0 .. L_v ] in TAB_v`	`v` is the index of a lookup table. `j` ranges over the set of rows `LOOK_v` for which the lookup constraint is switched on.

Table 7

Here `q_{v,s} : Fp^m -> Fp` for `s <- 0 .. L_v` are multivariate polynomials that collectively map the witness entries `w_j` on the lookup row `j in LOOK_v` to a tuple of field elements. This tuple will be constrained to match some row of the table `TAB_v`.

4. IANA Considerations

This document has no actions for IANA.

5. Informative References

[Thomas22] Thomas, M., "Arithmetization of Sigma relations in Halo 2", IACR ePrint Archive 2022/777, 2022, <https://eprint.iacr.org/2022/777>.

[ZKProofCommunityReference] ZKProof Community, "ZKProof Community Reference", 2023, <https://docs.zkproof.org/reference>.

Appendix A. Acknowledgements

Firstname Lastname, Firstname Lastname.

Author's Address

Firstname (Insert-Author-Name-here;-bug-707)
(Insert Author affiliation here)