Forschungsprojekt: Model Order Cycle Breaking in Lingua Franca

Problem Description

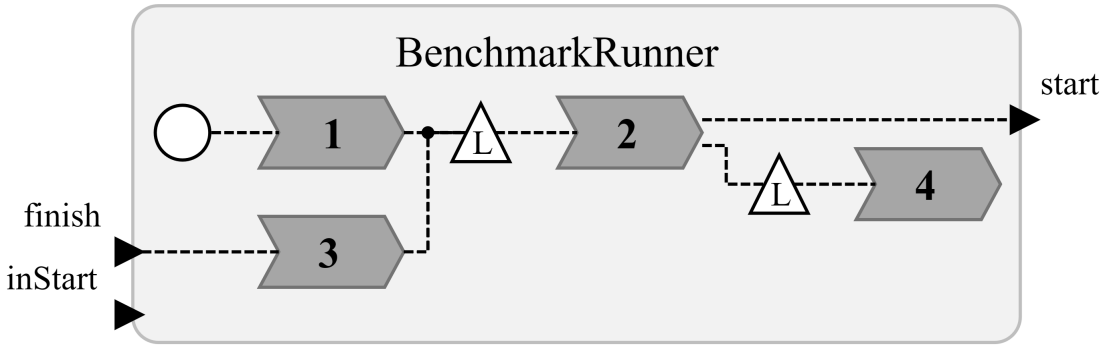
Previous works and studies suggest that the textual model, used for synthesis of graphical models, have an inherent order. Adapting the Cycle-Breaking stage of the Layered Algorithm to work with this order and to preserve it in the graphical model provides the creator of the models with some level of control and creates better graphical results. The results mentioned above originate in an analysis that has been done with SCCharts.
Lingua Franca is a modeling language with different types of nodes (mainly actions, reactions and reactors). In the textual representation, nodes are grouped based on their type. Additionally, the order of nodes in the textual representation is part of the semantics of Lingua Franca, as this order is used for scheduling, to ensure deterministic behavior.
This project focused on finding and comparing different approaches for Cycle Breaking. The general goal is to create better-synthesized models.

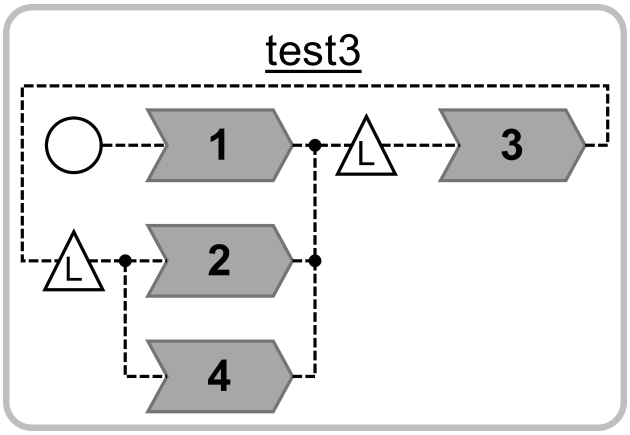
General Approach

While it initially was the goal to find a Cycle Breaking strategy which works for any modeling language with different Node-Types, Lingua Franca has many peculiarities and approaches presented here will focus on these peculiarities and make assumptions based on them.
The grouping of nodes in the textual representation yields the problem that comparing the Model Order between different Node-Types is not sensible.
Within a type the Cycle-Breaking strategy could simply utilize the Model-Order. Different approaches for edges between different Node-Types will be discussed here.
It has to be ensured, that the strategy removes all cycles between different Node-Types. The cycles within a type are handled by the Model-Order.
Another peculiarity of Lingua Franca is that node-types alternate, actions connect to reactions, which in turn connect back to actions.

Exemplary Models using the current Cycle Breaking Strategy.

This section will show two graphical models using the current strategy. This is meant as a reference when looking at the results of different approaches.





(Approach 0) Using the Depth-First Cycle Breaker

Considering that the Depth First Cycle Breaker and the Model Order Cycle Breaker produced the same results for the majority of the models for SCCharts, simply using the Depth First approach might be viable. This might be interesting, if the textual order of Lingua Franca models does not represent any idea of the model creator, due to the order being part of the semantics of Lingua Franca.

Approach 1 Node-Type Priority

Idea: Have a strict reversal order between different node types.

Define priorities (type-priorities) for the different Node-Types. If an edge starts in a node with a lower type priority than the target node, reverse this edge. This ensures, that there are no cycles between different Node-Types. However, due to the alternating nature of LF Node-Types, this leads to layers of node-types. The Model-Order is completely ignored, as edges are only reversed based on the type priority. This may also create large amounts of backwards-edges.

```
FOR v in V:
    FOR e in v.outgoing:
        if e.source.type_priority > e.target.type_priority:
            e.reverse()
        else if e.source.type_priority == e.target.type_priority:
            if e.source.model_order > e.target.model_order:
                e.reverse()
```

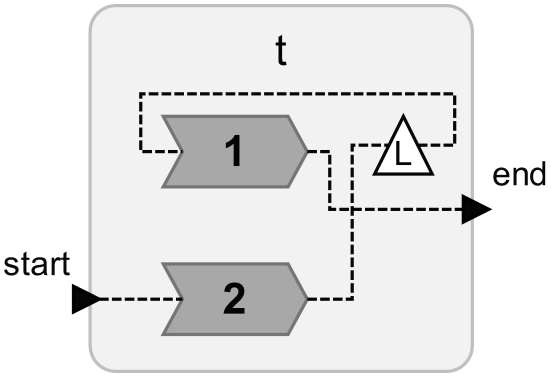
Advantages

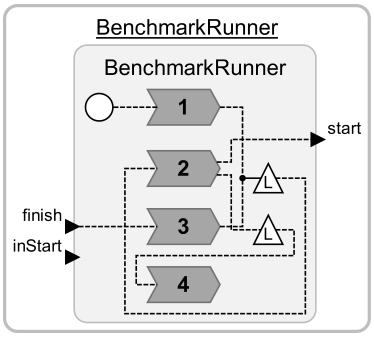
- Easy and fast.

Disadvantages

- Creates mostly isolated layers for different Node-Types.
- Bad performance regarding classical aesthetic criteria.

The following two images show the problems discussed above. Having unnecessary backward edges and creating layers for different node types.





Approach 2 Breadth-First or Depth-First as Secondary Order

Idea: Traverse the graph with Breadth-First-Search/Depth-First-Search and create a second ordering to use between different node-types.

Start a BFS/DFS from a selected node (e.g external port, start-up node), in general from sources, and create a second ordering. This ordering is used as a tie-breaker when comparing nodes of different types. Again this approach has the same problem as the first approach. As nodes alternate in LF, the Model-Order is completely ignored and edges are only reversed based on the ordering of BFS/DFS. Therefore for Lingua Franca this approach is equivalent to approach 0.

```
Queue bfs_queue
List order
function presort():
    for v in V.sources:
        bfs_queue.add(v)
    while (!bfs_queue.isEmpty):
        v = bfs_queue.poll()
        order.add(v)
        bfs(v)
function bfs(v):
    if(!v.was_visited):
        ordered.add(v)
        for e in v.outgoing:
            if !e.target.was_visited:
                bfs_queue.add(e.target)
```

```
for v in V:
    for e in v.outgoing:
        if e.source.type != e.target.type:
            if order.index_of(e.source) > order.index_of(e.target):
                e.reverse()
        else if e.source.model_order > e.target.model_order:
                e.reverse()
```

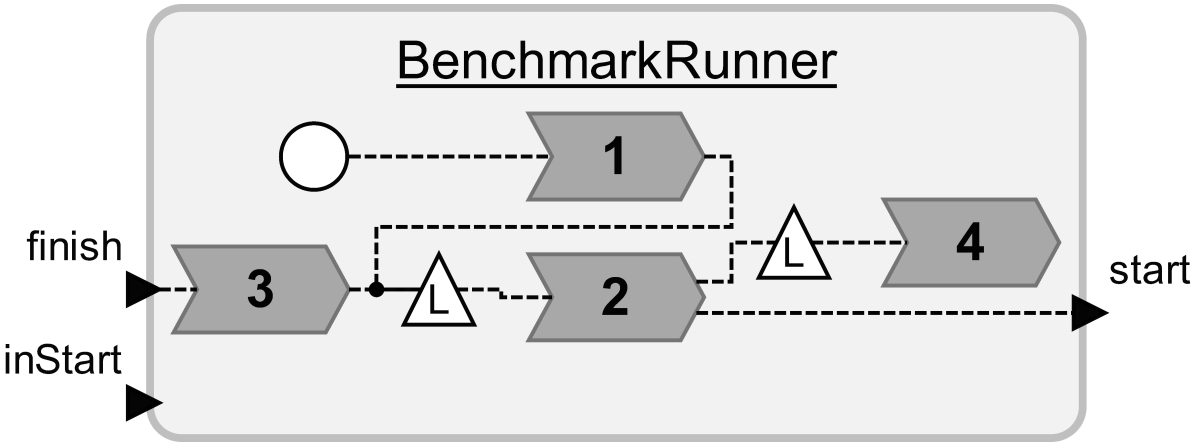
Advantages

- Easy and Fast

Disadvantages

- For a language like LF, where the different Node-Types mostly alternate, the Model Order is lost and this strategy is equivalent to the simple Depth First Cycle Breaker

The Following image shows, that this introduces problems of the Depth / Breadth First Cycle Breaker. The edge between reaction 1 and the logical action does not need to be reversed, but as the logical action was already visited in a previous step, the edge is reversed.



Approach 3 Model-Order Look Ahead

Idea: Start a Breadth-First-Search from every node, if a search-path reaches a node of the same type, compare the model order and reverse an edge / multiple edges based on this.
During the development of this Cycle Breaker, some additional features (in the form off preprocessor or intermediate steps) have been discovered. Some of them alter the layout, some of them are mainly used to reduce the search space. These features might be used in other strategies as well. These features and the differences they induce will be discussed here. However the basic algorithm will discussed first.

Since it is not sensible to compare the model order between different Node Types, the idea is to skip over these edges and check for the next node(s) of the same Node Type. For that a Breadth First Search is started at every node, to determine the successors of the same node type. In general, if any of the nodes reached by the BFS has a model order lower or equal (in the case of a self-loop) to the starting node, the initial edge has to be reversed.

```
for v in V:
    for e in v.outgoing:
        if e.source.type != e.target.type:
            //Using BFS search for the next node of the same type and compare
the model order of these nodes.
            nextNodes = getNextNodesWithGroupPriority(e.source, e.target)
            for seqNode in nextNodes:
                if e.source.model_order >= seqNode.model_order: // The "=" is
needed incase of a selfloop
                    e.reverse()
                    continue
        else if e.source.model_order > e.target.model_order:
                e.reverse()
```

Advantages

- Does not need a metric other than the Model Order.
- Achieved promising results in Test-graphs.
- Computationally inexpensive in LF. (due to alternating Node-Types)

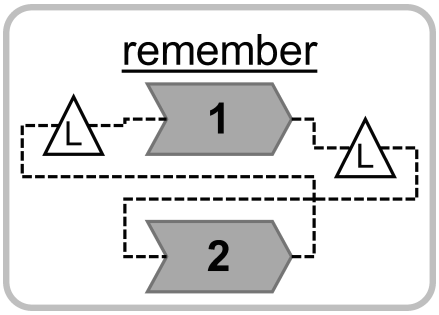
Disadvantages

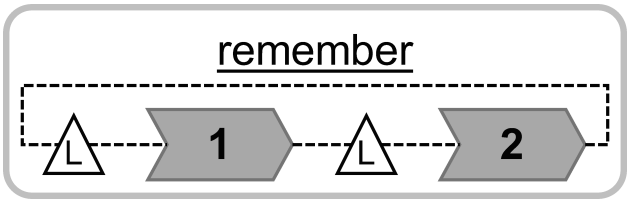
- In Languages where node-types alternate very infrequent or have unique nodes, may be computationally intensive.
- Some edge-cases may lead to unnecessary edge reversals.

Remembering previously reversed edges

As this approach starts a Breadth-First Search for every node, it is important to reduce the search space, especially for larger models. If an edge has been reversed previously, there is no need to check beyond this edge as all cycles containing this edge have already been broken. Additionally, this may reduce the number of backward edges, as shown below.

The first Image shows what happens when the BFS does not abort if it interacts with an edge already marked for reversal. This happens because the model order of the reactions and the model order of the actions induces a reversal.





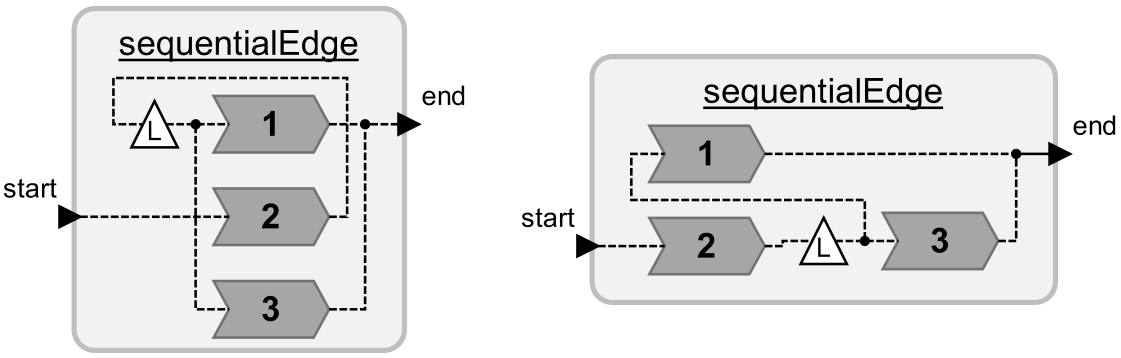
Reordering the Groups of Node-Types

In Lingua Franca the Model-Order of the reactions is of the highest interest. Selecting from which types of nodes we start with the strategy leads to different results if combined with the previously mentioned technique of remembering reversed edges. Therefore the default approach is to start with the reactions. Presorting the nodes additionally ensures determinism for this approach.

Reversing subsequent edges.

Instead of reversing the outgoing edge of the starting node s ,with type T, it is possible to reverse the incoming edge of the next node with type T.
This leads to longer straight layouts and allows more precise edge reversals, however, it might also lead to more backward edges.

The first image shows an example where due to one of the edges being of lower model order, the outgoing edge of reaction 2 is reversed. Reversing the next incoming edge instead of the outgoing edge may results in a more granular control over edge reversals, as shown in the second example.

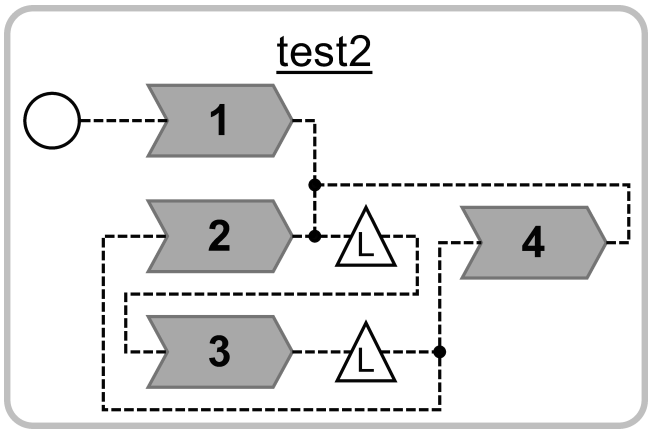


This, however, creates other problems, shown in the following section.

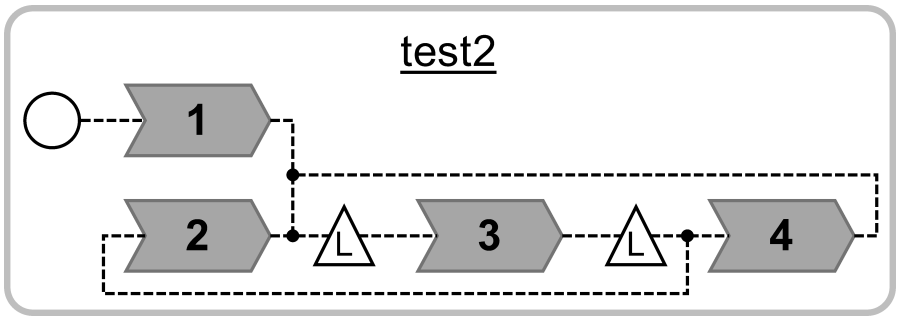
Fallback-edges

Using sequential edge reversals, as described above, may lead to conflicts regarding the model order, as shown below.

The model order of reaction 2 and reaction 3 prefers the edge between the upper logical action and reaction 3 not to be reversed. The model order of reaction 4 however leads to the reversal of this edge.



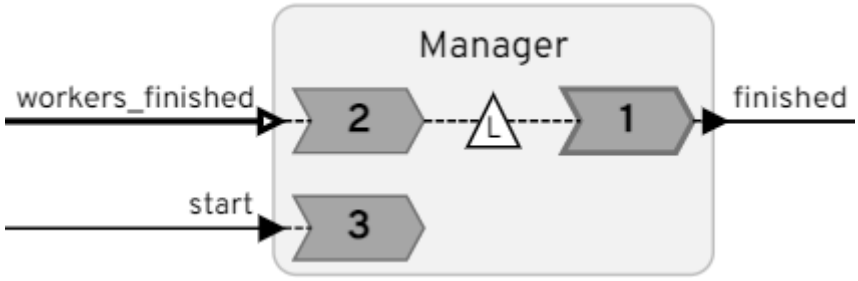
This problem can be dealt with, by fixing edge directions. During the Breadth First Search of reaction 2, the layout direction of the problematic edge is marked as fixed direction. When the BFS from reaction 4 reaches this edge, it is recognized, that the edge should be reversed to break a cycle, but it is marked as fixed. In this case the outgoing edge of reaction 4 is chosen as fallback edge to reverse. During this step no other edge is reversed other than the fallback edge. This is shown in the following image.



One might like the first or the second better. For this domain experts should be surveyed. (Which will be done at a later time.)

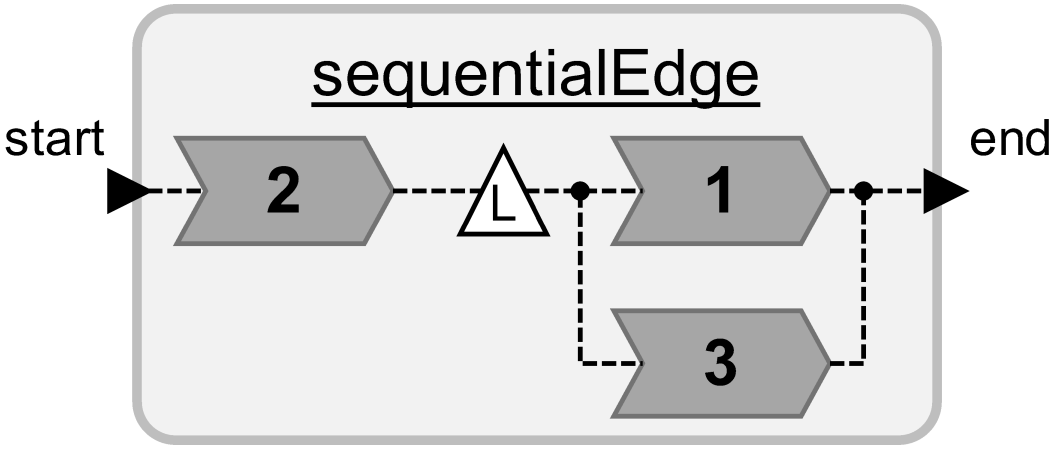
Strict Model-Order is not always applicable

In the following example reaction 1 has to be defined before reaction 2 due to determancy, however strict Model-Order edge reversal would now force the reversal of an edge.



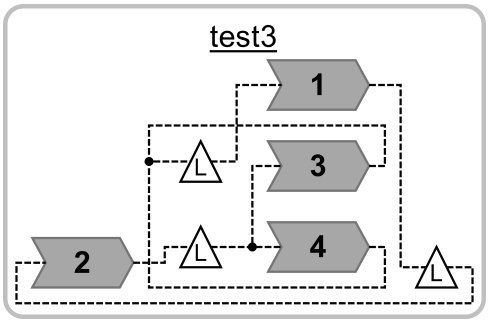
To cope with this problem edges are marked as part of a loop if they are part of a strongly connected component. Only if an edge is part of a strongly connected component it is part of a cycle. Edge targets are therefore only checked if the edge connects nodes within a strongly connected component. This allows the following layout, for the graph shown in the sequential edge section. For this edges are marked using the Tarjan Algorithm for strongly connected components.
(https://en.wikipedia.org/wiki/Tarjan%27s_strongly_connected_components_algorithm)

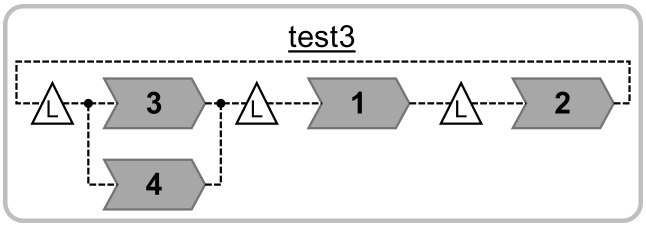
No edge reversal is needed for an acyclic graph.



The following example shows that this strategy may still reverse too many edges.

The right image shows, that one edge reversal suffices, to break all cycles. With this strategy all edges are reversed if they are initially part of a strongly connected component and go against the model order.





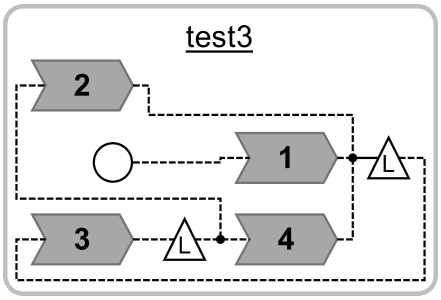
The following approach eliminates this problem. It however comes with a (significant) runtime trade-of.

Approach 4 Greedy Strongly Connected Components

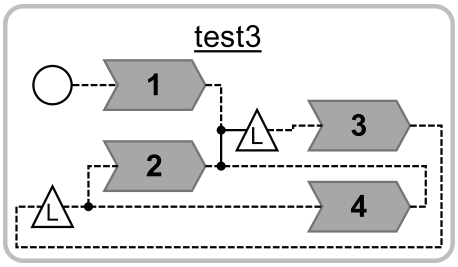
Idea: Use Tarjans algorithm and reverse the edges off the node with the highest or lowest model order. Repeat this process, until no strongly connected components are left.

This strategy utilizes the fact that reversing all incoming or outgoing edges of a node in a strongly connected component breaks all circles containing this node. After reversing the edges, the Tarjan algorithm is used again. These steps are repeated until no strongly connected component may be found. Since at least n nodes are cleared from all cycles in every repetition (n being the amount of SCC found in this iteration), after V repetitions no further SCC may be found, resulting in a worst-case runtime of O((V+E)*V). However, this is extremely unlikely as it only applies to fully connected graphs.

There is one major parameter one could alter for this approach, choosing the node. Concerning Model-Order there are two main approaches:
Select the node with the lowest Mode-Order and reverse all of the incoming edges originating in nodes that are part of the strongly connected component. As seen in the following image.



Select the node with the highest Model-Order and reverse all of the outgoing edges ending in a node that is part of the strongly connected component, as seen here:



This does not eliminate all of the unnecessary edge reversals, but it may drastically reduce them. One could try to improve the node selection for this method, disregarding Model-Order for a heuristic approach like the one used in the Greedy Cycle Breaker. This could create a very good approach in regard of edge reversals.

Outlook

Improvement for the MO-Look ahead Cycle Breaker (smart selection of initial or sequential edge Reversals)
Quantitative analysis using GrAna.
Qualitative analysis, by surveying domain experts.