We take apart a single change scenario for detailed analysis. We extend the sample scenario by applying the *add condition* change operation in combination with the *versioning* change strategy. Concretely, the condition of 99% of all meters has been added while still having to satisfy all existing constraints. This change is visualized in Fig. 1. The color of the nodes identifies the associated ISC: white being the common path for shared attributes (i.e, same context, connection and condition). Orange nodes and edges represent the old ISC, whereas the blue nodes and edges represent the new ISC. As can be seen the impact reaches all levels of the rete structures: alpha as well as beta networks.

Static and Dynamic Impacts on the Alpha Network.

One static impact on the alpha network is due to the router creating the alpha nodes A4, representing old process instances whose instance start time is $< t_C$, and correspondingly A6, representing new process instances started after t_C. The alpha node filter for catching all instance start times stays white due to it being shared by both A4 and A6. On the dynamic level, we trigger a reindexing process where facts are matched for the newly created alpha nodes A4 and A6. These are fed through the alpha network to determine which ISC instances fall under the different ISC versions. We assume that there is an event with

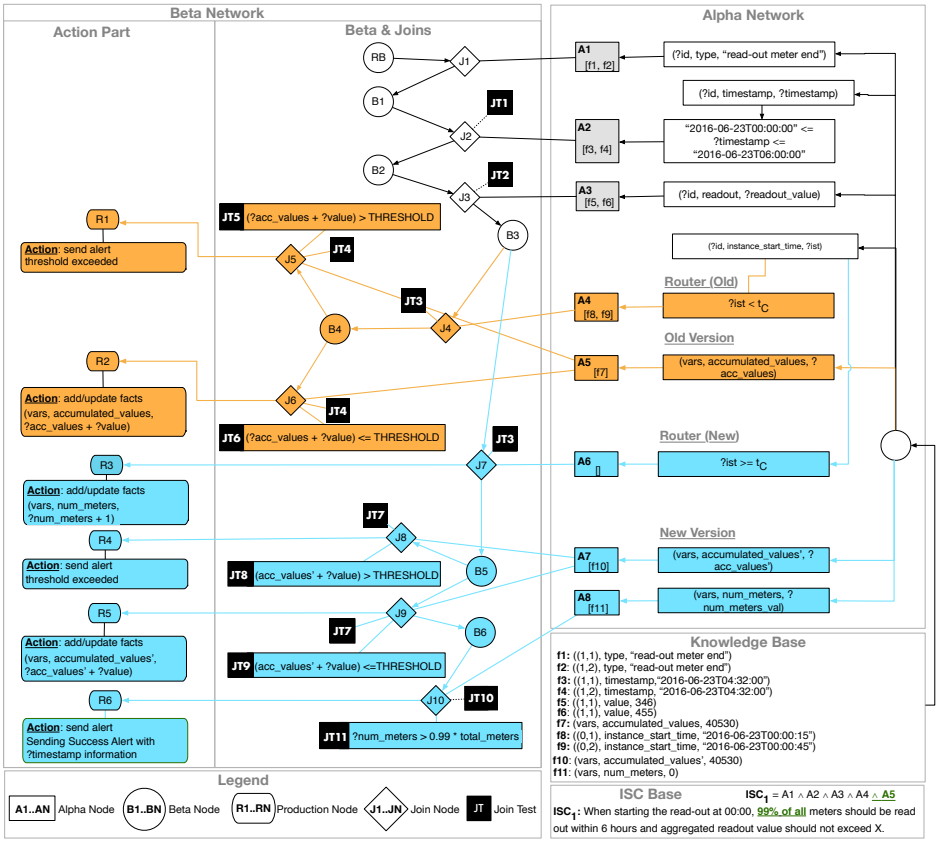

Figure 1: Change Impact Evaluation: Adding Condition + Versioning

event$_{id} = 0$ that registers the process instance's start time, which in Fig. 1 are transformed to facts $f8$ and $f9$ representing the two process instances goverened under ISC_old. There are no facts yet matching A6 for ISC_new. The second static impact on the alpha network is due to *namespacing* of the *shared variables* in the old ISC. Noticeable here is that A5 is reused from the old ISC, responsible for maintaining the *shared variable* of accumulated readout values. New alpha nodes A7 and A8 are created to maintain the state for the new ISC. On the one hand A7 is the result of *namespacing* the shared variable *accumulated_values* by *copying* its value to a new shared variable *accumulated_values*$'$ and on the other a new shared variable is introduced (A8) to maintain the actual number of meters being read out. On the corresponding dynamic impact for this part of the alpha network, the facts $f10$ and $f11$ are initialized and matched to A7 and A8 respectively. Whereas $f10$ is a simple copy of an existing shared variable (*accumulated_values*), $f11$ is initialized as the counter 0. A *copy* implies the reuse of already computed state for ISC_new, and thus considers all of the evaluated events up to the point at t_C. From this point on the two *shared variables accumulated_values* and *accumulated_values*$'$ diverge in processing of subsequent facts and represent the ISC instances of ISC_old and ISC_new respectively. Old process instances under the effect of ISC_old would only update *accumulated_values*. Similarly, only new process instances started after t_C update the state *accumulated_values*$'$, effectively isolating the two different ISC versions.

Static and Dynamic Impacts on the Beta Network Structurally, the same beta and join nodes are reused for the old ISC (marked in orange): J4, B4, J5, J6, R1 and R2. One static impact is the linking of A4 (for activating the path for instance_start_time with $< t_C$) to the old ISC's top most condition. For the new ISC, a similar static construct covering the migrated shared variables, conceptually handling the same logic as the old ISC can be identified with J7, B5, J8, J9, R4 and R5. J7 serves as the top-most join node responsible for activating the sub graph for the new ISC. This node is linked from the alpha node A6, responsible for filtering new process instances started after t_C. A new production node R3 is created that increments the number of meters that performed a read out value event (for ISC_new). This production node will always be fired when J7 is activated. Another new subgraph are the nodes B6, J10 and R6 which is responsible for verifying that the newly added 99% of all meter condition is satisfied. The key link is the alpha node A8's connection to J10, to perform the critical join test for checking the added condition. Compensating actions might need to be defined for revoking possibly already executed alerts.

Activation Paths: Old ISC vs New ISC An activation path following *ISC_new* would be taken from A6 $\rightarrow$ J7, directly firing production node R3 for updating the number of meters. Subsequently, B5 is activated to store tokens, then J8 and J9 are tested in parallel to compute whether the threshold is exceeded. Depending on the result, either R4 or R5 is fired. In case J9 succeeds (the accumulated readout value is not yet exceeded) then tokens are stored in B6, after which J10 is tested to test whether the 99% requirement has been achieved or not. If so, R6 is fired for sending a success alert, otherwise no

alert is sent. If the old rule is checked, by finding an already running instance (instance_start_time $< t_C$), then A4 is activated followed by the J4 which only tests for equal ?id part. On success, tokens are stored in B4. Joins J5 and J6 are tested in parallel. If the threshold is exceeded then R1 is fired for sending the appropriate alert, otherwise R2 is fired for updating the accumulated readout value shared variable. While this change scenario highlights ISC versioning, it incorporates the core state migration step for shared variables which is also a common part of the other change strategies: migration and clean state. The clean state strategy would simply reset the state: accumulated_values to 0, and add a router component solely handling process instances started after t_C. Since the added condition requires a new shared variable: num_meters, it is not necessary to reason about it at this point. Although further changes to the new rule would require both shared variables to be reasoned about for state migration: accumulated_values and num_meters. Change operations of type Delete would be handled similarly: maintain diverging ISC with shared attributes (versioning and clean state) and independent state, keep the old ISC until all associated process instances are completed, migrate shared variables and define compensating actions.