Axiomatic Memory Model Specifications

Akshay Gopalakrishnan (self Reference material)

November 26, 2024

Preliminary Definitions

Language syntax:

$$E ::= \ r \mid v \mid X \mid E + E \mid E * E \mid E \leq E \mid ...$$

$$C ::= \ skip \mid C; C \mid r = E \mid r = X \mid X = E \mid r = RMW(X, E, E) \mid r = F$$

$$\mid br\ label \mid br\ label\ label \mid ...$$

$$P ::= \ X = v; ...; X = v; \{C|...|C\}$$

In the above syntax:

- $X \in Locs$ - Where Locs is a map of the shared memory on which programs will operate.
- $r \in Reg$ - Where Reg is the set of registers per thread.
- $v \in Val$ - Where Val is the set of literals.

Preliminary Definitions cnt'd

Given a binary relation R, here are the useful variants for our purpose:

- R^{-1} - Inverse relation $\langle a, b\rangle \in R \;\Rightarrow\; \langle b, a\rangle \in R^{-1}$.
- $R^{?}$ - Reflexive relation $\langle a, a\rangle \in R$.
- R^{+} - Transitive relation $\langle a, b\rangle \in R \;\wedge\; \langle b, c\rangle \in R \;\Rightarrow\; \langle a, c\rangle \in R$.
- R^{*} - Reflexive Transitive relation $R^{+}UR^{?}$.
- $R1; R2$ - Sequential composition $\langle a, b\rangle \in R1 \;\wedge\; \langle b, c\rangle \in R2 \;\Rightarrow\; \langle a, c\rangle \in R1; R2$
- $[E]$ - Identity relation $\langle a, b\rangle \in [E] \;\Rightarrow\; a = b$.
- $dom(R)$ - Domain of R.
- $codom(R)$ - Range of R.

Preliminary Definitions cnt'd

Event is:

- Of the form $\langle id, tid, lab \rangle$.
- *id* - Unique identifier distinguishing event in the whole program.
- *tid* - Thread identifier distinguishing event in its own thread.
- $lab = \langle op, loc, rval, wval \rangle$ - Event label specifying the operation and the memory/fence involved.
  - $op ::= \ Ld \mid St \mid U \mid F$. - Load, Store, Update or Fence.
  - *loc* - Memory location on which the operation is.
  - *rval* - Read value of the operation.
  - *wval* - Write value of the operation.

Preliminary Definitions cnt'd

Axiomatic events:

- Read event $\Re = Ld \cup U$.
- Write event $W = St \cup U$.
- Fence - Can be of various types depending on the architecture / language's memory model.

Binary relations:

- po - Program order per thread.
- rf - Between a read and the write from which its read value comes.
- co - Order between writes/updates to same location.
- mo - Order between memory writes/updates.

Sequential Consistency (SC)

Additional binary relations:

- hb - Happens before $hb = (po \cup rf)^+$
- rb- Reads before (from reads) $rb = (rf^{-1}; mo_{|loc})$

SC is more commonly known to be the following acyclicity constraint that must hold for any program execution

$$(po \cup rf \cup rb \cup co) \text{ acyclic.}$$

SC: Alternative Definition

SC can also be defined using the following irreflexivity relations that must hold for any program execution:

a) mo irreflexive and total.

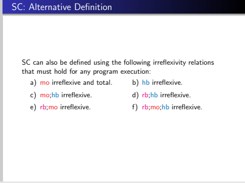
b) hb irreflexive.

c) mo;hb irreflexive.

d) rb;hb irreflexive.

e) rb;mo irreflexive.

f) rb;mo;hb irreflexive.



SC: Alternative Definition

SC can also be defined using the following irreflexivity relations that must hold for any program execution:

a) mo irreflexive and total.
b) hb irreflexive.
c) mo;hb irreflexive.
d) rb;hb irreflexive.
e) rb;mo irreflexive.
f) rb;mo;hb irreflexive.

The above model is equivalent to the acyclic model of SC, which is $(po \cup rf \cup mo \cup fr)$ acyclic. Because $(po \cup rf)$ is hb, the above definition becomes $(hb \cup mo \cup fr)$ acyclic. This acyclic condition can be divided into the above 6 irreflexivity conditions (just pick them in combination, and they must be irreflexive). A simple permutation of sets and property of acyclic binary unions. The next slide does in fact show graphs of these cases for better intuition.

Figures to explain above axioms of irreflexivity

$$\begin{tikzcd}
\mathtt{R/W/U/F} \arrow[loop above, "\mathtt{hb}"]
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathtt{W/U/F} \arrow[d, bend left, "\mathtt{hb}"] \\
\mathtt{W/U/F} \arrow[u, bend left, dotted, "\mathtt{mo}"]
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathtt{W}_x/\mathtt{U}_x \arrow[rr, dotted, "\mathtt{mo}"] \arrow[dr, dashed, "\mathtt{rf}"'] & & \mathtt{W}_x/\mathtt{U}_x \arrow[dl, "\mathtt{hb}"] \\
& \mathtt{R}_x/\mathtt{U}_x &
\end{tikzcd}$$

$$\begin{tikzcd}
\mathtt{W}_x/\mathtt{U}_x \arrow[rr, dotted, "\mathtt{mo}"] \arrow[dr, dashed, "\mathtt{rf}"'] & & \mathtt{W}_x/\mathtt{U}_x \arrow[dl, dotted, "\mathtt{mo}"] \\
& \mathtt{U}_x &
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathtt{W}_x/\mathtt{U}_x \arrow[r, dotted, "\mathtt{mo}"] \arrow[d, dashed, "\mathtt{rf}"'] & \mathtt{W}_x/\mathtt{U}_x \arrow[d, dotted, "\mathtt{mo}"] \\
\mathtt{R}_x/\mathtt{U}_x & \mathtt{W/U/F} \arrow[l, "\mathtt{hb}"']
\end{tikzcd}$$

x86 Total Store Order (TSO)

Additional binary relations:

- xhb - Happens before $xhb = (po \cup rf)^+$
- fr - Reads before (from reads) $fr = (rf^{-1}; mo_{|loc})$
- rfe - Reads from external $rf \setminus po$

x86 TSO is defined using the following irreflexivity relations that must hold for any program execution:

a) xhb irreflexive.

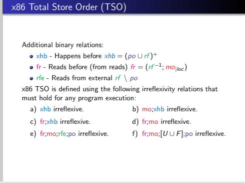
b) mo;xhb irreflexive.

c) fr;xhb irreflexive.

d) fr;mo irreflexive.

e) fr;mo;rfe;po irreflexive.

f) fr;mo;$[U \cup F]$;po irreflexive.



x86 Total Store Order (TSO)

Additional binary relations:

- xhb - Happens before $xhb = (po \cup rf)^+$
- fr - Reads before (from reads) $fr = (rf^{-1}; mo_{|loc})$
- rfe - Reads from external $rf \setminus po$

x86 TSO is defined using the following irreflexivity relations that must hold for any program execution:

a) xhb irreflexive.
b) mo;xhb irreflexive.
c) fr;xhb irreflexive.
d) fr;mo irreflexive.
e) fr;mo;rfe;po irreflexive.
f) fr;mo;$[U \cup F]$;po irreflexive.

The last two rules show the difference betweeen TSO and SC. Basically, they ONLY restrict read values from main memory to be that of the last written value. But, cases where a read takes value from its own thread's write buffer is perfectly fine.

Figures to explain the last two axioms of irreflexivity.

$$\begin{tikzcd}
a : \mathtt{W}_x/\mathtt{U}_x \arrow[rr, dotted, "\mathtt{mo}"] \arrow[dr, dashed, "\mathtt{rf}"'] & & b : \mathtt{W}_x/\mathtt{U}_x \arrow[rr, dotted, "\mathtt{mo}"] & & c : \mathtt{W}/\mathtt{U} \arrow[dl, dashed, "\mathtt{rf} \setminus \mathtt{po}"] \\
& d : \mathtt{R}_x/\mathtt{U}_x & & e : \mathtt{R}_y/\mathtt{U}_y \arrow[ll, "\mathtt{po}"] &
\end{tikzcd}$$

$$\begin{tikzcd}
a : \mathtt{W}_x/\mathtt{U}_x \arrow[r, dotted, "\mathtt{mo}"] \arrow[d, dashed, "\mathtt{rf}"'] & b : \mathtt{W}_x/\mathtt{U}_x \arrow[d, dotted, "\mathtt{mo}"] \\
d : \mathtt{R}_x/\mathtt{U}_x & c : \mathtt{U}/\mathtt{F} \arrow[l, "\mathtt{po}"]
\end{tikzcd}$$

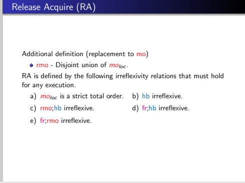
Release Acquire (RA)

Additional definition (replacement to mo)

- rmo - Disjoint union of mo_{loc}.

RA is defined by the following irreflexivity relations that must hold for any execution.

a) mo_{loc} is a strict total order.
b) hb irreflexive.
c) rmo;hb irreflexive.
d) fr;hb irreflexive.
e) fr;rmo irreflexive.



Release Acquire (RA)

Additional definition (replacement to mo)

- rmo - Disjoint union of mo_{loc}.

RA is defined by the following irreflexivity relations that must hold for any execution.

a) mo_{loc} is a strict total order.
b) hb irreflexive.
c) rmo;hb irreflexive.
d) fr;hb irreflexive.
e) fr;rmo irreflexive.

The release acquire model is mainly known for its non-multi-copy-atomic behavior. This means, threads can see updates to memory in different orders or timelines. The typical example to show this is called the IRIW litmus tests (literally means, immeidate read, immediate write). Other than this feature, release acquire also allows reordering of the forms W-R (write-read reordering). As a special transformation, it is observed that many non-multi-copy atomic behaviors can be explained in a multi-copy atomic sense using the Sequentialization optimization, which is sound under RA. Proof for this can be written as a side exercise during PhD.

Example: IRIW (non-multi-copy atomic behaviors shown by RA)

$$\begin{array}{l} a := x; \quad /\!/\, 1 \\ b := y; \quad /\!/\, 0 \end{array} \;\Bigg\|\; x := 1; \;\Bigg\|\; y := 1; \;\Bigg\|\; \begin{array}{l} c := y; \quad /\!/\, 1 \\ d := x; \quad /\!/\, 0 \end{array}$$

Note: This behavior is neither allowed under SC nor under TSO as they exhibit only multi-copy atomic behaviors.

JavaScript Memory Model

Additional type for each memory access - Unordered(uo)—Initialize($init$)—SequentiallyConsistent(sc)

- $W \ ::= \ uo|init|sc.$
- $R \ ::= \ uo|sc.$
- $RMW \ ::= \ sc$

Memory location(s) have three forms - Disjoint(dj)—Overlap(op)—Equal(eq) Wherever relevant, in a binary relation we will use the above three forms.
Additional Binary Relations:

- sw - Synchronize-with $po \ \cup \ ([W_{sc}]; rf; [R_{sc}])_{eq}$
- jhb - Happens before $sw \ \cup \ \{\langle a, b\rangle | a = W[init] \ \wedge \ b = W|R|RMW \ \wedge \ ([a]; xhb; [b])_{op}\}$
- jmo - Memory order (total order of all events in execution)

JavaScript Memory Model: Axioms

a) jhb irreflexive.

b) jhb;jmo irreflexive.

c) jhb;rf irreflexive.

d) $(fr; xhb)_{op}$ irreflexive.

1. $(([R_{sc}]; fr; [W_{sc}])_{eq}; mo; [W'_{sc}]; mo)_{eq}$ irreflexive.
2. $([W_{sc}]; jhb; [R]; fr; [W'_{sc}])_{eq}; jhb; [R]; rf^{-1}$ irreflexive.
3. $[W]; jhb; ([R_{sc}]; fr; [W'_{sc}])_{eq}; jhb; [R_{sc}]; rf^{-1}$ irreflexive.

C11/C++ Memory Consistency Model

Additional binary relations:

Mapping from SC to TSO

- SC Load — TSO Load $LD \mapsto LDR$.
- SC Store —- MFENCE;TSO Store $ST \mapsto MFENCE; STR$.

The above mapping from SC —¿ TSO is correct.

Mapping from SC to TSO : Proof Statement

Consider P_{SC} and P_{TSO} to be programs corresponding to each memory model such that the above mapping is applied $P_{SC} \mapsto P_{TSO}$. Let us consider X_{SC} and X_{TSO} to be the respective consistent executions under the memory models. Then, the following holds true

$$P_{SC} \mapsto P_{TSO} \implies \forall X_{TSO} \, \exists X_{SC} \; s.t. \; B(X_{SC}) = B(X_{TSO}).$$

where B is a function that maps an execution to its corresponding observable behavior (reads-from relations).

Proof idea

We do it using proof by contradiction.
We take one execution X_{TSO} which is consistent. If the above proof statement is to be false, then the same execution must have been illegal under SC, X_{SC} is not consistent. Inconsistent execution would mean that one of the irreflexivity axioms of SC fails to hold, while all of those of TSO holds.
We prove that this is not the case, by showing that if it is inconsistent in SC, it is inconsistent in TSO as per the mapping.

Proof cnt'd

Firstly, note that the following relations of *SC* are contained in *TSO*

a) po
b) rf
c) mo
d) hb
e) fr

From the above, and by the property that if a sequential composition is irreflexive and contains another composition, then the latter also is irreflexive.

$$R1 \subseteq R2 \ \wedge \ R2 \text{ irreflexive} \implies R1 \text{ irreflexive.}$$

Thus the following axioms hold for *SC* too:

a) hb irreflexive.
b) mo;hb irreflexive.
c) fr;hb irreflexive.
d) fr;mo irreflexive.

Note: mo remains a strict total order the same way.

Proof cnt'd

Now what remains is the last axiom of SC - $fr; mo; hb$ Assume that X_{SC} does fail due to this axiom. Thus, we have $fr; mo; hb$ to be reflexive. Then, the resulting composition due ot mapping to TSO, must also be reflexive

$$fr; mo; hb$$
$$fr; mo; [ST]; hb$$
$$fr; mo; [F; STR]; xhb$$

$fr; mo; [F; STR]; xhb$ has following 3 variants due to xhb.

a) fr;mo;[F];po.
b) fr;mo;rfe;po.
c) fr;mo;rfi;po.
d) beee

Proof cnt'd

1. fr;mo;[F];po is irreflexive as X_{TSO} is consistent.
2. fr;mo;rfe;po is irreflexive as X_{TSO} is consistent.
3. fr;mo;rfi;po is irreflexive as xhb is irreflexive.

Both possibilities $fr; mo; rfe; po$ and $fr; mo; [F]; po$ are axioms of TSO which are irreflexive. The third case is a subset of the first. Thus, by contradiction, we can infer $fr; mo; hb$ is irreflexive. Hence, proved.

Fence elimination in x86-TSO

Categorize fences as eliminable and non-eliminable. Often it is easier to specify fences which are non-eliminable. Doing so will also give us the condition (direct negation) on which Fence Optimization can be done.

An MFENCE in an x86 program is non-eliminable if it is the only fence in the program path from a store to a load in the same thread.

Categorize fences as eliminable and non-eliminable. Often it is easier to specify fences which are non-eliminable. Doing so will also give us the condition (direct negation) on which Fence Optimization can be done.

An MFENCE in an x86 program is non-eliminable if it is the only fence in the program path from a store to a load in the same thread.

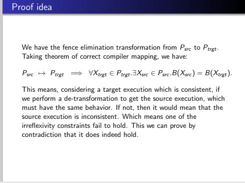
1. Store-Store need to be ordered per thread under TSO (as the write buffer is also ordered)
2. Load-Load need to be ordered as the first Load would influence the second load value (axiomatically one can think of this as happens-before).
3. Load-Store also needs to be ordered as the load would imply an ordering between the subsequent store and stores from other threads (if load is from Main Memory, this would make visible the current write value which is going to be overwritten by the subsequent write).
4. Store-Load however, does not have any reason to be ordered. A load from main memory would imply that the previous store is not relevant and could be in write buffer or could be updated in main memory. This choice is independant of when the following load is done.

Proof idea

We have the fence elimination transformation from P_{src} to P_{trgt}. Taking theorem of correct compiler mapping, we have:

$$P_{src} \mapsto P_{trgt} \implies \forall X_{trgt} \in P_{trgt}.\exists X_{src} \in P_{src}.B(X_{src}) = B(X_{trgt}).$$

This means, considering a target execution which is consistent, if we perform a de-transformation to get the source execution, which must have the same behavior. If not, then it would mean that the source execution is inconsistent. Which means one of the irreflexivity constraints fail to hold. This we can prove by contradiction that it does indeed hold.



Proof idea

We have the fence elimination transformation from P_{src} to P_{trgt}. Taking theorem of correct compiler mapping, we have:

$$P_{src} \mapsto P_{trgt} \implies \forall X_{trgt} \in P_{trgt}.\exists X_{src} \in P_{src}.B(X_{src}) = B(X_{trgt}).$$

This means, considering a target execution which is consistent, if we perform a de-transformation to get the source execution, which must have the same behavior. If not, then it would mean that the source execution is inconsistent. Which means one of the irreflexivity constraints fail to hold. This we can prove by contradiction that it does indeed hold.

To prove that the transformation is safe, note that the transformed program's behaviors should at least be contained (if not equal) in that of the original ($P_{trgt} \subseteq P_{src}$). If we consider the source code and perform a transformation $P_{src} \mapsto P_{trgt}$, then we have a similar property as that of compiler mapping $\forall X_{trgt}.\exists X_{src}.O(X_{trgt}) = O(X_{src})$. We can prove this straightforward by contradiction, as if there does not exist a source execution, it would only mean that it is illegal, as we know behaviors must be contained. Which means one of the irreflexivity conditions fail to hold, which we can verify similar to the compiler mapping proof. This is a very intelligent way to proving transformations, one that I did not know back then. This also solves my doubt of why a de-transformation is considered while proving program transformation. Also, this method seems to be independent of the transformation done, which greatly helps in our case.

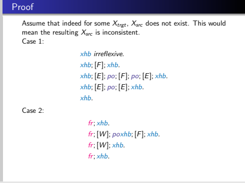
Proof

Assume that indeed for some X_{trgt}, X_{src} does not exist. This would mean the resulting X_{src} is inconsistent.
Case 1:

$$\begin{aligned}&xhb \textit{ irreflexive}.\\ &xhb;[F];xhb.\\ &xhb;[E];po;[F];po;[E];xhb.\\ &xhb;[E];po;[E];xhb.\\ &xhb.\end{aligned}$$

Case 2:

$$\begin{aligned}&fr;xhb.\\ &fr;[W];poxhb;[F];xhb.\\ &fr;[W];xhb.\\ &fr;xhb.\end{aligned}$$



Proof

Assume that indeed for some X_{trgt}, X_{src} does not exist. This would mean the resulting X_{src} is inconsistent.
Case 1:

$$xhb \textit{ irreflexive}.$$
$$xhb; [F]; xhb.$$
$$xhb; [E]; po; [F]; po; [E]; xhb.$$
$$xhb; [E]; po; [E]; xhb.$$
$$xhb.$$

Case 2:

$$fr; xhb.$$
$$fr; [W]; poxhb; [F]; xhb.$$
$$fr; [W]; xhb.$$
$$fr; xhb.$$

For Case 1, note that if a cycle exists, $[F]$ must be a part of it. Also note that, this $[F]$ and xhb relations with it are formed through direct program order relations, which must, by transitive property also be part of the cycle. However, this also means, the set of events without $[F]$ also should form a cycle, which is a contradiction as X_{trgt} is consistent. For Case 2, note that again, it implies a cycle must exist without $[F]$ being involved. Thus by contradiction Case 1 and Case 2 are eliminated.

Proof cnt'd

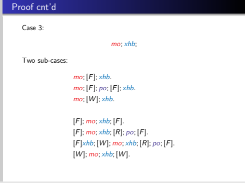
Case 3:

$$mo; xhb;$$

Two sub-cases:

$$mo; [F]; xhb.$$
$$mo; [F]; po; [E]; xhb.$$
$$mo; [W]; xhb.$$

$$[F]; mo; xhb; [F].$$
$$[F]; mo; xhb; [R]; po; [F].$$
$$[F]xhb; [W]; mo; xhb; [R]; po; [F].$$
$$[W]; mo; xhb; [W].$$



Proof cnt'd

Case 3:

$$mo; xhb;$$

Two sub-cases:

$mo; [F]; xhb.$
$mo; [F]; po; [E]; xhb.$
$mo; [W]; xhb.$

$[F]; mo; xhb; [F].$
$[F]; mo; xhb; [R]; po; [F].$
$[F] xhb; [W]; mo; xhb; [R]; po; [F].$
$[W]; mo; xhb; [W].$

For Case 3, note that apart from the naive transitive relations that imply to X_{trgt} to have cycles, we can also have $[F]$ such that it links mo and xhb, thus introducing a cycle. In the first case, $[F]$ is the codomain of mo and domain of xhb. However, note that xhb relations with $[F]$ cannot be direct, and must come from some event po ordered after it. Thus, by transitivity, requiring that there also exists a cycle without the fence. The same reasoning is used for the second case, with the added observation that mo edges cannot exist with $[F]$ unless there also exists xhb relation. This might sound a bit difficult to grasp, but on drawing example graphs, the intuition will be more clear. The crux is that there is no *direct* edge of mo or xhb with $[F]$ unless its derived from po.

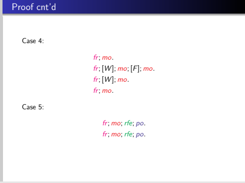
Proof cnt'd

Case 4:

$$fr; mo.$$
$$fr; [W]; mo; [F]; mo.$$
$$fr; [W]; mo.$$
$$fr; mo.$$

Case 5:

$$fr; mo; rfe; po.$$
$$fr; mo; rfe; po.$$



Proof cnt'd

Case 4:

$$fr; mo.$$
$$fr; [W]; mo; [F]; mo.$$
$$fr; [W]; mo.$$
$$fr; mo.$$

Case 5:

$$fr; mo; rfe; po.$$
$$fr; mo; rfe; po.$$

Case 4 and 5 have no case (apart from transitive relations) for having cycles with $[F]$. Thus, both cases are eliminated.

Proof cnt'd

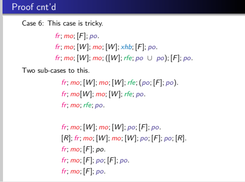
Case 6: This case is tricky.

$$fr; mo; [F]; po.$$
$$fr; mo; [W]; mo; [W]; xhb; [F]; po.$$
$$fr; mo; [W]; mo; ([W]; rfe; po \;\cup\; po); [F]; po.$$

Two sub-cases to this.

$$fr; mo; [W]; mo; [W]; rfe; (po; [F]; po).$$
$$fr; mo[W]; mo; [W]; rfe; po.$$
$$fr; mo; rfe; po.$$

$$fr; mo; [W]; mo; [W]; po; [F]; po.$$
$$[R]; fr; mo; [W]; mo; [W]; po; [F]; po; [R].$$
$$fr; mo; [F]; po.$$
$$fr; mo; [F]; po; [F]; po.$$
$$fr; mo; [F]; po.$$



Proof cnt'd

Case 6: This case is tricky.

$$fr; mo; [F]; po.$$
$$fr; mo; [W]; mo; [W]; xhb; [F]; po.$$
$$fr; mo; [W]; mo; ([W]; rfe; po \cup po); [F]; po.$$

Two sub-cases to this.

$$fr; mo; [W]; mo; [W]; rfe; (po; [F]; po).$$
$$fr; mo[W]; mo; [W]; rfe; po.$$
$$fr; mo; rfe; po.$$

$$fr; mo; [W]; mo; [W]; po; [F]; po.$$
$$[R]; fr; mo; [W]; mo; [W]; po; [F]; po; [R].$$
$$fr; mo; [F]; po.$$
$$fr; mo; [F]; po; [F]; po.$$
$$fr; mo; [F]; po.$$

Firstly, note that there cannot be any direct relation W-mo-F unless W-po-F. The reason is because of the definition of happens before between two threads, which only is formed between reads and writes. And because of the reads-from-external, a memory order is implied between writes, but not fences. So a case like $w - f - r||w'$, where $w' - rf - r$, it implies $w - w'$, but nothing with the fence. If, however, we have like w-mo-w'-xhb-f, then we can infer that w-mo-f. The two sub-cases are a result of the above observation. The first requires that X_{trgt} is also reflexive, thus eliminating this case by contradiction. The latter, is the tricky case. Firstly, if $[F]$ is the only fence, it counts as non-eliminable. If it isn't then, by transitivity, we again have the same composition(final expression) to also be a cycle in X_{trgt}. Thus by contradiction, this case also is eliminated.