TIP 0003: Subset Argument for Memory Consistency

TIP	0003
authors:	Alan Szepieniec
title:	Subset Argument for Memory Consistency
status:	draft
created:	2022-08-25
issue tracker:	
pdf:	tip-0003.pdf

Abstract. In the current specification, the memory-like tables `RamTable`, `JumpStackTable`, and `OpStackTable` do not satisfy memory-consistency. Specifically, they are vulnerable to Yuncong's attack, which exploits the unverified and thus possibly-incorrect sorting in these tables. TIP-0002 addresses one part of the issue by introducing a new table argument, the *multi-table subset argument*. It establishes that certain values in different tables have a matching representative in a given lookup table. Applied to the present context, this technique establishes that every clock jump is positive.

Unfortunately TIP-0002 introduces a lot of design complexity by. The cause of this complexity is the Polynomial IOP for correct calculation of the formal derivative.

This TIP proposes an alternative way to establish the correct calculation of the formal derivative, namely through a more traditional AIR/RAP construction. While there is a performance penalty of one extra polynomial commitment, the end-result is a noticeably simpler design.

TIPs 0002 and 0003 are mutually exclusive companions to TIP-0001, which introduces an new argument to establish the contiguity of regions of constant memory pointer. Together, TIP-0001 and (TIP-0002 or TIP-0003) fix the memory consistency issue.

Introduction

How to establish that a clock jump is directed forward (as opposed to direct backward, which would indicate malicious behavior)? One strategy is to show that the *difference*, *i.e.*, the next clock cycle minus the current clock cycle, is itself a clock cycle. Recall that the Processor Table's clock cycles run from 0 to $T-1$. Therefore, valid differences belong to $\{2, ..., T-1\}$ and invalid ones to $\{p-T+1, ..., p-2\}$.

Standard subset arguments can show that the clock jump differences are elements of the Processor Table's clock cycle column. However, it is cumbersome to repeat this argument for three separate tables.

I present here an AIR/RAP argument for showing that all clock jump differences in all three memory-like tables live also in the Processor Table's `clk` column. It introduces two extension columns in each memory-like table; two extension columns in Processor Table; and two committed polynomials that do not correspond to any columns.

Intuition

Let $f(X)$ denote the polynomial

$$f(X) = \left(\prod_{\delta_r}(X - \delta_r)\right) \cdot \left(\prod_{\delta_o}(X - \delta_o)\right) \cdot \left(\prod_{\delta_j}(X - \delta_j)\right)$$

where the δ_r, δ_o, and δ_j denote the clock jump differences in the Ram Table, OpStack Table, and Jump Stack Table, respectively.

This polynomial can be (and is) evaluated in a scalar α supplied by the verifier through three running products. (All three running products use the same challenge α.) However, it is additionally committed to. The point α is used to verify that the committed polynomial $[f(X)]$ does indeed correspond to the product of running product columns.

$$h(X) = \prod_{\delta}(X - \delta)$$

in some scalar β supplied by the verifier, where δ ranges over the values of `clk` in *indicated* rows. There does not need be an explicit indicator column; the prover knows when to accumulate the factor and when not to (and he can't cheat).

Note that the roots of $f(X)$ correspond to roots of $h(X)$. However, the roots occur with various multiplicities in $f(X)$ and with multiplicity 1 in $h(X)$. To harmonize the multiplicities, we need the formal derivative $f'(X)$ of $f(X)$ to cancel all the square factors. Specifically, let $g(X) = \gcd(f(X), f'(X))$ then $h(X)g(X) = f(X)$ and to establish the correct gcd relation we need Bézout coefficients $a(X)$ and $b(X)$ for the Bézout relation

$$a(X)f(X) + b(X)f'(X) = g(X).$$

Detailed Description

Memory-like Tables

Here are the constraints for the RAM Table. The constraints for other two tables are analogous and are therefore omitted. The constraints are relative to the indeterminate X which anticipates the verifier's challenge α via the implicit substitution $X \mapsto \alpha$.

Use mp to abstractly refer to the memory pointer. The extension column starts with a random initial rp_I and no boundary constraints. Before the substitution $X \mapsto \alpha$, rp_I is a uniformly random polynomial of degree at most 3 in X.

The transition constraint enforces the accumulation of a factor $(X - \mathsf{clk}^\star + \mathsf{clk})$ whenever there is a clock jump and the memory pointer is the same. If there is no clock jump or the memory pointer is changed, the same running product is carried to the next row. The transition constraint is

$$(1 - (\mathsf{mp}^\star - \mathsf{mp}) \cdot \mathsf{di}) \cdot (\mathsf{clk}^\star - \mathsf{clk} - 1) \cdot (\mathsf{rp}^\star - \mathsf{rp} \cdot (X - \mathsf{clk}^\star + \mathsf{clk}))$$

$$+ (\mathsf{mp}^\star - \mathsf{mp}) \cdot (\mathsf{rp}^\star - \mathsf{rp}) \quad .$$

Note that di is the difference inverse of the Ram Table but for the other two tables this factor can be dropped since the corresponding memory pointer can only change by either 0 or 1 between consecutive rows.

The running product has a terminal rp_T but a trivial terminal constraint: $\mathsf{rp} - \mathsf{rp}_T$.

The prover computes $\mathsf{rp}_T(X)$ symbolically for every memory-like table. The polynomial $f(X)$ is the product of all such symbolic terminals.

Processor Table

The Processor Table also has a running product extension column `rp`. In the first row it is unconstrained and set to a random initial $\mathsf{rp}_I \in \mathbb{F}$. In every row it is either updated with a factor $(X - \mathsf{clk})$ or not – the prover knows what to do because it knows which cycle counts happen to coincide with clock jump differences. The verifier only needs to verify the transition constraint $(\mathsf{rp}^\star - \mathsf{rp}) \cdot (\mathsf{rp}^\star - \mathsf{rp} \cdot (X - \mathsf{clk}))$, which stipulates that the running product is either updated correctly or remains the same. The terminal constraint is likewise $(\mathsf{rp}_T - \mathsf{rp}) \cdot (\mathsf{rp}_T - \mathsf{rp} \cdot (X - \mathsf{clk}))$

The formal indeterminate X here anticipates the verifier's second challenge β. The prover computes rp_T symbolically before β is supplied – in fact, this symbolic terminal is the polynomial $h(X)$.

Square-Freeness Argument

All memory-like tables' clock jump differences are members of the Processor Table's `clk` column if $h(X)$ is the maximal square-free divisor of $f(X)$. This relation is implied by three points:

- $f'(X)$ is the formal derivative of $f(X)$
- $g(X)$ is the greatest common divisor of $f(X)$ and $f'(X)$
- $h(X) = f(X)/g(X)$.

To see this, let $f(X) = \prod_i f_i(X)^{m_i}$. Then $f'(X) = \sum_i m_i f_i(X)^{m_i-1} f_i'(X) \prod_{j \neq i} f_j(X)$. And $g(X) = \prod_i f_i(X)^{m_i-1}$ because every factor in the product appears in all terms of $f'(X)$, but there will be one term that limits the multiplicity to $m_i - 1$. And from this it follows that $h(X) = f(X)/g(X)$.

Note that is is possible to eliminate $g(X) = f(X)/h(X)$ from the above bullet points. Therefore, two relations remain to be shown: the formal derivative, and the gcd.

Greatest Common Divisor

Let $a(X)$ and $b(X)$ be randomized Bézout coefficients such that

$$a(X)f(X) + b(X)f'(X) = g(X) \ \ .$$

Specifically, the prover computes coefficients $a^\star(X)$ and $b^\star(X)$ using the extended Euclidean algorithm. Then he samples a random factor $k \in \mathbb{F}$ and sets $a(X) = a^\star(X) + kf'(X)$ and $b(X) = b^\star(X) - kf(X)$.

The verifier probes this identity of polynomials in a randomly chosen point $z \xleftarrow{\$} \mathbb{F}$.

Formal Derivative

The formal derivative of a polynomial $f(X) = \sum_{i=0}^{d} c_i X^i$ is $f'(X) = \sum_{i=1}^{d} c_i i X^{i-1}$. The correct relation between two polynomial oracles $[f(X)]$ and $[f'(X)]$ can be verified by probing the relation $X \cdot f'(X) = (N_d \circ f)(X)$, where $N_d(X) = \sum_{i=1}^{d} iX^i$ is the natural arithmetic polynomial up to some degree bound $d \geq \deg(f(X))$, and where $\circ$ denotes the Hadamard (coefficient-wise) product.

Note that the natural arithmetic polynomial can be evaluated efficiently via the following recursive relation, which assumes that d is one less than a power of 2.

$$\begin{aligned} N_d(X) = \sum_{i=1}^{d} iX^i &= \sum_{i=0}^{\frac{d+1}{2}-1} iX^i + \sum_{i=0}^{\frac{d+1}{2}-1} (\frac{d+1}{2} + i) X^{\frac{d+1}{2}+i} \\ &= \sum_{i=0}^{\frac{d+1}{2}-1} iX^i + X^{\frac{d+1}{2}} \cdot \left(\frac{d+1}{2} \sum_{i=0}^{\frac{d+1}{2}-1} X^i + \sum_{i=0}^{\frac{d+1}{2}-1} iX^i \right) \\ &= \left(1 + X^{\frac{d+1}{2}}\right) \sum_{i=0}^{\frac{d+1}{2}-1} iX^i + \frac{d+1}{2} \cdot X^{\frac{d+1}{2}} \cdot \frac{X^{\frac{d+1}{2}} - 1}{X - 1} \end{aligned}$$

$$= \left(1 + X^{\frac{d+1}{2}}\right) N_{\frac{d+1}{2}-1}(X) + \frac{d+1}{2} \cdot \frac{X^{d+1} - X^{\frac{d+1}{2}}}{X-1}$$

Therefore, no further polynomial oracles are required to verify the formal derivative relation beyond those that are required to verify the Hadamard relation. For verifying the Hadamard relation, the §3.5 of the Claymore paper provides a solution, which is summarized below in unrolled form.

- The verifier samples $\alpha \xleftarrow{\$} \mathbb{F}\backslash\{0\}$ and sends α to the prover.
- The prover computes the coefficient vector $\mathbf{c}$ of $N_d(\alpha \cdot X) \cdot f(X^{-1}) \cdot X^d$ and the polynomial $\bar{h}(X) = \sum_{i=0}^{2d} \bar{h}_i X^i$, where the i th coefficient is $\bar{h}_i = \frac{c_i}{\gamma^d - \gamma^i}$ and $\bar{h}_d \xleftarrow{\$} \mathbb{F}$. The prover sends $\bar{h}(X)$ to the verifier along with the implicit claim that its degree is bounded by $2d$. Here $\gamma \in \mathbb{F}$ is some system-wide parameter with a sufficiently large multiplicative order.
- The verifier probes the polynomial identity $\bar{h}(X) \cdot \gamma^d - \bar{h}(\gamma X) + \alpha \cdot f'(\alpha) \cdot X^d = N_d(\alpha \cdot X) \cdot f(X^{-1}) \cdot X^d$ in a random point $z \xleftarrow{\$} \mathbb{F}\backslash\{0\}$.

Putting Everything Together

1. The prover computes $f(X)$, $h(X)$, $f'(X), a(X), b(X)$ and all base columns.
2. The prover commits to $f(X)$, $a(X)$, $b(X)$ and to all base columns.
3. The verifier supplies uniformly random challenges α, β, z.
4. The prover computes the extension columns, and $\bar{h}(X)$, and commits to them. He also reveals the terminals.
5. Let α be the challenge for the Processor Table's running product such that $h(\alpha) = \mathsf{rp}_T$. The prover reveals $a(\alpha), b(\alpha), f(\alpha)$. Note that $f'(\alpha) = -\frac{a(\alpha)f(\alpha) - f(\alpha)/h(\alpha)}{b(\alpha)}$.
6. The prover reveals $f(z^{-1})$, $\bar{h}(z)$ and $\bar{h}(\gamma z)$.
7. The verifier probes the polynomial identity $\bar{h}(X) \cdot \gamma^d - \bar{h}(\gamma X) + \alpha \cdot f'(\alpha) \cdot X^d = N_d(\alpha \cdot X) \cdot f(X^{-1}) \cdot X^d$ in $X = z$ with the revealed values.
8. Let β be the challenge for computing the running products of the memory-like tables, resulting in three terminals which, multiplied together, give $f(\beta)$. Let $i_f(X)$ be the lowest-degree interpolant through $(\alpha, f(\alpha)), (\beta, f(\beta)), (z^{-1}, f(z^{-1}))$, and $i_{\bar{h}}(X)$ the lowest-degree interpolant through $(z, \bar{h}(z)), (\gamma z, \bar{h}(\gamma z))$. Add to the nonlinear combination:

- the polynomials $f(X), a(X), b(X), \bar{h}(X)$ of degrees at most $T+9$, $T+8$, $T+9$, and $2T+18$, respectively.
- the quotient $\frac{f(X) - i_f(X)}{(X-\alpha)(X-\beta)(X-z^{-1})}$ of degree at most $T+6$
- the quotient $\frac{a(X) - a(\alpha)}{X-\alpha}$ of degree at most $T+7$
- the quotient $\frac{b(X) - b(\alpha)}{X-\alpha}$ of degree at most $T+8$.
- the quotient $\frac{\bar{h}(X) - i_{\bar{h}}(X)}{(X-z)(X-\gamma z)}$ of degree at most $2T+16$.

The reason why $f(X)$ has degree at most $T+9$, not $3T+9$ as one might expect, is because any instruction can affect only one memory-like table. As a result, every cycle generates at most one clock jump.

Security

The claim is that within each contiguous region of constant memory pointer, the clock cycle column is sorted in ascending order. The AIR verifies increments by one. Therefore the claim boils down to this: increments by more than one within the same contiguous region – *clock jumps* – increase the clock cycle column by some value in $\{2, \ldots, T\}$.

The entire technique reduces this claim to an identity of polynomials. Specifically, the polynomial $\prod_\delta (X - \delta)$, where δ ranges over all clock jump differences, after removing all factors with multiplicity greater than one, must be equal to the polynomial $\prod_i (X - \mathsf{clk}_i)$, where the product ranges over all clock cycles that correspond to some clock jump difference.

Completeness

Completeness follows from construction, except in special cases where α. Note that if α coincides with one of the roots of the initials of one of the memory-like tables, then $f(X) = f'(X) = 0$ but this degeneration does not affect completeness. However, if two of the initials have a nontrivial common divisor, then $f'(X)$ will contain a factor that $h(X)$ does not contain. In this case the prover fails.

What is the probability that the initials have non-trivial common factors? The first polynomial has 3 linear factors over an appropriate extension and the probability that the second is zero modulo any one of them is at most $3/|\mathbb{F}|$. The probability the probability that the third polynomial is zero modulo any one of the linear factors of either the first or second polynomial is at most $6/|\mathbb{F}|$. By the union bound, the probability of the initials sharing a non-trivial factor is therefore $9/|\mathbb{F}|$. This quantity is the completeness error. (Recall that $\mathbb{F}$ is the extension field, not the base field.) □

Soundness

The square-freeness argument generates a false positive when $a(X)f(X) + b(X)f'(X) \neq g(X)$ except in the point $X = \alpha$ where it was sampled. The degree of this polynomial identity is $2T + 17$, and so by the Schwarz-Zippel lemma the probability of false positive is at most $(2T + 17)/|\mathbb{F}|$.

The formal derivative argument boils down to a Hadamard product, whose soundness error is bounded by $3(T + 9)/|\mathbb{F}|$. By the union bound, the soundness error of the entire construction is bounded by $(5T + 26)/|\mathbb{F}|$. □

Zero-Knowledge

The prover releases 4 terminals $\mathsf{rp}_T^{(r)}$, $\mathsf{rp}_T^{(o)}$, $\mathsf{rp}_T^{(j)}$, $\mathsf{rp}_T^{(p)}$, which are the running products of the three memory-like tables and the Processor Table, respectively. In addition to that, the prover also releases the evaluations $f(\alpha)$, $a(\alpha)$, $b(\alpha)$, $f(z^{-1})$, $\bar{h}(z)$, $\bar{h}(\gamma z)$.

For every tuple $\mathsf{rp}_T^{(r)}$, $\mathsf{rp}_T^{(o)}$, $\mathsf{rp}_T^{(j)}$, $f(\alpha)$, $f(z^{-1})$, there are consistent random initials $\mathsf{rp}_I^{(r)}$, $\mathsf{rp}_I^{(o)}$, $\mathsf{rp}_I^{(j)}$.

- The mapping $\mathsf{rp}_{I,0}^{(o)} \mapsto \mathsf{rp}_T^{(o)}$ is affine and the coefficient is the product $\prod(\beta - \delta^{(o)})$ where $\delta^{(o)}$ ranges over all clock jumps in the OpStack table. This coefficient is zero with negligible probability.
- The mapping $\mathsf{rp}_{I,0}^{(j)} \mapsto \mathsf{rp}_T^{(j)}$ is affine and the coefficient is the product $\prod(\beta - \delta^{(j)})$ where $\delta^{(j)}$ ranges over all clock jumps in the JumpStack table. This coefficient is zero with negligible probability.
- The mapping $(\mathsf{rp}_{I,0}^{(r)}, \mathsf{rp}_{I,1}^{(r)}, \mathsf{rp}_{I,2}^{(r)}, \mathsf{rp}_{I,3}^{(r)}) \mapsto (f(\beta), f(\alpha), f(z^{-1}), f(\gamma^{-1}z^{-1}))$ is invertible with high probability because it is affine and its coefficient matrix is Vandermonde. A singular Vandermonde matrix requires α, β, z^{-1}, $\gamma^{-1}z^{-1}$ to coincide or to have low order, which occurs with negligible probability.
    - The mapping $f(\beta) \mapsto \mathsf{rp}_T^{(r)}$ is invertible with high probability because the factor $\mathsf{rp}_T^{(o)}\mathsf{rp}_T^{(j)}$ is fixed and nonzero with overwhelming probability.
    - The mapping $f(\gamma^{-1}z^{-1}) \mapsto \bar{h}(\gamma z) = N_d(\alpha \cdot \gamma z) \cdot f(\gamma^{-1}z^{-1}) \cdot \gamma^d z^d$ is linear and the coefficient is nonzero with overwhelming probability.

For every set of initials fixed so far, for every evaluation $\bar{h}(z)$ there is a consistent random coefficient $\bar{h}_d$. The mapping $\bar{h}_d \mapsto \bar{h}(z)$ is affine and the coefficient if z^d, which is invertible with overwhelming probability.

For every terminal $\mathsf{rp}_T^{(p)}$ there is a consistent initial $\mathsf{rp}_I^{(p)}$ because the mapping $\mathsf{rp}_I^{(p)} \mapsto \mathsf{rp}_T^{(p)}$ is linear and the coefficient is given by the product $\prod(\alpha - \mathsf{clk}_i)$ which ranges over all relevant clock cycles. This coefficient is nonzero with overwhelming probability.

For every set of initials, and for every evaluation $a(\alpha)$ there is a consistent Bézout randomizer k. The mapping $k \mapsto a(\alpha)$ is affine and the coefficient is $f'(\alpha)$, which is nonzero with overwhelming probability.

The evaluation $b(\alpha)$ is fixed by the relation

$$\bar{h}(z) \cdot \gamma^d - \bar{h}(\gamma z) + \alpha \cdot \frac{a(\alpha)f(\alpha) - f(\alpha)/h(\alpha)}{b(\alpha)} \cdot z^d = N_d(\alpha \cdot z) \cdot f(z^{-1}) \cdot z^d$$

and therefore does not leak any new information. □

Memory-Consistency

This section shows that TIPs 0001, 0002, and the permutation arguments that are already present in the documentation, jointly imply memory-consistency.

Whenever the Processor Table reads a value "from" a memory-like table, this value appears nondeterministically and is unconstrained by the base table AIR constraints. However, there is a permutation argument that links the Processor Table to the memory-like table in question. *The construction satisfies memory-consistency if it guarantees that whenever a memory cell is read, its value is consistent with the last time that cell was written.*

The above is too informal to provide a meaningful proof for. Let's put formal meanings on the proposition and premises, before reducing the former to the latter.

Let P denote the Processor Table and M denote the memory-like table. Both have height T. Both have columns `clk`, `mp`, and `val`. For P the column `clk` coincides with the index of the row. P has another column `ci`, which contains the current instruction, which is `write`, `read`, or `any`. Obviously, `mp` and `val` are abstract names that depend on the particular memory-like table, just like `write`, `read`, and `any` are abstract instructions that depend on the type of memory being accessed. In the following math notation we use `col` to denote the column name and *col* to denote the value that the column might take in a given row.

Definition 1 (contiguity): The memory-like table is *contiguous* iff all sublists of rows with the same memory pointer `mp` are contiguous. Specifically, for any given memory pointer *mp*, there are no rows with a different memory pointer *mp'* in between rows with memory pointer *mp*.

$$\forall i < j < k \in \{0, \dots, T-1\} : mp \triangleq M[i][\texttt{mp}] = M[k][\texttt{mp}] \Rightarrow M[j][\texttt{mp}] = mp$$

Definition 2 (regional sorting): The memory-like table is *regionally sorted* iff for every contiguous region of constant memory pointer, the clock cycle increases monotonically.

$$\forall i < j \in \{0, \dots, T-1\} : M[i][\texttt{mp}] = M[j][\texttt{mp}] \Rightarrow M[i][\texttt{clk}] <_{\mathbb{Z}} M[j][\texttt{clk}]$$

The symbol $<_{\mathbb{Z}}$ denotes the integer less than operator, after lifting the operands from the finite field to the integers.

Definition 3 (memory-consistency): A Processor Table P has *memory-consistency* if whenever a memory cell at location *mp* is read, its value corresponds to the previous time the memory cell at location *mp* was written. Specifically, there are no writes in between the write and the read, that give the cell a different value.

$$\forall k \in \{0, \ldots, T-1\} : P[k][\texttt{ci}] = \textit{read} \Rightarrow ((1) \Rightarrow (2))$$

$$(1)\exists i \in \{0, \ldots, k\} : P[i][\texttt{ci}] = \textit{write} \wedge P[i{+}1][\texttt{val}] = P[k][\texttt{val}] \wedge P[i][\texttt{mp}] = P[k][\texttt{mp}]$$

$$(2)\nexists j \in \{i+1, \ldots, k-1\} : P[j][\texttt{ci}] = \textit{write} \wedge P[i][\texttt{mp}] = P[k][\texttt{mp}]$$

Theorem 1 (memory-consistency): Let P be a Processor Table. If there exists a memory-like table M such that

- selecting for the columns `clk`, `mp`, `val`, the two tables' lists of rows are permutations of each other; and
- M is contiguous and regionally sorted;

then P has memory-consistency.

Proof. For every memory pointer value *mp*, select the sublist of rows $P_{mp} \triangleq \{P[k] \mid P[k][\texttt{mp}] = mp\}$ in order. The way this sublist is constructed guarantees that it coincides with the contiguous region of M where the memory pointer is also *mp*.

Iteratively apply the following procedure to P_{mp}: remove the bottom-most row if it does not correspond to a row k that constitutes a counter-example to memory consistency. Specifically, let i be the clock cycle of the previous row in P_{mp}.

- If i satisfies (1) then by construction it also satisfies (2). As a result, row k is not part of a counter-example to memory-consistency. We can therefore remove the bottom-most row and proceed to the next iteration of the outermost loop.
- If $P[i][\texttt{ci}] \neq \textit{write}$ then we can safely ignore this row: if there is no clock jump, then the Processor Table's AIR constraints guarantee that *val* cannot change; and if there is a clock jump, then the memory-like table's AIR constraints guarantee that *val* cannot change. So set i to the clock cycle of the row above it in P_{mp} and proceed to the next iteration of the inner loop. If there are no rows left for i to index, then there is no possible counterexample for k and so remove the bottom-most row of P_{mp} and proceed to the next iteration of the outermost loop.
- The case $P[i+1][\texttt{val}] \neq P[k][\texttt{val}]$ cannot occur because by construction of i, *val* cannot change.
- The case $P[i][\texttt{mp}] \neq P[k][\texttt{mp}]$ cannot occur because the list was constructed by selecting only elements with the same memory pointer.

- This list of possibilities is exhaustive.

When P_{mp} consists of only two rows, it can contain no counter-examples. By applying the above procedure, we can reduce every correctly constructed sublist P_{mp} to a list consisting of two rows. Therefore, for every mp, the sublist P_{mp} is free of counter-examples to memory-consistency. Equivalently, P is memory-consistent. $\square$