

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 $\{1, \dots, T - 1\}$ and invalid ones to $\{p - T + 1, \dots, p - 1\}$.

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 - one extension column in each memory-like table; - three extra base columns in the Processor Table; and - three extension columns in the Processor Table.

Intuition

- A multi-table permutation argument establishes that all clock jump differences are contained in a new column `cjd` of the Processor Table. Every memory-like table needs one extension column and the Processor Table needs one matching extension column to effect this permutation argument.
- An inverse column `invm` in the Processor Table allows for selecting all *nonzero* `cjd`'s, thus skipping padding rows.
- An inverse column `invu` in the Processor Table allows for selecting the last row of *every* contiguous region of `cjd`.
- An evaluation argument establishes that a selection of clock cycles and the selected clock jump differences are identical lists. This evaluation argument requires two more extension columns on the Processor Table.

The total is 3 extra base columns and 6 extension columns.

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 from this section. Elsewhere the superscripts ^(r), ^(j) and ^(o) disambiguate. 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 first extension column, `rp`, computes a running product. It starts with a random initial $\text{rp}_I \xleftarrow{\$} \mathbb{F}$.

The transition constraint enforces the accumulation of a factor $(X - \text{clk}^* + \text{clk})$ whenever there is a clock jump and the memory pointer is the same. If the memory pointer is changed, the same running product is carried to the next row. The transition constraint is

$$(1 - (\text{mp}^* - \text{mp}) \cdot \text{di}) \cdot (\text{rp}^* - \text{rp} \cdot (X - \text{clk}^* + \text{clk})) \\ + (\text{mp}^* - \text{mp}) \cdot (\text{rp}^* - \text{rp}) \ .$$

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.

Clock Jump Differences with Multiplicities

The clock jump differences of the memory-like tables are all listed in the `cjd` column of the Processor Table. The values are sorted and the padding inserts “0” rows at the bottom. This relation comes with another extension column computing a running product. Denote this column by `rp`.

The initial value of this column is the product of the initial values of the matching columns of the Ram Table, JumpStack Table, and OpStack Table. This gives rise to the following cross-table initial boundary constraint:

$$rp - rp^{(r)} \cdot rp^{(j)} \cdot rp^{(o)}$$

This running product accumulates a factor $(X - \text{cjd})$ in every row where $\text{cjd} \neq 0$. Assume there is a column `invm` which is which is the inverse-or-zero of `cjd`. Then the transition constraint is

$$\text{cjd} \cdot (rp^* - rp \cdot (X - \text{cjd})) + (1 - \text{invm} \cdot \text{cjd}) \cdot (rp^* - rp) \cdot$$

The consistency constraints for the inverse are $-\text{cjd}^2 \cdot \text{invm} - \text{cjd}$, and $-\text{cjd} \cdot \text{invm}^2 - \text{invm}$.

The terminal value of this column must be equal to the terminal values of the matching running products of the memory-like tables. The terminal boundary constraint is therefore identical to the initial boundary constraint:

$$rp - rp^{(r)} \cdot rp^{(j)} \cdot rp^{(o)}$$

Relevant Clock Jumps

Assume there is an indicator `ind` which is 1 whenever the current row of the Processor Table corresponds with a `cjd` that is the last in a contiguous region. Using this indicator, build a running evaluation that accumulates one step of evaluation relative to `cjd` for each contiguous region, excluding the padding region.

The initial value is a random scalar. The running evaluation accumulates one step of evaluation whenever the indicator bit is set, and does not modify the running evaluation otherwise. The following transition constraint captures this transition.

$$(1 - \text{ind}) \cdot (re^* - re) + \text{ind} \cdot (re^* - \beta \cdot re - \text{cjd})$$

To verify that the indicator is correctly indicating the last row of every contiguous region, we need another column, `invu` which contains the inverse-or-zero of

every consecutive pair of cjd values. This induces two transition constraints: $-\text{invu}^2 \cdot (\text{cjd} - \text{cjd}^*) - \text{invu} - \text{invu} \cdot (\text{cjd} - \text{cjd}^*)^2 - (\text{cjd} - \text{cjd}^*)$

Then we can use the invu column to simulate the indicator bit via $\text{ind} = \text{invu} \cdot (\text{cjd} - \text{cjd}^*)$.

Relevant Clock Cycles

Assume the prover knows when the clock cycle clk is also *some* jump and when it's not. Then it can apply the right running evaluation step as necessary.

Specifically, the initial value is identical to the running evaluation of “Relevant Clock Jumps”. This gives rise to an initial boundary constraint: $\text{re}^{(\text{clk})} - \text{re}^{(\text{cjd})}$.

In every row, the running evaluation step is either applied or else the running evaluation remains the same.

$$(\text{re}^* - \text{re}) \cdot (\text{re}^* - \beta \cdot \text{re} - \text{clk})$$

The terminal value must be identical to the running evaluation of “Relevant Clock Jumps”. This gives rise to a terminal boundary constraint that is identical to the initial boundary constraint: $\text{re}^{(\text{clk})} - \text{re}^{(\text{cjd})}$.

The indicator does not need to be constrained since if the prover fails to indicate certain rows he will have a harder time not easier time convincing the verifier.

Memory-Consistency

This section shows that TIPs 0001, 0003 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 \mathbf{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 .

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

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.

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

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.

$$\forall k \in \{0, \dots, T-1\} : P[k][\mathbf{ci}] = \text{read} \Rightarrow ((1) \Rightarrow (2))$$

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

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

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

- selecting for the columns \mathbf{clk} , \mathbf{mp} , \mathbf{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 $P_{mp} \stackrel{\Delta}{=} \{P[k] \mid P[k][\mathbf{mp}] = mp\}$ 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 $P[i][\text{ci}] \neq \text{write}$ 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 $P[i+1][\text{val}] \neq P[k][\text{val}]$ cannot occur because by construction of i , val cannot change.
- The case $P[i][\text{mp}] \neq P[k][\text{mp}]$ 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