Hypersafety Proofs for Structured Data Manipulations: Appendix

ANONYMOUS AUTHOR(S)

A A COMMENTARY FOR THE MECHANISED PROOF OF RL SUMMATION

We have mechanised the case study in Sections 1 and 2 of the manuscript in Coq. Our mechanised proof mainly follows the proof outline of Section 2, and it is modular in the sense that each important intermediate proof step, represented as hyper-triple here, as a lemma.

We have attached a navigation guide of our mechanisation in the accompanying Coq project directory for illustrating the correspondence between the results in the paper and the lemmas proved in Coq. Here we only provide a general, conceptual explanation of the differences between our mechanised proof and the textual proof.

Some technical details that are present in the Coq proof, but not in the paper:

- An array not only contains a sequence of memory blocks, but also includes a header that stores its length information. In the semantics, applying the length operator on the header will return its content.
- Functions are written in a form similar to that of an intermediate language resembling three-address code, in order to fit the operational semantics.

Some changes in the Coq formalisation that are made to reduce the reasoning burden:

- The lengths of arrays x_{ind}, x_{val} are hard-coded by assuming them as existing variables. In other words, they are not obtained via the length operator. Similarly for their contents.
- The local variables (*e.g.*, `i` in `rlsum`) are freed before the function returns. This change is made since reasoning about $\top$ would be complicated when applying the MERGE rule.

B CASE STUDY: SPARSE TENSOR MULTIPLICATION (FULL PROOF)

Specifying sparse matrix/vector multiplication. To specify the result of spmspv compactly, let us introduce some convenient notation. First, we will use **0** to denote a 0-filled vector of a suitable size evident from the context. Second, we will abbreviate the joint assertion about all required arrays across multiple components, which are not being modified, simply as *Arrs*:

$$Arrs \triangleq \mathtt{arr}(\mathtt{m_{ind}}(\cdot),\ m_{ind}) \ast \mathtt{arr}(\mathtt{m_{val}}(\cdot),\ m_{val}) \ast \mathtt{arr}(\mathtt{x_{ind}}(\cdot),\ x_{ind}) \ast \mathtt{arr}(\mathtt{x_{val}}(\cdot),\ x_{val})$$

The desired specification is encoded by the following hypersafety triple:

$$\left\{\begin{array}{c}\mathtt{arr}(\mathtt{ans}(1),\ \mathbf{0}) \\ \ast\ Arrs\end{array}\right\} \left[\begin{array}{rl} 1 : & \mathtt{spmspv}(\mathtt{m_{ind}}, \mathtt{m_{val}}, \mathtt{x_{ind}}, \mathtt{x_{val}}, \mathtt{ans}) \\ \langle 2, i, j\rangle_{\substack{i \in [0, M) \\ j \in [0, N)}} : & \mathtt{csrij}(\mathtt{m_{ind}}, \mathtt{m_{val}}, i, j) \\ \langle 3, j\rangle_{j \in [0, N)} : & \mathtt{svi}(\mathtt{x_{ind}}, \mathtt{x_{val}}, i) \end{array}\right] \left\{\begin{array}{c|l} - & \exists a, \mathtt{arr}(\mathtt{ans}(1),\ a) \ast \\ \overline{m} & \ulcorner \forall i, a[i] = \sum_j \overline{m}(i,j)\overline{x}(j) \urcorner \\ \overline{x} & \ast\ Arrs \ast \top \end{array}\right\} \quad (1)$$

The first line of the post-condition states that the result of the computation stored in the array ans of the first component is a dense vector a of the size M, while the second line asserts, in the common mathematical notation, that a is a result of multiplying the dense representation of the matrix $\overline{m}$ by the corresponding dense vector $\overline{x}$. We slightly abuse the hyper-value notation by referring to elements of $\overline{m}$ in the post-condition as $\overline{m}(i, j)$, assuming that the natural lexicographic order on the elements of the corresponding index set $\{\langle 2, i, j\rangle \mid i \in [0, M), j \in [0, N)\}$.

The proof is best explained in two steps, each focusing on one of its loops.

2022. 2475-1421/2022/1-ART1 $15.00
https://doi.org/

Step 1: Top-level `for`*-loop decomposition.* The body of `spmspv` is represented by a loop that iterates through all rows of the matrix m via the variable `i`:

$$\left\{\begin{array}{c}\mathsf{arr}(\mathsf{ans}(1),\ \mathbf{0})\\ *\ \mathit{Arrs}\end{array}\right\}\left[\begin{array}{rl} 1: & \mathtt{for}\ i\ \mathtt{in}\ \mathtt{range}(0,|m_{ind}|)\ \ \{p(i)\}\\ \langle 2,i,j\rangle_{\substack{i\in[0,M)\\ j\in[0,N)}} & : \mathtt{csrij}(\mathtt{m_{ind}},\mathtt{m_{val}},i,j)\\ \langle 3,j\rangle_{j\in[0,N)} & : \mathtt{svi}(\mathtt{x_{ind}},\mathtt{x_{val}},j)\end{array}\right]\left\{\begin{array}{c|c} - & \exists a, \mathsf{arr}(\mathsf{ans}(1),\ a) * \\ \overline{m} & \ulcorner \forall i, a[i] = \sum_j \overline{m}(i,j)\overline{x}(j)\urcorner\\ \overline{x} & * \mathit{Arrs} * \top\end{array}\right\} \tag{2}$$

Each iteration of the `for`-loop handles exactly one row of m, computing the value of multiplying it by all values of x, encoded via `x_ind` and `x_val`. To proceed with our proof, we employ the general form of the For rule. $I_{\mathtt{for}}(i,\overline{mx})$ for the `for`-loop as follows:

$$I_{\mathtt{for}}(i,\overline{mx}) \triangleq \mathop{\huge\ast}_{k=0}^{i-1} \mathsf{ans}(1)+k \mapsto \sum_j \overline{m}(i,j)\overline{x}(j) \ * \mathop{\huge\ast}_{k=i}^{M-1} \mathsf{ans}(1)+k \mapsto 0$$

Notice that the invariant above only constraints the hyper-state relevant to the component 1. Next, we represent the index sets of the format components as the following two unions:

$$\left\{\langle 2,i,j\rangle \;\middle|\; \begin{array}{c} i\in[0,M)\\ j\in[0,N)\end{array}\right\} = \biguplus_{i=0}^{M-1}\{\langle 2,i,j\rangle \ \mid j\in[0,N)\}$$

$$\{\langle 3,j\rangle \ \mid j\in[0,N)\} = \bigcup_{0}^{M-1}\{\langle 3,j\rangle \ \mid j\in[0,N)\}$$

That is, while the first set has been split into M disjoint components, the second has been represented as a union of its own M identical copies, as we will need it for *each* of M iterations of the `for`-loop. Choosing the invariant $I_{\mathtt{for}}$ and the split leads to the following proof obligation, after framing out all the unchanged components of the state in the invariant:

$$\left\{\begin{array}{c}\mathsf{ans}(1)+i\mapsto 0\\ *\ \ldots\end{array}\right\}\left[\begin{array}{cl} 1 & : p(i)\\ \langle 2,i,j\rangle_{j\in[0,N)} & : \mathtt{csrij}(\mathtt{m_{ind}},\mathtt{m_{val}},i,j)\\ \langle 3,j\rangle_{j\in[0,N)} & : \mathtt{svi}(\mathtt{x_{ind}},\mathtt{x_{val}},j)\end{array}\right]\left\{\begin{array}{c|c} - & \\ \overline{m} & \mathsf{ans}(1)+i\mapsto \sum_j \overline{m}(i,j)\overline{x}(j)\\ \overline{x} & *\ \ldots\end{array}\right\} \tag{3}$$

In the goal the implicitly universally quantified variable i ranges from 0 up to $|m_{ind}|-1$. Since i is fixed across all the programs in the second component product, let us re-index it, simultaneously rewriting the call to `csrij` with its definition (also notice a subtle change in using $\overline{m}$ in the postcondition):

$$\left\{\begin{array}{c}\mathsf{ans}(1)+i\mapsto 0\\ *\ \ldots\end{array}\right\}\left[\begin{array}{cl} 1 & : p(i)\\ \langle 2,j\rangle_{j\in[0,N)} & : \mathtt{svi}(\mathtt{m_{ind}}[\mathtt{i}],\mathtt{m_{val}}[\mathtt{i}],j)\\ \langle 3,j\rangle_{j\in[0,N)} & : \mathtt{svi}(\mathtt{x_{ind}},\mathtt{x_{val}},j)\end{array}\right]\left\{\begin{array}{c|c} - & \\ \overline{m} & \mathsf{ans}(1)+i\mapsto \sum_j \overline{m}(j)\overline{x}(j)\\ \overline{x} & *\ \ldots\end{array}\right\} \tag{4}$$

Progressing beyond line 3 of `spmspv` and allocating the variable `iM` and `iX` leaves with the goal:

$$\left\{\begin{array}{c}\mathsf{ans}(1)+i\mapsto 0\ *\\ \mathtt{iX}\mapsto 0\ *\ \mathtt{iM}\mapsto 0\ *\\ \ldots\end{array}\right\}\left[\begin{array}{cl} 1 & : \mathtt{while}\ (\ldots)\{\ldots\}\\ \langle 2,j\rangle_{j\in[0,N)} & : \mathtt{svi}(\mathtt{m_{ind}}[\mathtt{i}],\mathtt{m_{val}}[\mathtt{i}],j)\\ \langle 3,j\rangle_{j\in[0,N)} & : \mathtt{svi}(\mathtt{x_{ind}},\mathtt{x_{val}},j)\end{array}\right]\left\{\begin{array}{c|c} - & \\ \overline{m} & \mathsf{ans}(1)+i\mapsto \sum_j \overline{m}(j)\overline{x}(j)\\ \overline{x} & *\ \ldots\end{array}\right\} \tag{5}$$

First we consider a case when at least one of x_{ind} or $m_{ind}[i]$ is empty, (say x_{ind} then), `while`-loop condition will be immediately false. More over using Prod and the following specification for `svi`

$$\left\{\mathsf{arr}(\mathsf{ind},\ \mathit{ind})\ *\ \mathsf{arr}(\mathsf{val},\ \mathit{val})\ *\ \ulcorner j\notin \mathit{ind}\urcorner\right\}\left[\,e : \mathsf{svi}(\mathsf{ind},\mathsf{val},j)\right]\left\{\,x\ \middle|\ \ulcorner x = 0\urcorner\ *\ \ldots\ \right\}$$

which can be easily proven as a regular separation logic property, we conclude

$$\left\{\begin{array}{c}\mathsf{ans}(1)+i\mapsto 0\ *\\ \mathtt{iX}\mapsto 0\ *\ \mathtt{iM}\mapsto 0\ *\\ \ldots\end{array}\right\}\left[\langle 3,j\rangle_{j\in[0,N)} : \mathtt{svi}(\mathtt{x_{ind}},\mathtt{x_{val}},j)\ \right]\left\{\overline{x}\ \middle|\ulcorner\overline{x} = \mathbf{0}\urcorner\ *\ \ldots\right\} \tag{6}$$

Hence we can reduce (7) to

$$\left\{ \begin{array}{c} \mathtt{ans(1)} + i \mapsto 0 \; * \\ \mathtt{iX} \mapsto 0 \; * \; \mathtt{iM} \mapsto 0 \; * \\ \ldots \end{array} \right\} \left[\langle 3, j \rangle_{j \in [0,N)} : \mathtt{svi}(\mathtt{x_{ind}}, \mathtt{x_{val}}, j) \right] \left\{ \begin{array}{c} - \\ \overline{m} \\ \overline{x} \end{array} \middle| \begin{array}{c} \mathtt{ans(1)} + i \mapsto 0 \\ * \; \ldots \end{array} \right\} \tag{7}$$

Which immediately follows form svi termination (it is a for-loop under the hood). To proceed in the case when both x_{ind} and $m_{ind}[i]$ are both non-empty, let us make an important observation. Remember that both arrays $m_{ind}[i]$ and x_{ind} contain indices of non-zero elements of the corresponding dense vectors. Therefore any elements of those vectors positioned at indices larger than $st = \min(\max(m_{ind}[i]), \max(x_{ind}))$ will *not* contribute to the sum stored in ans(1) due to either being zero or being multiplied by zero. This fact can be phrased as the following hyper-safety triple:

$$\left\{ \ldots \right\} \left[\begin{array}{l} \langle 2, j \rangle_{st < j < N} : \mathtt{svi}(\mathtt{m_{ind}[i]}, \mathtt{m_{val}[i]}, j) \\ \langle 3, j \rangle_{st < j < N} : \mathtt{svi}(\mathtt{x_{ind}}, \mathtt{x_{val}}, j) \end{array} \right] \left\{ \begin{array}{c} \overline{m} \\ \overline{x} \end{array} \middle| \ulcorner \forall j > st, \; \overline{m}(j)\overline{x}(j) = 0 \urcorner \right\} \tag{8}$$

Using the rule SeqU we can, therefore, reduce (7) to the following goal:

$$\left\{ \begin{array}{c} \mathtt{ans(1)} + i \mapsto 0 \; * \\ \mathtt{iX} \mapsto 0 \; * \; \mathtt{iM} \mapsto 0 \; * \\ \ldots \end{array} \right\} \left[\begin{array}{l} 1 : \mathtt{while} \; (\mathtt{cond(iM, iX)}) \, \{\ldots\} \\ \langle 2, j \rangle_{j \le st} : \mathtt{svi}(\mathtt{m_{ind}[i]}, \mathtt{m_{val}[i]}, j) \\ \langle 3, j \rangle_{j \le st} : \mathtt{svi}(\mathtt{x_{ind}}, \mathtt{x_{val}}, j) \end{array} \right] \left\{ \begin{array}{c} - \\ \overline{m} \\ \overline{x} \end{array} \middle| \begin{array}{c} \mathtt{ans(1)} + i \mapsto \sum_j \overline{m}(j)\overline{x}(j) \\ * \; \ldots \end{array} \right\} \tag{9}$$

where cond(iM, iX) abbreviates the actual condition of the while-loop.

Step 2: Verifying the while*-loop.* We are going to use the general While rule to verify the goal (9). Recall that this rule in LGTM provides access to the *counting* aspect of a while-loop. This is thanks to the second parameter s of its invariant of the shape $I_{\mathtt{wl}}(b, s, \overline{v})$ that "keeps track" of the number of (format-related) "auxiliary" program components used to verify the previous loop iterations. The first invariant parameter, b, keeps the status of the loop condition, while the last, $\overline{v}$ accumulates the hyper-value containing the results of the auxiliary programs executed in alignment with the past iterations. To make use of the rule for our example, we choose the following invariant:

$$\begin{aligned} I_{\mathtt{wl}}(b, s, \overline{mx}) \triangleq \exists i_m, i_x, \; & \mathtt{iM} \mapsto i_m \le |m_{ind}[i]| \; * \; \mathtt{iX} \mapsto i_x \le |x_{ind}| \; * \; \mathtt{cond}(i_m, i_x) \Leftrightarrow b \; * \\ & \ulcorner \min(x_0, m_0) = s \urcorner \; * \\ & \mathtt{ans(1)} + i \mapsto \textstyle\sum_{j<s} \overline{m}(j)\overline{x}(j) \; * \\ & \ulcorner x_0 < m_0 \Rightarrow \forall m \in m_{ind}[i], \; m < m_0 \Rightarrow m < x_0 \urcorner \; * \\ & \ulcorner m_0 < x_0 \Rightarrow \forall x \in x_{ind}, \; x < x_0 \Rightarrow x < m_0 \urcorner \end{aligned} \tag{10}$$

Where with x_0 we denote $x_{ind}[i_x]$ if $i_x < \mathtt{length}(x_{ind})$ and $\max(x_{ind}) + 1$ otherwise. Same for m_0.

The first line of the invariant simply gives names to the values stored in the vector counters (i_m and i_x) and relates their values to the loop condition b. The second line is the most subtle: it captures the overall "progress" through the dense pair of the dense vectors being multiplied by storing the minium of $x_{ind}[ix]$ and $m_{ind}[i][im]$ in the variable s, which according to the rule premises, also counts the number of auxiliary programs "dispatched" by that time. This assertion is crucial for determine the *lower boundary* for the index j determining which components to include for alignment with the current loop iteration and what portion of the product of the two vectors has been already accumulated in the element $\mathsf{ans}(1) + i$ of the result (the last assertion in (10)).

Finally last two lines ensure that x_0 and m_0 are in some sense closest values of x_{ind} and $m_{ind}[i]$. For instance if $x_0 < m_0$, m_0 is the smallest value of $m_{ind}[i]$ which is still greater that x_0.

We assume our index set division to be

$$\{\langle 2, j \rangle \mid j \in [s, st + 1)\} \cup \{\langle 3, j \rangle \mid j \in [s, st + 1)\} = \bigcup_{j=0}^{st+1-1} \{\langle 2, j \rangle, \langle 3, j \rangle\}$$

Note that $I_{\texttt{wl}}(s, \mathrm{false}, \overline{v})$ implies that $\mathrm{cond}(i_x, i_m)$ is false, hence either $i_x \geq |x_{ind}|$ or $i_m \geq |m_{ind}[i]|$. However $I_{\texttt{wl}}$ ensures $i_x \geq |indx|$ and $i_m \leq |indm[i]|$. So in that case one of i_m and i_x equals to the size of the correspondent array, hence $s = st + 1$. Which is exactly our upper bound (M in WHILE).

Abbreviating the body of the `while`-loop as q and applying the rule WHILE gives us the following:

$$\left\{\, I_{\texttt{wl}}(s, \mathrm{true}, \overline{mx}) \right\} \left[\begin{array}{l} \quad 1 \quad : q; \texttt{while}(\texttt{cond(iM, iX)})\ \{q\} \\ \langle 2, j\rangle_{s \leq j \leq st} : \texttt{svi}(\texttt{m}_{ind}[\texttt{i}], \texttt{m}_{val}[\texttt{i}], j) \\ \langle 3, j\rangle_{s \leq j \leq st} : \texttt{svi}(\texttt{x}_{ind}, \texttt{x}_{val}, j) \end{array}\right] \left\{ \begin{array}{c} - \\ \overline{m} \\ \overline{x} \end{array} \middle| \, I_{\texttt{wl}}(\mathrm{false}, st+1, \overline{mx}) \right\} \tag{11}$$

Since the body of the loop is an `if`-statement, the rest of the proof proceeds by the case analysis.

In first case, $x_{ind}[ix] = m_{ind}[i][im]$. Let us define $s' \triangleq \min(x_{ind}[ix+1], m_{ind}[i][im+1])$, which should be a new value of the "auxiliary program counter" s after this iteration. This brings us to

$$\left\{\, I_{\texttt{wl}}(\mathrm{true}, s, \overline{mx}) \right\} \left[\begin{array}{l} \quad 1 \quad : q \\ \langle 2, j\rangle_{j \in [s, s')} : \texttt{svi}(\texttt{m}_{ind}[\texttt{i}], \texttt{m}_{val}[\texttt{i}], j) \\ \langle 3, j\rangle_{j \in [s, s')} : \texttt{svi}(\texttt{x}_{ind}, \texttt{x}_{val}, j) \end{array}\right] \left\{ \begin{array}{c} - \\ \overline{m}' \\ \overline{x}' \end{array} \middle| \, I_{\texttt{wl}}(b', s', \overline{m}'\overline{mx}'\overline{x}) \right\} \tag{12}$$

The post-condition of (12) will therefore serve as a precondition to the induction hypothesis about the next iteration of the `while`-loop from the premise of the WHILE rule, Furthermore, the statement of (12) means that the current iteration of the body of the loop, will process all elements $x[i]$ and $m[i][j]$ in a pair-wise fashion for all j such that $s \leq j < s'$. To prove this goal, let us execute q to completion, updating the values of `ans`, `iM`, and `iX`, resulting in the following goal, where y binds the accumulated sum $\sum_{j<s} \overline{m}(j)\overline{x}(j)$ and all irrelevant parts of the invariant are omitted:

$$\left\{ \begin{array}{l} \mathrm{ans}(1) + i \mapsto \\ \qquad y + x_{ind}[i_x] m_{ind}[i][i_m] \\ * \ \ldots \end{array} \right\} \left[\begin{array}{l} \quad 1 : \texttt{skip} \\ \langle 2, j\rangle_{j \in [s, s')} : \texttt{svi}(\ldots) \\ \langle 3, j\rangle_{j \in [s, s')} : \texttt{svi}(\ldots) \end{array}\right] \left\{ \begin{array}{c} - \\ \overline{m}' \\ \overline{x}' \end{array} \middle| \begin{array}{l} \mathrm{ans}(1) + i \mapsto \\ \qquad y + \sum_{j=s}^{s'-1} \overline{m}'(j)\overline{x}'(j) \\ * \ \ldots \end{array} \right\} \tag{13}$$

We prove this goal by showing that $x_{val}[i_x]$ and $m_{val}[i][i_m]$ are the only *simultaneously* non-zero values of $\overline{x}'$ and $\overline{m'}$. Recall that $s = x_{ind}[i_x] = m_{ind}[i][i][i_m]$ and $s' = \min(x_{ind}[i_x + 1], m_{ind}[i][i][i_m + 1])$, which excludes both $x_{ind}[i_x + 1]$ and $m_{ind}[i][i][i_m + 1]$ from the domains of $\overline{x}'$ and $\overline{m'}$.

This fact can be derived from the following two complementary specifications of `svi`:

$$\left\{ \mathrm{arr}(\mathrm{ind},\ ind) * \mathrm{arr}(\mathrm{val},\ val) * \ulcorner j = ind[i] \urcorner \right\} \left[\, e : \texttt{svi}(\mathrm{ind}, \mathrm{val}, j) \right] \left\{\, x \,\middle|\, \ulcorner x = val[i] \urcorner * \ \ldots \right\} \tag{14}$$

$$\left\{ \mathrm{arr}(\mathrm{ind},\ ind) * \mathrm{arr}(\mathrm{val},\ val) * \ulcorner j \notin ind \urcorner \right\} \left[\, e : \texttt{svi}(\mathrm{ind}, \mathrm{val}, j) \right] \left\{\, x \,\middle|\, \ulcorner x = 0 \urcorner * \ \ldots \right\} \tag{15}$$

The second specification can be used for any j, such that $s < j < s'$, which implies that it's neither in x_{ind} nor in $m_{ind}[i][i]$. Both (14) and (15) can be derived in unary separation logic.

Let't consider other parts of the invariant, that we have to prove:

$$\begin{array}{l} \mathtt{iM} \mapsto i_m + 1 \leq |m_{ind}[i]| \ * \ \mathtt{iX} \mapsto i_x + 1 \leq |x_{ind}| \ * \\ \ulcorner \min(x_{ind}[i_x + 1], im_{ind}[i_m + 1]) = s' \urcorner * \\ \ulcorner x_0 < m_0 \Rightarrow \forall m \in m_{ind}[i],\ m < m_0 \Rightarrow m < x_0 \urcorner * \\ \ulcorner m_0 < x_0 \Rightarrow \forall x \in x_{ind},\ x < x_0 \Rightarrow x < m_0 \urcorner \end{array}$$

Where with x_0 and m_0 we denote $x_{ind}[i_x + 1]$ and $m_{ind}[i][i_m + 1]$. First two lines are trivial. Let's consider last two lines. As they are symmetric we consider only first of them. Assume $x_{ind}[i_x + 1] < m_{ind}[i][i_m + 1]$ and $m \in m_{ind}[i]$ such that $m < m_{ind}[i][i_m + 1]$. Because of the fact that $m_{ind}[i]$ is sorted we derive that $m \leq m_{ind}[i][i_m]$. The fact that $m < x_0$ follow from

$$m \leq m_{ind}[i][im] = x_{ind}[ix] < x_0$$

In the second case for (11), $x_{ind}[i_x] < m_{ind}[i][i_m]$, which makes the goal (12) true for $s' = \min(x_{ind}[i_x + 1], m_{ind}[i][i_m])$. To prove this goal, let us execute q to completion, updating the values of `ans`, `iM`, and `iX`, resulting in the following goal, where y binds the accumulated sum $\sum_{j<s} \overline{m}(j)\overline{x}(j)$ and all irrelevant parts of the invariant are omitted:

$$\left\{\begin{array}{c} \mathsf{ans}(1) + i \mapsto \\ y \\ * \ \ldots \end{array}\right\} \left[\begin{array}{c} 1 : \mathsf{skip} \\ \langle 2, j\rangle_{j\in[s,s')} : \mathsf{svi}(\ldots) \\ \langle 3, j\rangle_{j\in[s,s')} : \mathsf{svi}(\ldots) \end{array}\right] \left\{\begin{array}{c|c} \_ & \mathsf{ans}(1) + i \mapsto \\ \overline{m}' & y + \sum_{j=s}^{s'-1} \overline{m}'(j)\overline{x}'(j) \\ \overline{x}' & * \ \ldots \end{array}\right\} \tag{16}$$

We prove this goal by showing that there no *simultaneously* non-zero values of $x_{val}[i_x]$ and $m_{val}[i][i_m]$ between s and s'. Recall that $s = x_{ind}[i_x] < m_{ind}[i][i_m]$ and $s' = \min(x_{ind}[i_x{+}1], m_{ind}[i][i_m])$, which excludes $x_{ind}[i_x + 1]$. So the only non-zero value of x_{ind} there is $x_{ind}[i_x]$. Assume that there is $m \in m_{ind}[i]$ such that $m = x_{ind}[i_x]$. In that case $m < m_{ind}[i][i_m]$. And because of our invariant it should be less then $x_{ind}[i_x]$. So there is no such value m, thus all values of $\overline{m'x'}$ are zeros.

Now we have to prove

$$\begin{array}{l} \mathtt{iM} \mapsto i_m \le |m_{ind}[i]| \ * \ \mathtt{iX} \mapsto i_x + 1 \le |x_{ind}| \ * \\ \ulcorner \min(x_{ind}[i_x + 1], im_{ind}[i_m]) = s' \urcorner * \\ \ulcorner x_0 < m_0 \Rightarrow \forall m \in m_{ind}[i],\ m < m_0 \Rightarrow m < x_0 \urcorner * \\ \ulcorner m_0 < x_0 \Rightarrow \forall x \in x_{ind},\ x < x_0 \Rightarrow x < m_0 \urcorner \end{array}$$

Where with x_0 and m_0 we denote $x_{ind}[i_x + 1]$ and $m_{ind}[i][i_m]$. First two lines are trivial. Let's consider last two lines.

Assume $x_{ind}[i_x + 1] < m_{ind}[i][i_m]$ and $m \in m_{ind}[i]$ such that $m < m_{ind}[i][i_m]$. We know that $x_{ind}[i_x] < x_{ind}[i_x + 1]$, because x_{ind} is sorted. Hence $x_{ind}[i_x] < m_{ind}[i][i_m]$. But our invariant holds for i_m and i_x, hence by its last line we conclude that $m < x_{ind}[i_x]$.

Finally consider the the case when $m_{ind}[i][i_m] < x_{ind}[i_x{+}1]$, and $x \in x_{ind}$ such that $x < x_{ind}[i_x{+}1]$. The last fact due to sortedness of x_{ind} implies that $x \le x_{ind}[i_x]$. But we are considering the case where $x_{ind}[i_x] < m_{ind}[i][i_m]$. Hence $x < m_{ind}[i][i_m]$.

The third case when $x_{ind}[i_x] > m_{ind}[i][i_m]$ is analogous to the second one.