Transformation from the original rule list to the final list of scope-based rules

1. Extend each deny rule to all elements of the hierarchies and then explicitly exclude those elements that must not be affected by this artificially enlarged scope
2. Float down obligation rules (see later how)
3. Float up allow rules until a rule is reached whose scope comprises the scope of the allow rule
Original rule list & hierarchies

1. $((u_1, d_0, p_2, a_2), (\circ, c_1, \bar{c}_1))$
2. $((u_3, d_1, p_4, a_2), (\neg, c_2, \bar{c}_2))$
3. $((u_2, d_2, p_2, a_0), (+, c_3, \bar{c}_3))$
1) Deny rules

• extend each deny rule to all elements of the hierarchies
• explicitly exclude those elements that must not be affected by this artificially enlarged scope.
2) Obligation rules

We have to distinguish among four cases:

1. If there is no overlap with the next lower rule, we swap both rules.

2. If the scope of the floating rule is contained in the scope of the next rule, the qualifier of that rule is appended to the floating rule’s qualifier and the obligation rule has reached its final position.

3. If the scope of the next rule is contained in the scope of the floating rule, we swap both rules but additionally append the qualifier of the floating rule to the qualifier sequence of the current rule.

4. If both rules overlap only partially, we swap the rules and additionally insert a new rule that deals with the overlap as follows:

\[
\text{overlap}((\langle u, d, p, a \rangle \text{ seq}_1), (\langle u', d', p', a' \rangle \text{ seq}_2)) =: (\langle u^*, d^*, p^*, a^* \rangle \text{ seq}_1)
\]

where

\[
u^* = \begin{cases} u & \text{if } u \leq u' \\ u' & \text{otherwise} \end{cases}
\]

and similarly for the other dimensions.
2) Obligation rules

We have to distinguish among four cases:

1. If there is no overlap with the next lower rule, we swap both rules.

2. If the scope of the floating rule is contained in the scope of the next rule, the qualifier of that rule is appended to the floating rule’s qualifier and the obligation rule has reached its final position.

3. If the scope of the next rule is contained in the scope of the floating rule, we swap both rules but additionally append the qualifier of the floating rule to the qualifier sequence of the current rule.

4. If both rules overlap only partially, we swap the rules and additionally insert a new rule that deals with the overlap as follows:

\[
\text{overlap}(\langle (u, d, p, a) \rangle, \langle (u', d', p', a') \rangle) = : \langle (u^*, d^*, p^*, a^*) \rangle
\]

where

\[
u^* = \begin{cases} 
u & \text{if } u \leq_Y u' \\ u' & \text{otherwise} \end{cases}
\]

and similarly for the other dimensions.
2) Obligation rules

We have to distinguish among four cases:

1. If there is no overlap with the next lower rule, we swap both rules.
2. If the scope of the floating rule is contained in the scope of the next rule, the qualifier of that rule is appended to the floating rule’s qualifier and the obligation rule has reached its final position.
3. If the scope of the next rule is contained in the scope of the floating rule, we swap both rules but additionally append the qualifier of the floating rule to the qualifier sequence of the current rule.
4. If both rules overlap only partially, we swap the rules and additionally insert a new rule that deals with the overlap as follows:

\[
\text{overlap}((\langle u, d, p, a \rangle \text{ seq}_1), (\langle u', d', p', a' \rangle \text{ seq}_2)) =: (\langle u^*, d^*, p^*, a^* \rangle \text{ seq}_1)
\]

where

\[
u^* = \begin{cases} u & \text{if } u \leq_{\mathcal{V}} u' \\ u' & \text{otherwise} \end{cases}
\]

and similarly for the other dimensions.
2) Obligation rules

We have to distinguish among four cases:

1. If there is no overlap with the next lower rule, we swap both rules.

2. If the scope of the floating rule is contained in the scope of the next rule, the qualifier of that rule is appended to the floating rule’s qualifier and the obligation rule has reached its final position.

3. If the scope of the next rule is contained in the scope of the floating rule, we swap both rules but additionally append the qualifier of the floating rule to the qualifier sequence of the current rule.

4. If both rules overlap only partially, we swap the rules and additionally insert a new rule that deals with the overlap as follows:

\[
\text{overlap}(((u, d, p, a) \text{ seq}_1), ((u', d', p', a') \text{ seq}_2)) =: ((u^*, d^*, p^*, a^*) \text{ seq}_1)
\]

where

\[
u^* = \begin{cases} u & \text{if } u \leq_Y u' \\ u' & \text{otherwise} \end{cases}
\]

and similarly for the other dimensions.
2) Obligation rules

We have to distinguish among four cases:

1. If there is no overlap with the next lower rule, we swap both rules.

2. If the scope of the floating rule is contained in the scope of the next rule, the qualifier of that rule is appended to the floating rule’s qualifier and the obligation rule has reached its final position.

3. If the scope of the next rule is contained in the scope of the floating rule, we swap both rules but additionally append the qualifier of the floating rule to the qualifier sequence of the current rule.

4. If both rules overlap only partially, we swap the rules and additionally insert a new rule that deals with the overlap as follows:

   \[
   \text{overlap}((u, d, p, a) \ \text{seq}_1, (u', d', p', a') \ \text{seq}_2) = ((u^*, d^*, p^*, a^*) \ \text{seq}_1)
   \]

   where 
   \[
   u^* = \begin{cases} 
   u & \text{if } u \leq u' \\
   u' & \text{otherwise}
   \end{cases}
   \]

   and similarly for the other dimensions.
2) Obligation rules

We have to distinguish among four cases:

1. If there is no overlap with the next lower rule, we swap both rules.

2. If the scope of the floating rule is contained in the scope of the next rule, the qualifier of that rule is appended to the floating rule’s qualifier and the obligation rule has reached its final position.

3. If the scope of the next rule is contained in the scope of the floating rule, we swap both rules but additionally append the qualifier of the floating rule to the qualifier sequence of the current rule.

4. If both rules overlap only partially, we swap the rules and additionally insert a new rule that deals with the overlap as follows:

   \[
   \text{overlap}((u, d, p, a) \ seq_1), ((u', d', p', a') \ seq_2)) =: ((u^*, d^*, p^*, a^*) \ seq_1)
   \]

   where

   \[
   u^* = \begin{cases} 
   u & \text{if } u \leq u' \\
   u' & \text{otherwise}
   \end{cases}
   \]

   and similarly for the other dimensions.
2) Obligation rules

We have to distinguish among four cases:

1. If there is no overlap with the next lower rule, we swap both rules.

2. If the scope of the floating rule is contained in the scope of the next rule, the qualifier of that rule is appended to the floating rule’s qualifier and the obligation rule has reached its final position.

3. If the scope of the next rule is contained in the scope of the floating rule, we swap both rules but additionally append the qualifier of the floating rule to the qualifier sequence of the current rule.

4. If both rules overlap only partially, we swap the rules and additionally insert a new rule that deals with the overlap as follows:

\[
\text{overlap}((u, d, p, a) \text{ seq}_1), (u', d', p', a') \text{ seq}_2) =: ((u^*, d^*, p^*, a^*) \text{ seq}_1)
\]

where

\[
u^* = \begin{cases} 
  u & \text{if } u \leq u' \\
  u' & \text{otherwise}
\end{cases}
\]

and similarly for the other dimensions.
2) Obligation rules

We have to distinguish among four cases:

1. If there is no overlap with the next lower rule, we swap both rules.

2. If the scope of the floating rule is contained in the scope of the next rule, the qualifier of that rule is appended to the floating rule’s qualifier and the obligation rule has reached its final position.

3. If the scope of the next rule is contained in the scope of the floating rule, we swap both rules but additionally append the qualifier of the floating rule to the qualifier sequence of the current rule.

4. If both rules overlap only partially, we swap the rules and additionally insert a new rule that deals with the overlap as follows:

   \[
   \text{overlap}((u, d, p, a) \text{ seq}_1), ((u', d', p', a') \text{ seq}_2) =: ((u^*, d^*, p^*, a^*) \text{ seq}_1)
   \]

   where

   \[
   u^* = \begin{cases} 
   u & \text{if } u \leq \forall u' \\
   u' & \text{otherwise} 
   \end{cases}
   \]

   and similarly for the other dimensions.
3) Allow rules

1. Float up allow rules until a rule is reached whose scope comprises the scope of the allow rule.

Note: If the step is applied to the letter, then rule 3 should be below rule 2c. But... (see next slide)
But...

- The obtained rule set should be equivalent to the original one
- Consider request \(<u_2,d_2,p_2,a_0>\)
3) Allow rules

1. Float up allow rules until a (obligation) rule is reached whose scope comprises the scope of the allow rule.
Comparison of Qualifier Sequences

**Goal:** given two sequences of qualifiers, check whether there is a refinement

For each qualifier in one sequence and in descending order,
- check the qualifiers in the second sequence for refinement
  - The conditions of the two qualifiers should be concurrently satisfied
- until a qualifier whose condition implies the qualifier’s condition of the first sequence is reached
Comparison in action

Consider the following sequences of qualifiers.

$(r_1, c_1 \land c_2 \land c_3, \bar{o}_1)$; $(r_2, c_1 \land c_2, \bar{o}_2)$; $(r_3, c_2, \bar{o}_3)$; $(r_4, c_1, \bar{o}_4)$; $(dr_1, \text{true, } \bar{d}o_1)$

$(r_1', c_1 \land c_3, \bar{o}'_1)$; $(r_2', c_3, \bar{o}'_2)$; $(r_3', c_1, \bar{o}'_3)$; $(dr_2, \text{true, } \bar{d}o_2)$

Does sequence (2) refine sequence (1)?
Comparison (1)

• Take the first qualifier of sequence (2) and assume that condition $c_1 \land c_3$ is true.
• Condition $c_1 \land c_2 \land c_3$ of the first qualifier $(r_1, c_1 \land c_2 \land c_3, \bar{c}_1)$ may be true concurrently.
• Compare $(r_1, o_1)$ and $(r'_1, o'_1)$:
  • if $r_1 \neq o$ (don’t care) then $r'_1 = r_1$?
  • $o'_1$ refines $o_1$?
• If this holds, continue the comparison with the next qualifier in sequence (1).
Comparison (2)

\[(r_1, c_1 \land c_2 \land c_3, \bar{o}_1); (r_2, c_1 \land c_2, \bar{o}_2); (r_3, c_2, \bar{o}_3); (r_4, c_1, \bar{o}_4); (dr_1, \text{true}, \bar{d}o_1) \]
\[(r_1', c_1 \land c_3, \bar{o}_1'); (r_2', c_3, \bar{o}_2'); (r_3', c_1, \bar{o}_3'); (dr_2, \text{true}, \bar{d}o_2) \]

\[(r_2, c_1 \land c_2, \bar{o}_2) \text{ (and condition } c_1 \land c_3 \text{ holds).} \]
\[(r_2, c_1 \land c_2, \bar{o}_2) \text{ (and condition } (c_1 \land c_3) \land \neg (c_1 \land c_2 \land c_3) \text{ holds.} \]
\[(r_2, c_1 \land c_2, \bar{o}_2) \text{ is applicable if } c_1 \land c_2 \text{ is true.} \]
\[((c_1 \land c_3) \land \neg (c_1 \land c_2 \land c_3)) \Rightarrow \neg (c_1 \land c_2) \text{ then } c_1 \land c_2 \text{ cannot be true.} \]
\[(r_2, c_1 \land c_2, \bar{o}_2) \text{ is not applicable.} \]
\[(r_3, c_2, \bar{o}_3) \text{ is not applicable.} \]
\[(r_2, c_1 \land c_2, \bar{o}_2) \text{ is not applicable.} \]
\[(r_3, c_2, \bar{o}_3) \text{ is not applicable.} \]
Comparison (3)

• Take qualifier \((r_4, c_1, \bar{o}_4)\) (and condition \(c_1 \land c_3\) holds).

• Since \(c_1 \land c_3 \Rightarrow c_1\), qualifier \((r_4, c_1, \bar{o}_4)\) is applicable

• Compare \((r_4, o_4)\) and \((r_1', o_1')\):
  - if \(r_4 =/= o\) (don’t care) then \(r_1' = r_4\)?
  - \(o_1'\) refines \(o_4\)?

• Because \(c_1 \land c_3\) implies \(c_1\), no remaining elements in sequence (1) must be checked
Comparison (4)

• Take the second qualifier of sequence (2)
• At this point, \( c_3 \) and \( \neg (c_1 \land c_3) \) hold
• Since \( c_1 \land c_3 \) does not hold, qualifier \( (r_1, c_1 \land c_2 \land c_3, \bar{o}_1) \) is not applicable
• Since \( \text{NOT}(c_1 \land c_3) \land c_3 \text{ holds} \), also qualifier \( (r_2, c_1 \land c_2, \bar{o}_2) \) is not applicable
• Since \( c_2 \text{ can be true} \), qualifier \( (r_3, c_2, \bar{o}_3) \) is applicable
• Compare \((r_3, o_3)\) and \((r'2, o'2)\):
  • if \( r'2 =/= o \) (don’t care) then \( r'2 = r3 \)?
  • \( o'2 \) refines \( o3 \)?
• If this holds, continue the comparison with the next qualifier in sequence (1).
Comparison (5)

- Take qualifier \((r_4, c_1, \bar{o}_4)\)
- At this point, \(c_3 \land \neg (c_1 \land c_3) \land \neg (c_2)\) holds
- Because of \(c_1\) cannot be true, qualifier \((r_4, c_1, \bar{o}_4)\) is not applicable
- Qualifier \((dr_1, true, do_1)\) is applicable
- Compare \((dr_1, do_1)\) and \((r'_2, o'_2)\):
  - if \(r'_2 =/= o\) (don’t care) then \(r'_2 = dr_1\)?
  - \(o'_2\) refines \(do_1\)?
- If this holds, continue the comparison with the next qualifier in sequence (2).
Comparison (6)

- Check the remaining elements in sequence (2)

<table>
<thead>
<tr>
<th>Satisfied Condition</th>
<th>Result given by seq (1)</th>
<th>Result given by seq (2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_1 \land c_2 \land c_3$</td>
<td>$(r_1, \bar{o}_1)$</td>
<td>$(r'_1, \bar{o}'_1)$</td>
</tr>
<tr>
<td>$c_1 \land c_3$</td>
<td>$(r_4, \bar{o}_4)$</td>
<td>$(r'_1, \bar{o}'_1)$</td>
</tr>
<tr>
<td>$c_2 \land c_3$</td>
<td>$(r_9, \bar{o}_9)$</td>
<td>$(r'_2, \bar{o}'_2)$</td>
</tr>
<tr>
<td>$c_3$</td>
<td>$(dr_1, \bar{d}o_1)$</td>
<td></td>
</tr>
<tr>
<td>$c_1 \land c_2$</td>
<td>$(r_2, \bar{o}_2)$</td>
<td>$(r'_3, \bar{o}'_3)$</td>
</tr>
<tr>
<td>$c_1$</td>
<td>$(r_4, \bar{o}_4)$</td>
<td>$(r'_3, \bar{o}'_3)$</td>
</tr>
<tr>
<td>$c_2$</td>
<td>$(r_3, \bar{o}_3)$</td>
<td>$(dr_2, \bar{d}o_2)$</td>
</tr>
<tr>
<td>-</td>
<td>$(dr_1, \bar{d}o_1)$</td>
<td></td>
</tr>
</tbody>
</table>

(1) $\quad (r_1, c_1 \land c_2 \land c_3, \bar{o}_1); (r_2, c_1 \land c_2, \bar{o}_2); (r_3, c_2, \bar{o}_3); (r_4, c_1, \bar{o}_4); (dr_1, \text{true}, \bar{d}o_1)$  
(2) $\quad (r'_1, c_1 \land c_3, \bar{o}'_1); (r'_2, c_3, \bar{o}'_2); (r'_3, c_1, \bar{o}'_3); (dr_2, \text{true}, \bar{d}o_2)$
Comparison of extended rule lists

**Goal:** check for refinement of two privacy policies by comparing their normalized, scope-based rule lists.

- If there is refinement for the qualifier sequences of all “matching” rules, then there is policy refinement.
Comparison of extended rule lists

Let SR1 and SR2 denote two scope-based rule lists.

• For each rule $\sigma_2 = \langle (u_2, d_2, p_2, a_2), \text{seq}_2 \rangle$ in SR2:
  • Process SR1 in descending order:
    • For each overlapping rule $\sigma_1 = \langle (u_1, d_1, p_1, a_1), \text{seq}_1 \rangle$ in SR1:
      • Check whether the qualifier sequences seq2 and seq1 constitute a refinement
      • If there is no refinement, the algorithm stops and returns false.
      • If $\text{scope}(\sigma_2) \subseteq \text{scope}(\sigma_1)$, the processing of rule $\sigma_2$ terminates.

This processing always terminates because every SR ends with rule(s) covering the entire hierarchies (by construction)
Comparison of extended rule lists

List $SR_1$

List $SR_2$

1. $\langle (u_2, d_2, p_2, a_0), seq_1 \rangle$
2. $\langle (u_4, d_0, p_2, a_2), seq_2 \rangle$
3. $\langle (u_4, d_0, p_0, a_0), seq_3 \rangle$
4. $\langle (u_1, d_0, p_2, a_2), seq_4 \rangle$
5. $\langle (u_0, d_0, p_0, a_0), seq_5 \rangle$

1'. $\langle (u_4, d_0, p_0, a_0), seq'_1 \rangle$
2'. $\langle (u_0, d_0, p_0, a_0), seq'_2 \rangle$
Comparison of extended rule lists

- Take rule 1’
- The first overlap is with rule 2 \((\text{scope}(u_4,d_0,p_2,a_2) \subseteq \text{scope}(u_4,d_0,p_0,a_0))\)
- IF the qualifier sequences seq’1 and seq 2 are a refinement
  - THEN continue
  - ELSE stop (SR2 is not a refinement of SR1)
- The next overlap is with rule 3.
- After successful comparison we do not have to check the remaining rules in SR1 because the scope of rule 3 completely covers the scope of rule 1’.
- Continue with the second rule in SR2 and check overlap with the rules in SR1 in descending order.
- Because rule 2’ has scope \((u_0,d_0,p_0,a_0)\), the qualifier sequence of every rule in SR1 must be checked.