Event Graph Representation with Deep Learning models

Contents

[Event graphs 2](#_Toc115305290)

[Graph Representation with Deep Neural Networks 4](#_Toc115305291)

[Our concept to represent event graphs 4](#_Toc115305292)

[Design options of Metagraphs 5](#_Toc115305293)

[Options for regularization 7](#_Toc115305294)

[Our sequential generation algorithm 8](#_Toc115305295)

[Interpretation of steps and random variables 8](#_Toc115305296)

[Depiction of the steps taken during the generation process 9](#_Toc115305297)

[Step by step example 10](#_Toc115305298)

[Step 1 (0 - Node, 0 - Event) 11](#_Toc115305299)

[Step 2 (1 - Activity attribute, 0 - Create Appl.) 11](#_Toc115305300)

[Step 3 (6 – Joint edge space, 4 – No edge) 12](#_Toc115305301)

[Step 4 (0 – Node, 1 – Application) 12](#_Toc115305302)

[Step 5 (6 – Joint edge space, 2 – :CORR) 13](#_Toc115305303)

[Step 6 (6 – Joint edge space, 8 – No edge) 13](#_Toc115305304)

[Step 7 (0 – Node, 2 – Resource) 14](#_Toc115305305)

[Step 8 (6 – Joint edge space, 1 – :CORR) 14](#_Toc115305306)

[Step 9 (6 – Joint edge space, 12 – No edge) 15](#_Toc115305307)

[Step 10 (0 – Node, 0 – Event) 15](#_Toc115305308)

[Step 11 (1 – Activity attribute, 1 – Appl. Ready) 16](#_Toc115305309)

[Step 12 (6 – Joint Edge space, 4 – :CORR) 16](#_Toc115305310)

[Step 13 (6 – Joint Edge space, 4 – :CORR) 17](#_Toc115305311)

[Step 14 (6 – Joint Edge space, 3 – :DF) 17](#_Toc115305312)

[Step 15 (5 – Correlation attribute, 0 – Application) 18](#_Toc115305313)

[Step 16 (6 – Joint Edge space, 3 – :DF) 18](#_Toc115305314)

[Step 17 (5 – Correlation attribute, 1 – Resource) 19](#_Toc115305315)

[Step 18 (6 – Joint edge space, 0 – No edge) 19](#_Toc115305316)

[Step 19 (0 – Node, 0 – Event) 20](#_Toc115305317)

[Step 20 (1 – Activity attribute, 2 – Create Offer) 20](#_Toc115305318)

[Step 21 (6 – Joint edge space, 20 – No edge) 21](#_Toc115305319)

[Step 22 (0 – Node, 3 – Offer) 21](#_Toc115305320)

[Step 23 (6 – Joint edge space, 18 – :CORR) 22](#_Toc115305321)

[Step 24 (6 – Joint edge space, 24 – No edge) 22](#_Toc115305322)

[Step 25 (0 – Node, 0 – Event) 23](#_Toc115305323)

[Step 26 (1 – Activity attribute, 3 – Send Offer) 23](#_Toc115305324)

[Step 27 (6 – Joint edge space, 20 – :CORR) 24](#_Toc115305325)

[Step 28 (6 – Joint edge space, 19 – :DF) 24](#_Toc115305326)

[Step 29 (5 – Correlation attribute, 2 – Offer) 25](#_Toc115305327)

[Step 30 (6 – Joint edge space, 28 – No edge) 25](#_Toc115305328)

[Step 31 (0 – Node, 0 – Event) 26](#_Toc115305329)

[Step 32 (1 – Activity attribute, 4 – Offer Cancelled) 26](#_Toc115305330)

[Step 33 (6 – Joint edge space, 20 – :CORR) 27](#_Toc115305331)

[Step 34 (6 – Joint edge space, 8 – :CORR) 27](#_Toc115305332)

[Step 35 (6 – Joint edge space, 27 – :DF) 28](#_Toc115305333)

[Step 36 (5 – Correlation attribute, 2 – Offer) 28](#_Toc115305334)

[Step 37 (6 – Joint edge space, 15 – :DF) 29](#_Toc115305335)

[Step 38 (5 – Correlation attribute, 1 – Resource) 29](#_Toc115305336)

[Step 39 (6 – Joint edge space, 32 – No edge) 30](#_Toc115305337)

[Step 40 (0 – Node, 4 – No node) 30](#_Toc115305338)

[Code 31](#_Toc115305339)

[Bibliography 32](#_Toc115305340)

# Event graphs

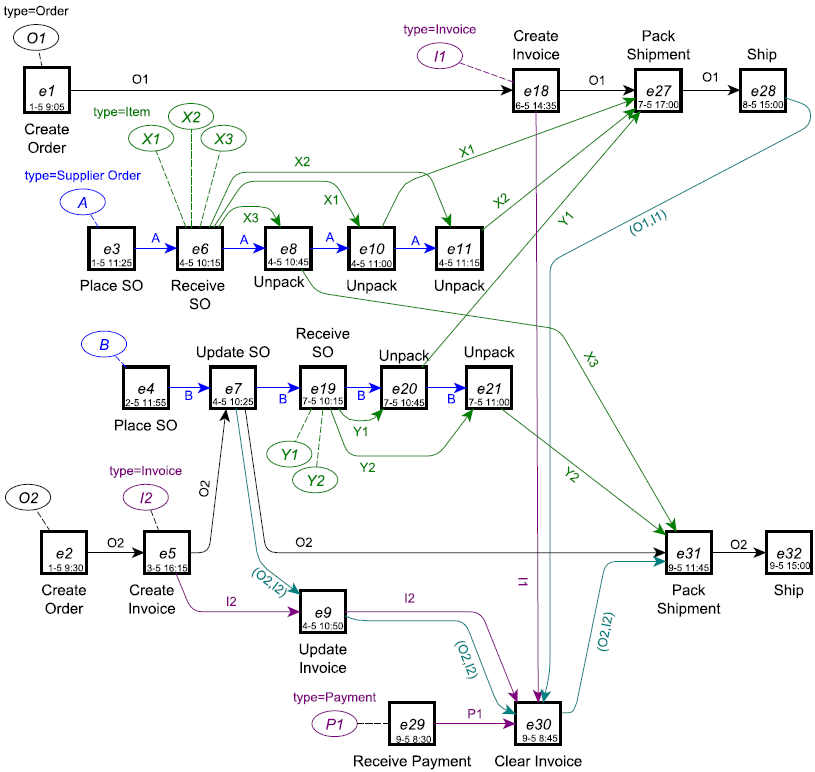


Figure 1 Event graph example of a retail order process

Event graph representation of event data over multiple entities (Fahland, Process Mining over Multiple Behavioral Dimensions with Event Knowledge Graphs, 2022). We aim to be 100% conform with Dirk’s definitions a notations. Terms which are essentially needed:

* Event (node)
  + Timestamp attribute
  + Activity label attribute
* Entity type and Entity (node)
* Relations:
  + Correlation relation between an Event and an Entity
  + Directly-Follows relation between to Events per Entity
  + Related relation between two Entities
* Local Directly-Follows path
* Derived Entity Types
* Starting, intermediate, ending Events in a DF-path
  + Entity creation, handover, and completion
* Interaction and no interaction among Entities
* Entities synchronization in an Event

# Graph Representation with Deep Neural Networks



Figure 2 Neighborhood aggregations in a graph

* Related work and graph learning tasks can be mastered with the Stanford lectures of (Leskovec, 2022)
* Graph generation paper is the basis of our solution (Yujia Li, 2018)
* Relational graph paper is the fundamental message passing solution on relational graphs (Michael Schlichtkrull, 2018)
* There are two major graph frameworks for Graph Neural Networks:
  + Deep Graph Library (DGL): (Deep Graph Library, 2022)
  + PyTorch Geometric: (PyTorch Geometric, 2022)

The two frameworks offer the same functionalities and we chose DGL because of an existing generative model implementation (Mufei Li, 2022).

# Our concept to represent event graphs

BPIC 2017 example with the 2 traces (Table 1 & Figure 3) also in (Fahland, Event Graph of BPI Challenge 2017, 2022).



Figure 3 Event graph representation of Table 1

Table 1 Two overlapping traces from BPIC 2017 for event graph representation

|  |  |
| --- | --- |
| Application\_1258227511  Event nodes noted by ex |  |
| Application\_1692826468  Event nodes noted by fx |  |

# Design options of Metagraphs

An Event graph is a labeled property graph on the one hand, and a heterogenous multigraph on the other hand. Hence, a graph schema so called metagraph is always defined of it. A basic metagraph of an Event graph as heterogenous graph is visualized in Row 1, Column 1 in Table 2. That is the most general version which does not incorporate node or edge attributes, although an Event graph has an activity attribute of Event nodes, and a correlation attribute of df-edges. Table 2 depicts eight more options for introducing these attributes into the metagraph heading to a more specific graph schema direction. During implementation in DGL, a graph schema has to be defined, so that constraints our stochastic model during training and inference to be conform. The metagraph in Row 3, Column 3 is the most specific option. It would provide the highest regularization which helps but it introduces the highest amount of node types and relations into the heterogenous graph which would make the representation learning more difficult. Our choice is Row 2, Column 2 which keeps the metagraph the most general and it introduces/defines the necessary attributes. The drawback is that for the representation of these attributes more steps (and predictions) are needed during the sequential generation process. Furthermore, an implementation detail but in DGL all nodes and edges have to have these attributes defined besides the Event nodes and df-edges. The attribute values can/have to be arbitrary for another node and edge types, but it has to be considered not to introduce any false patterns which would interfere with the representation learning. A safe solutions is to define a constant and out of real-range value, then always fill these attributes with that.

Table 2 Metagraph types

|  |  |  |  |
| --- | --- | --- | --- |
|  | Column 1:  No activity label of event node | Column 2:  Activity label is an event node attribute | Column 3:  Introducing additional event-node categories per activity labels |
| Row 1:  df-edge has no explicit correlation information |  |  |  |
| Row 2:  Correlation information is a df-edge attribute |  |  |  |
| Row 3:  Introducing additional df-edge categories per entity types |  |  |  |

# Options for regularization



Figure 4 Metagraph for regularization

The metagraph naturally provides regularization during training and inference. It is technically not possible to insert components into the GDL graph object which is not conform with the metagraph. It is implemented that during graph generation a more-likely prediction can happen to be rejected and substituted with a less-likely candidate to stay conform with the metagraph.

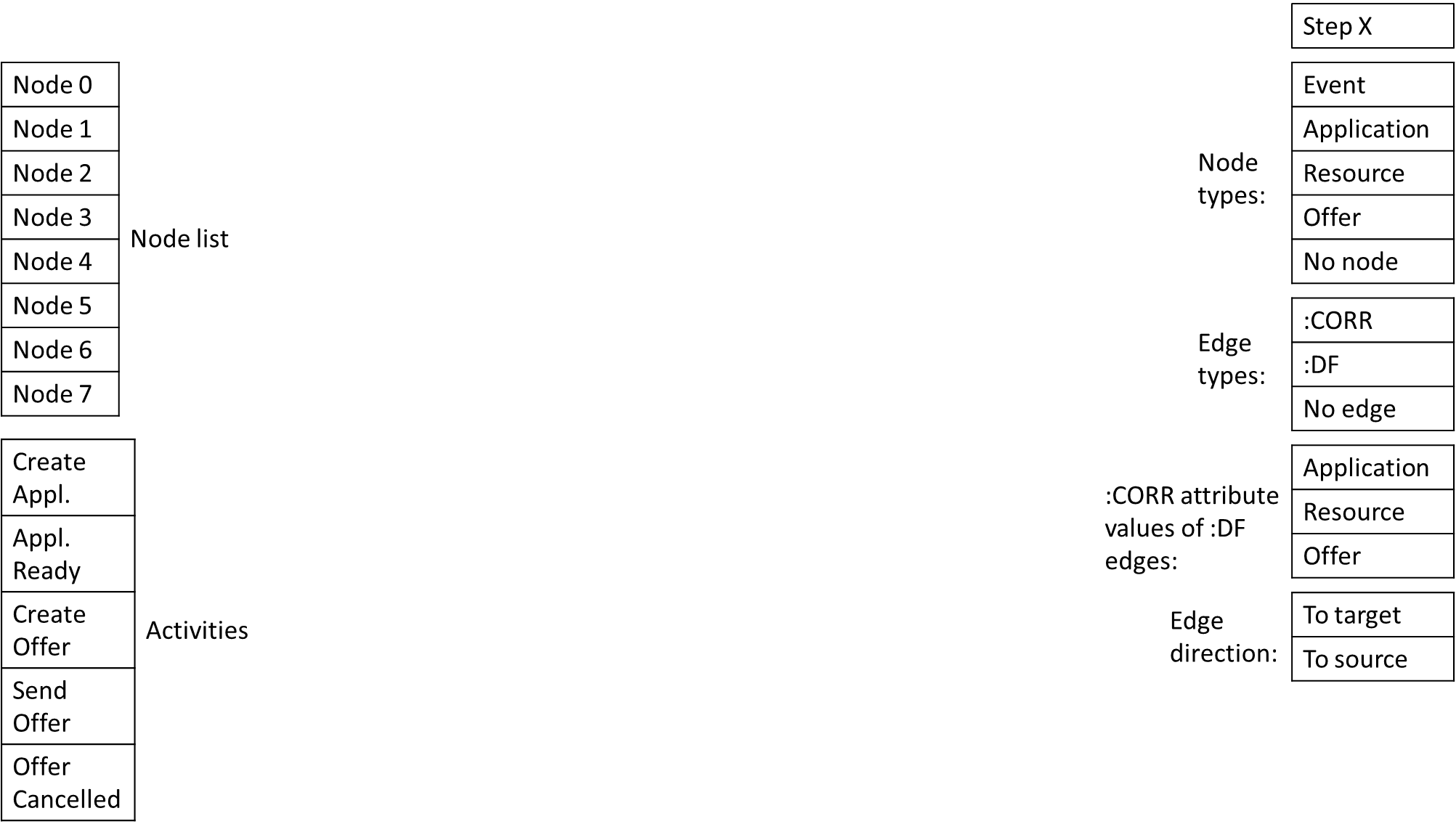
Furthermore during Event graph generation, there could be several components considered together as an atomic step:

* An Event node is always correlated to at least one Entity node (see Figure 4).
* If an Event node E1 is correlated to an Entity node which is also correlated to a set S of another Event nodes, then there must be at least one df-edge connecting E1 to one of S.

# Our sequential generation algorithm

Our generation methods follows (Yujia Li, 2018) but generalizes it to labeled property graphs, and heterogenous multigraphs.

## Interpretation of steps and random variables



* Node list: a dynamic (growing/shrinking by time) list of represented graph nodes. It serves a basis for predicting where an edge (belonging to the last-generated node) is connected to.
* Activities: a priori known set of activity labels. It is modelled as a categorical random variable and its value is stored in the activity label attribute of event nodes.
* Node types: a priori known set of graph node types. It is modelled as a categorical random variable. Its categories (i.e. sampling from it) expresses to predict whether the graph (in its current state) is expected to have one more node (and which type). That is why it contains the ‘No node’ category to express the case when the graph is complete.
* Edge types: a priori known set of graph edge types. It is modelled as a categorical random variable. Its categories (i.e. sampling from it) expresses to predict whether the graph (in its current state) is expected to have one more edge (and which type) connected to the last-generated node. That is why it contains the ‘No edge’ category to express the case when there is no additional connecting edge expected to the last-generated node in the current state of the graph.
* Entity types (:CORR attribute values of :DF edges): a priori known set of entity types. It is modelled as a categorical random variable and its value is stored in the :CORR attribute of :DF type of edges.
* Edge direction: it is a binary random variable expressing the direction of the last-generated edge in the viewpoint of the last-generated node.

For edge predictions, three random variables are jointly modelled because of implementation reasons in DGL:

1. The **Edge types** random variable is principally needed
2. An edge needs to be directed, so the **Edge direction** random variable is needed
3. An edge is connected to a node (apart of the last-generated one), so the **Node list** (for the another node) random variable is needed

Joint modelling of these three variables results the Cartesian space of:

Edge space **= Edge types** x **Edge direction** x **Node list**

meaning to express the edge type and the edge directionality and the connecting node, in the viewpoint of the last-generated node (that node of the edge is always fixed).

## Depiction of the steps taken during the generation process

1. Add node? (Sampling from Node types)
   1. If Node type is *not* ‘No node’:
      1. If Node type is ‘Event’:
         1. Sampling from Activities
         2. Fill Activity attribute of node
      2. If Node type is *not* ‘Event’:
         1. Fill Activity attribute of node with random or pre-defined padding value
      3. Add node to Node list and add node object to DGL graph object
      4. Add edge? (Sampling from the Edge space)
         1. If edge type is *not* ‘No edge’:
            1. If edge type is ‘:DF’:

Sampling from Entity types

Fill :CORR attribute of edge

* + - * 1. If edge type is *not* ‘:DF’:

Fill :CORR attribute of edge with random or pre-defined padding value

* + - * 1. Add edge object to DGL graph object connecting the current/last node and a sampled node from Node list
        2. GOTO 1.a.iv
      1. If edge type is ‘No edge’:
         1. GOTO 1
    1. GOTO 1
  1. If Node type is ‘No node’: graph is complete and we are done with generation

# Step by step example

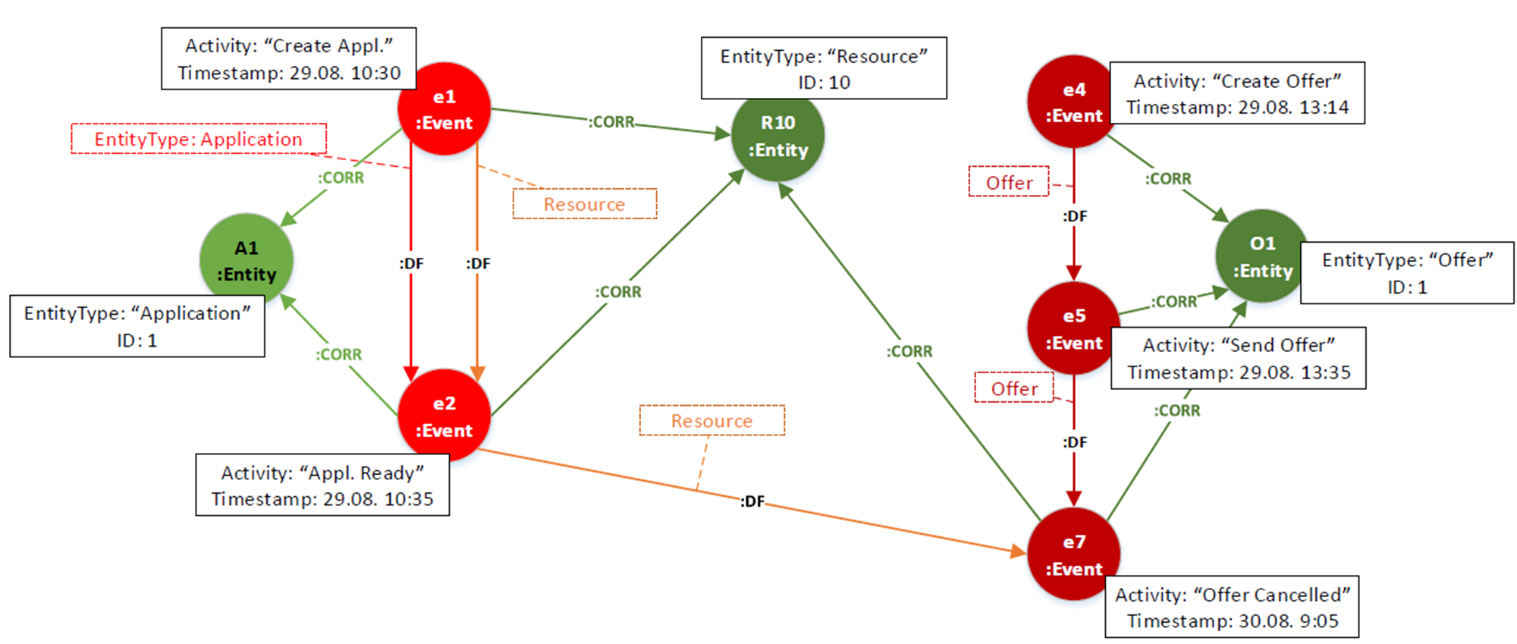


Figure 5 Example graph to detail our algorithm

Let’s take the graph in Figure 5. There are multiple possible serializations of the graph. The timestamp attribute of event nodes provides a natural order but the entity nodes and edges can be ordered arbitrarily. I arbitrary chose to generate in the following fashion:

1. Event node with Activity attribute
2. Its correlating Entity node(s)
3. Corresponding :CORR edge(s)
4. :DF edge(s) to existing/previous Event node(s) with the Entity type edge attribute(s)
5. GOTO 1

The actions are encoded into a list of two-tuples. Where in a tuple, the first number expresses the index of the algorithmic step, and the second one expresses the category index of the corresponding categorical variable.

See the definitions in Figure 6:

<https://github.com/ketyi/dgl/blob/671b1aae72d1c79a09e2eb07e187c9f7876770d7/examples/pytorch/r-dgmg/heterograph.py#L11>

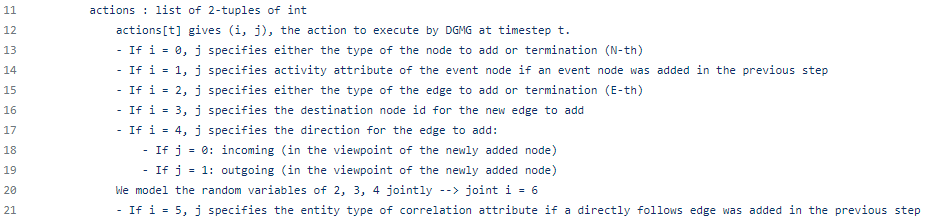
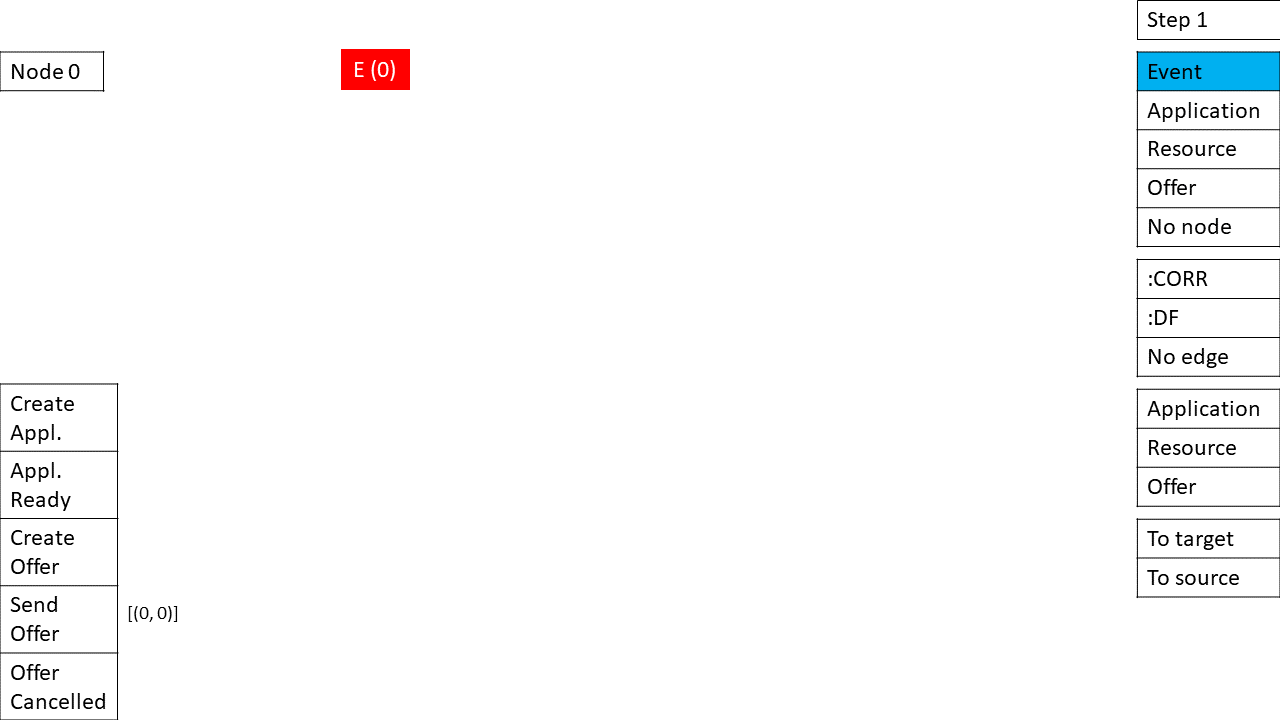


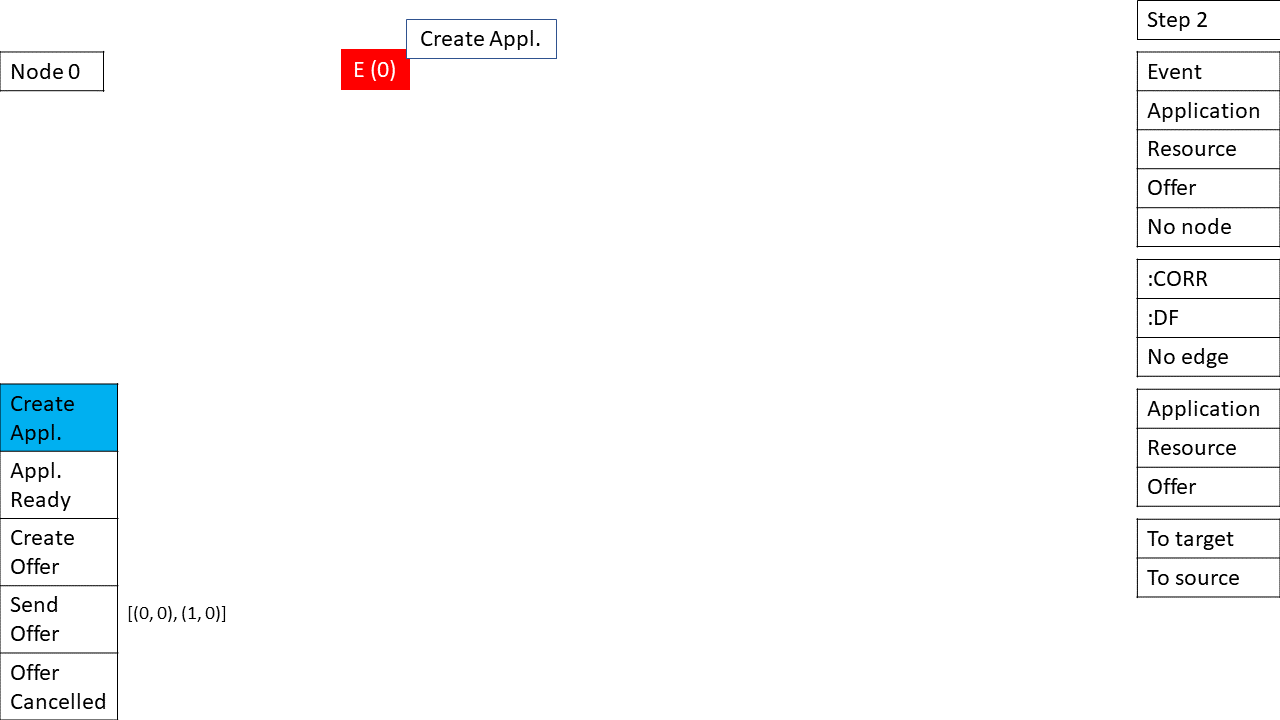
Figure 6 Definition of actions

The example consists of 40 steps.

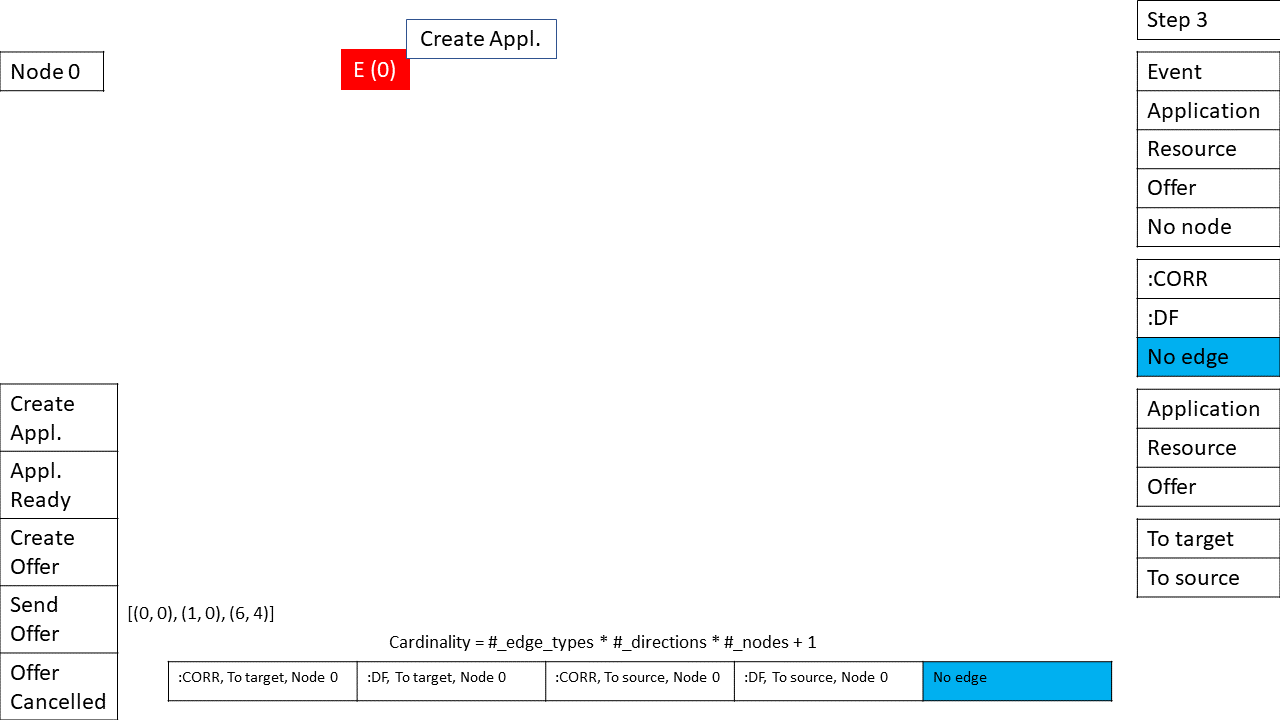
## Step 1 (0 - Node, 0 - Event)



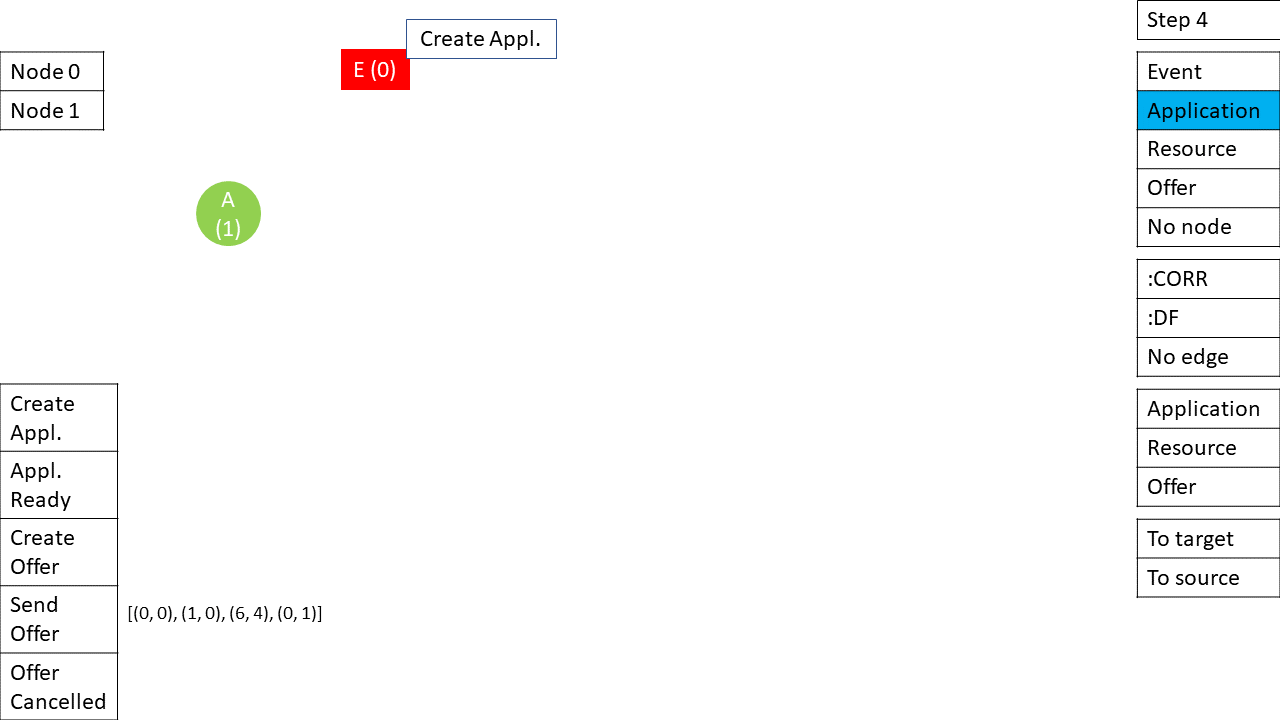
## Step 2 (1 - Activity attribute, 0 - Create Appl.)



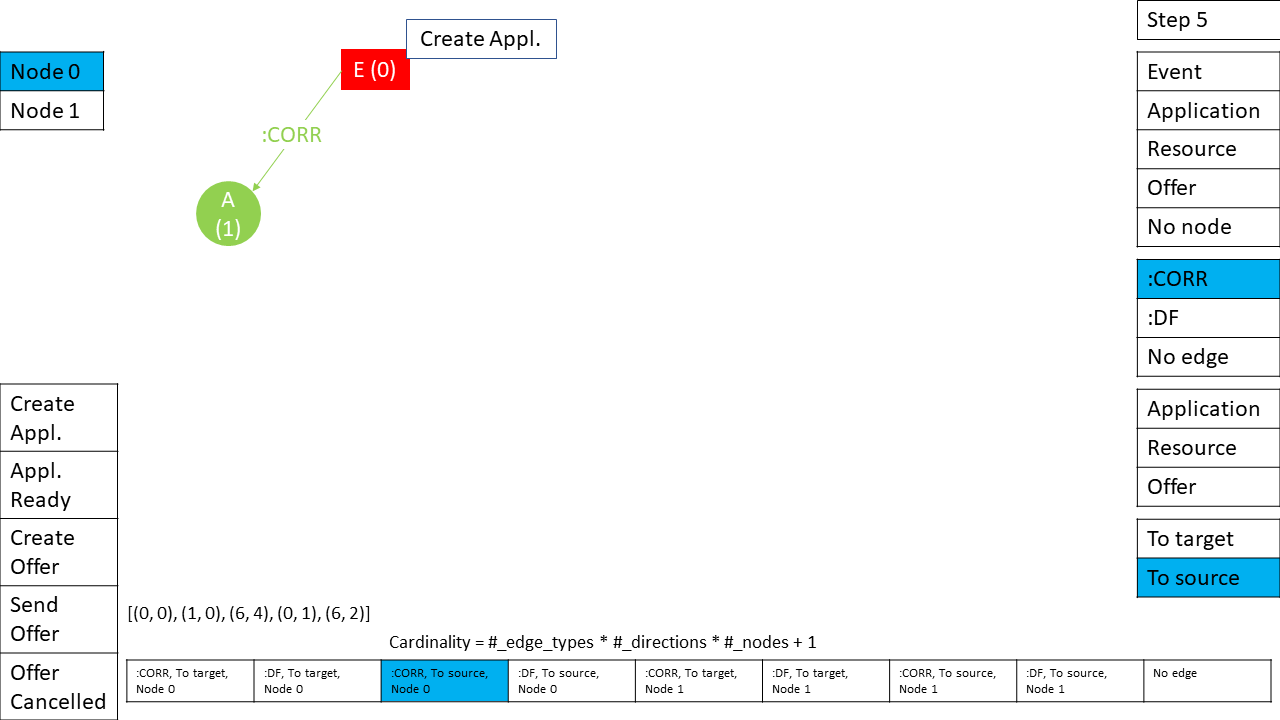
## Step 3 (6 – Joint edge space, 4 – No edge)



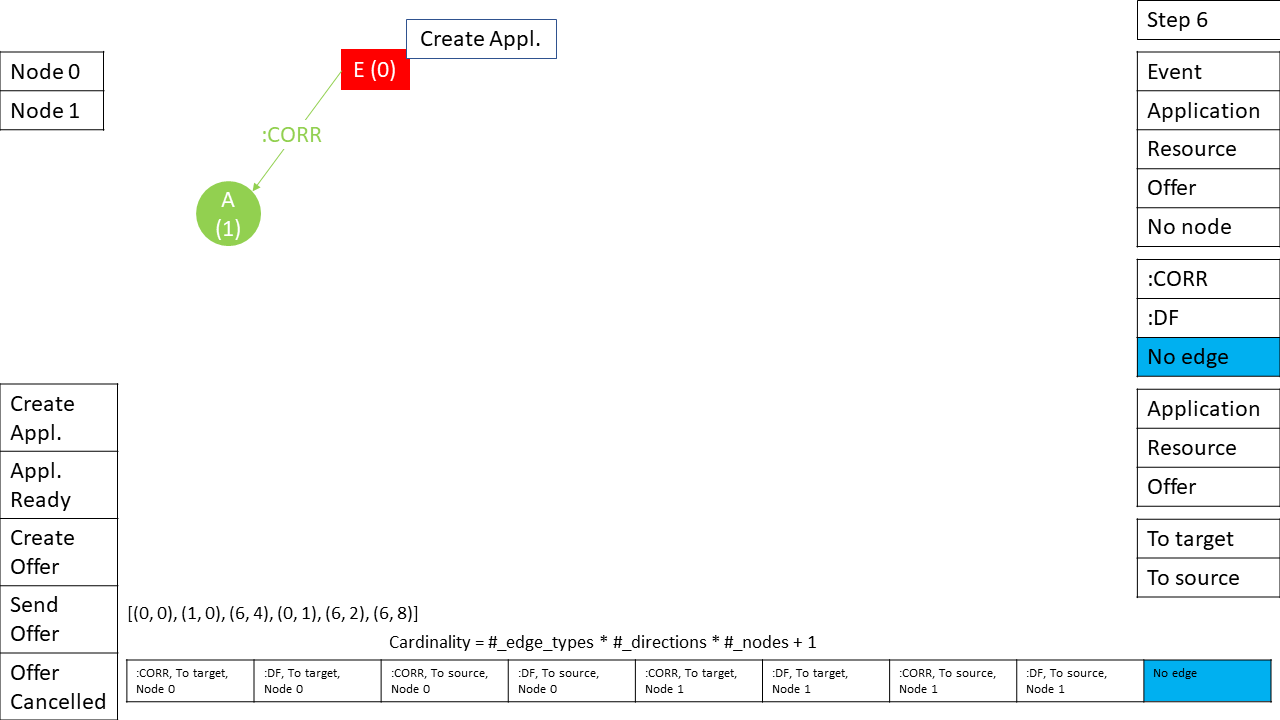
## Step 4 (0 – Node, 1 – Application)



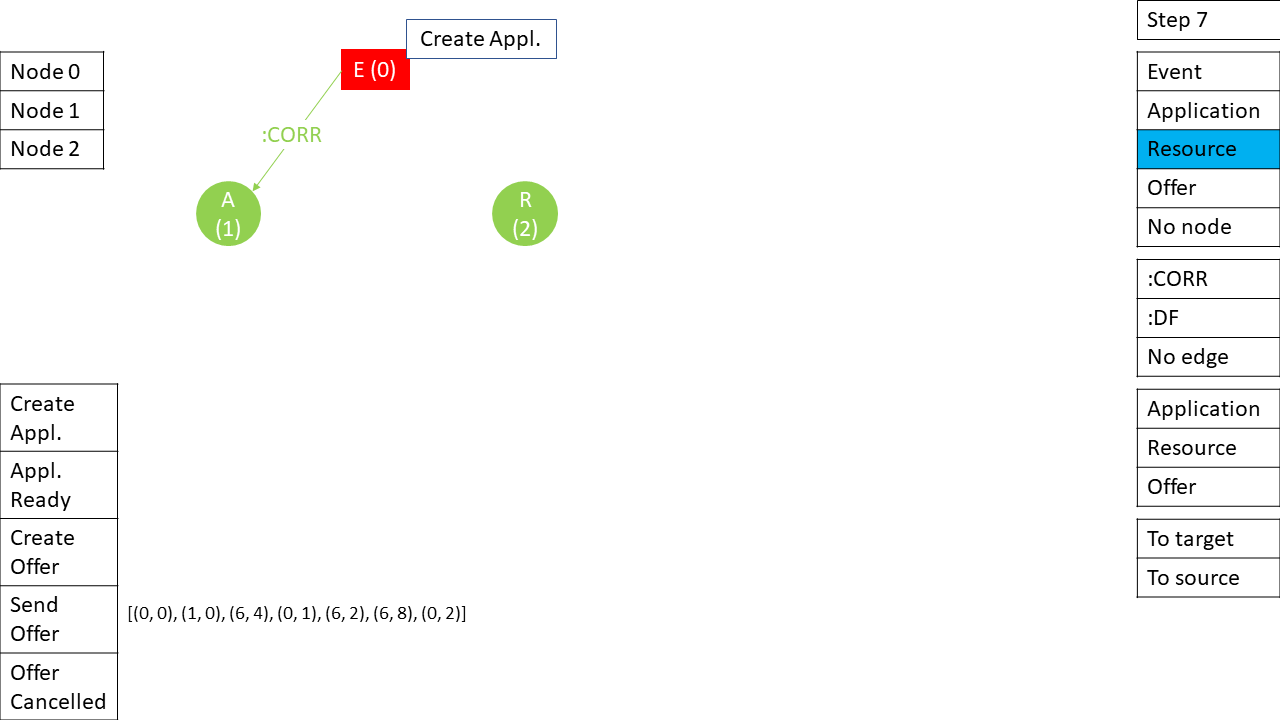
## Step 5 (6 – Joint edge space, 2 – :CORR)



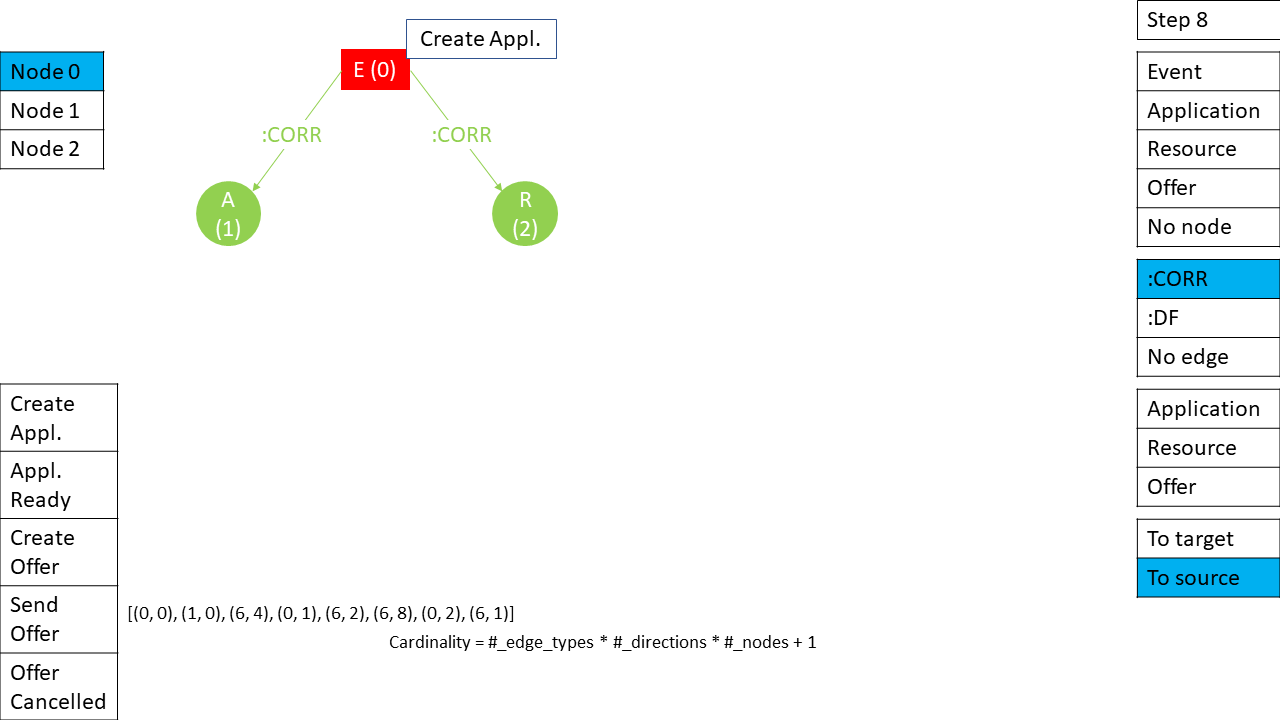
## Step 6 (6 – Joint edge space, 8 – No edge)



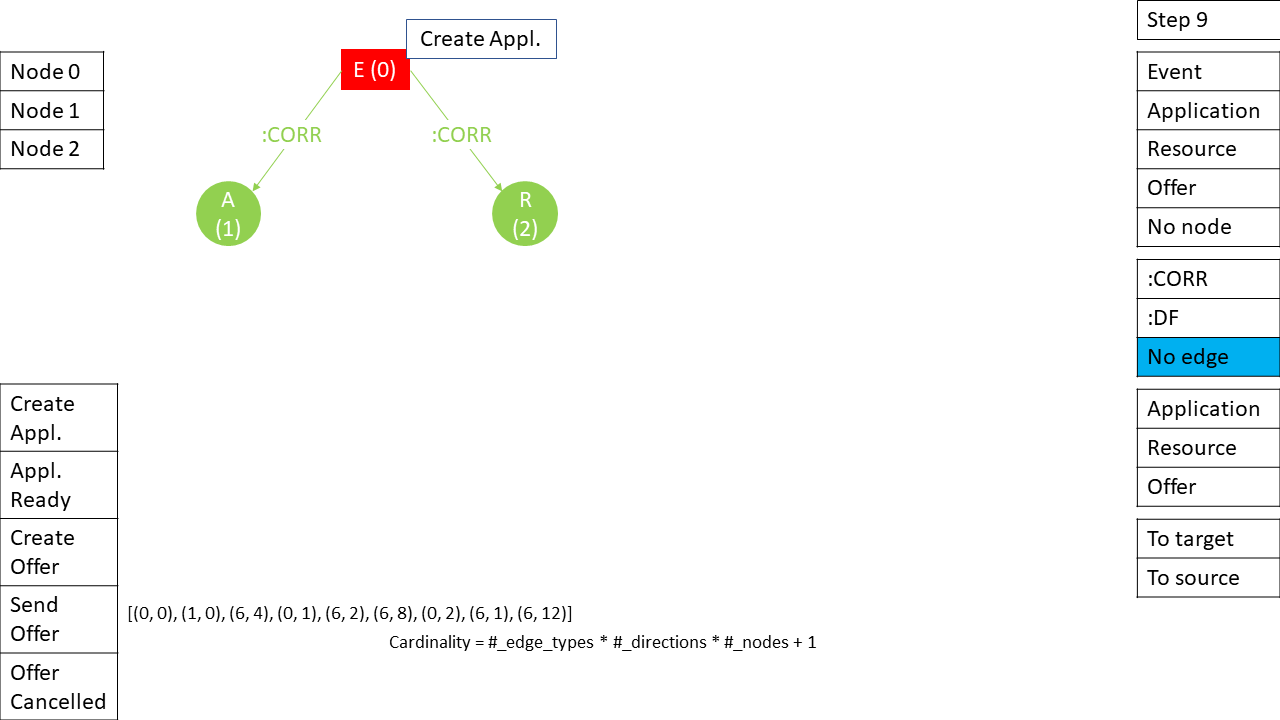
## Step 7 (0 – Node, 2 – Resource)



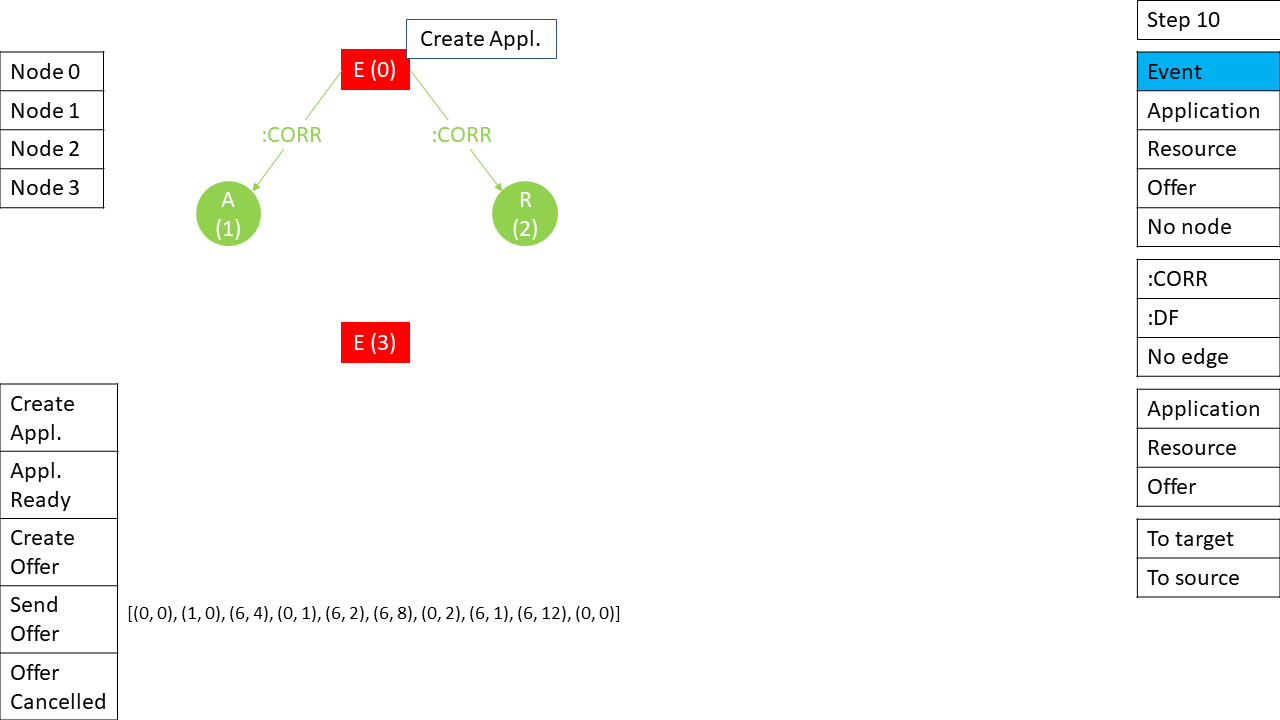
## Step 8 (6 – Joint edge space, 1 – :CORR)



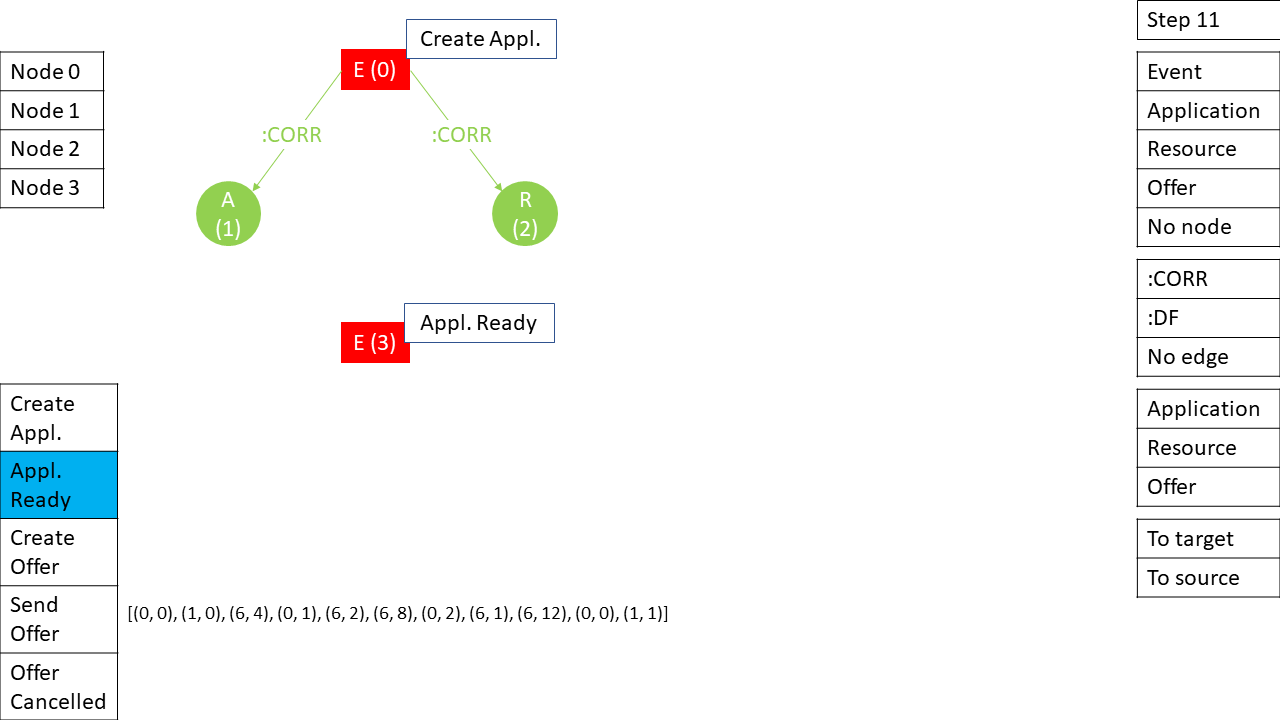
## Step 9 (6 – Joint edge space, 12 – No edge)



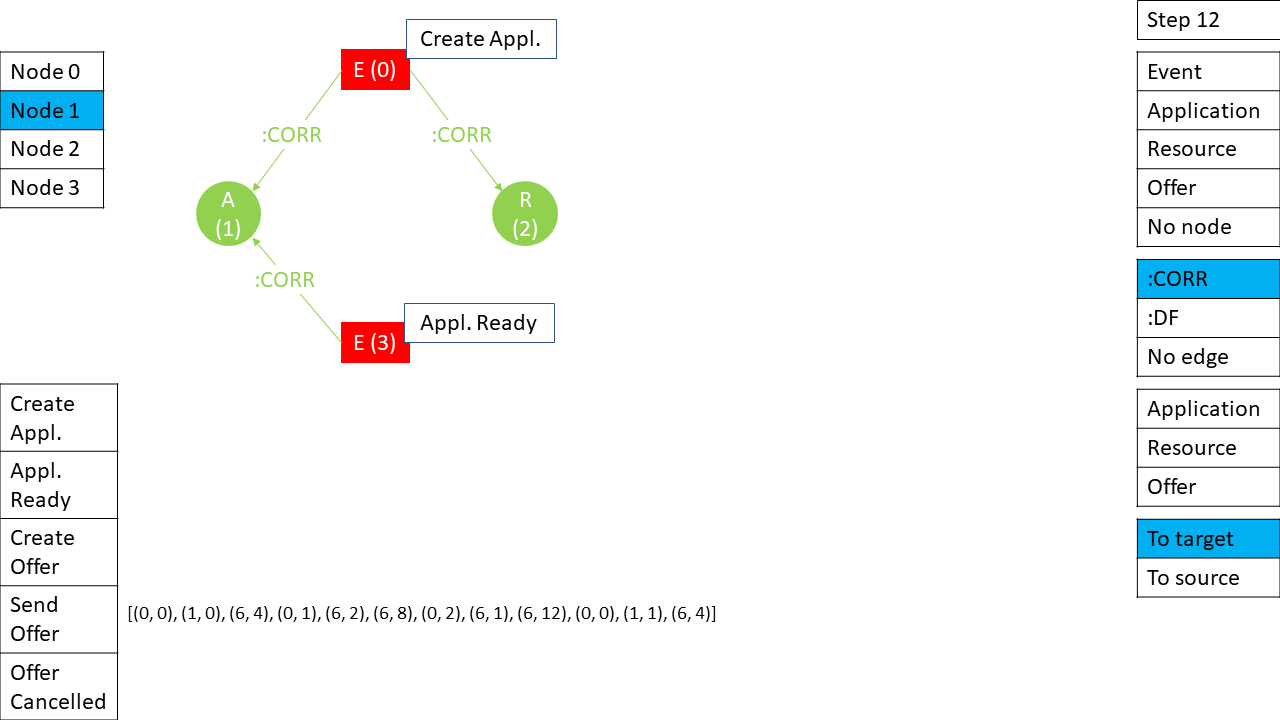
## Step 10 (0 – Node, 0 – Event)



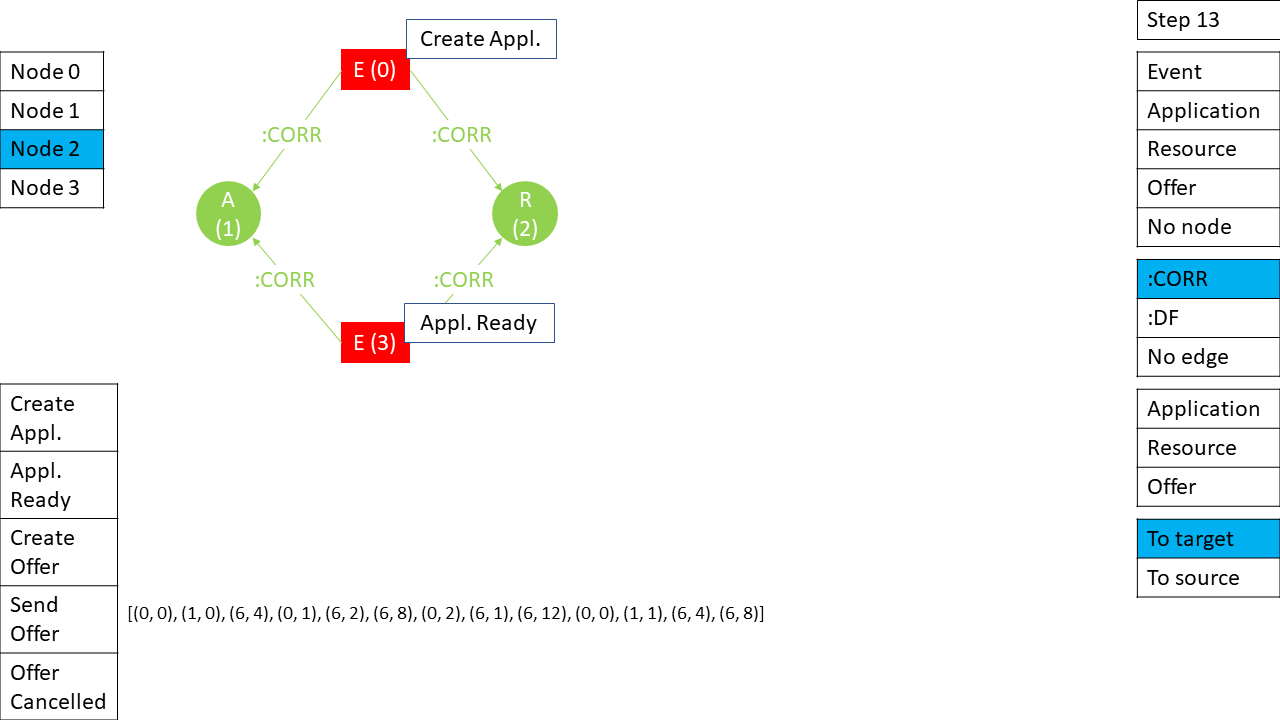
## Step 11 (1 – Activity attribute, 1 – Appl. Ready)



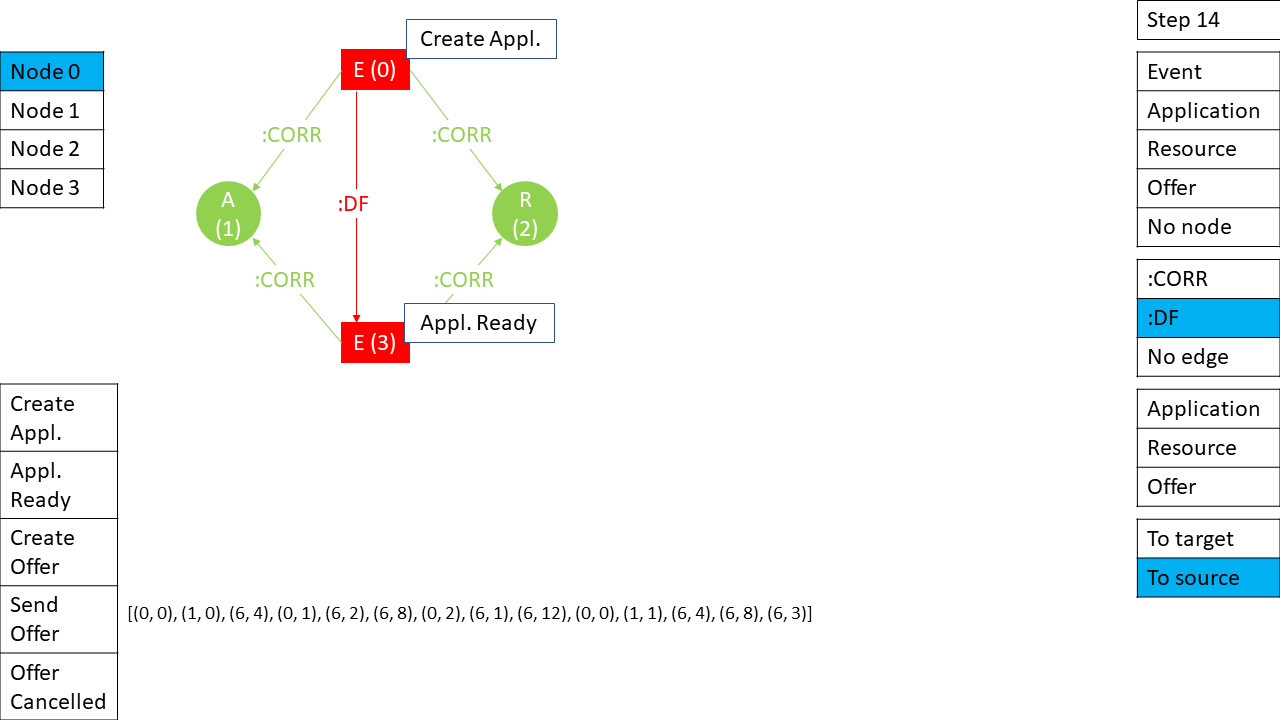
## Step 12 (6 – Joint Edge space, 4 – :CORR)



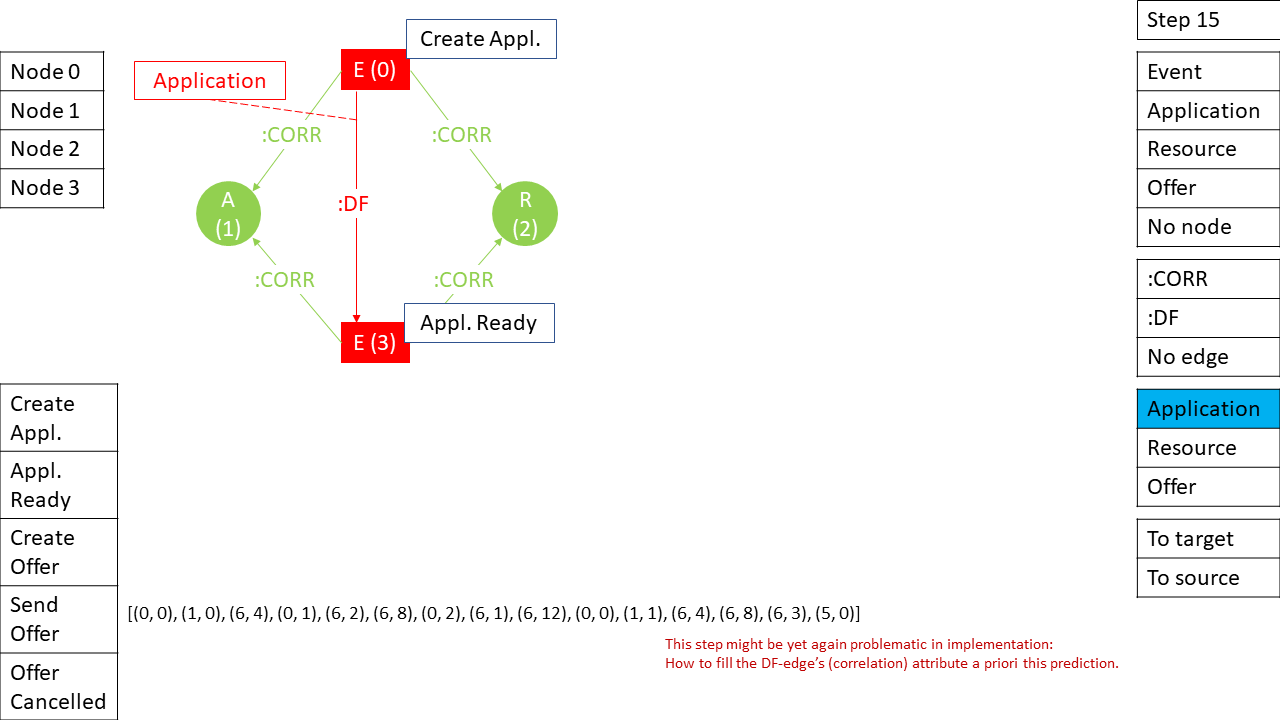
## Step 13 (6 – Joint Edge space, 4 – :CORR)



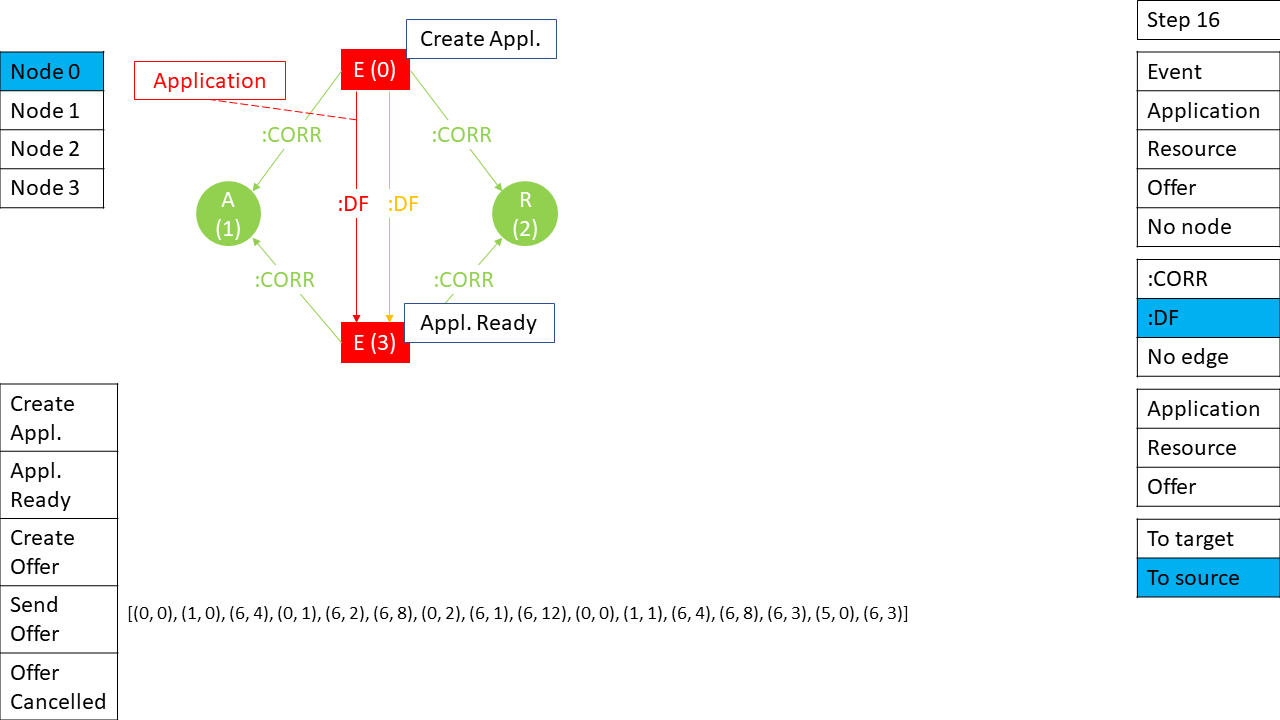
## Step 14 (6 – Joint Edge space, 3 – :DF)



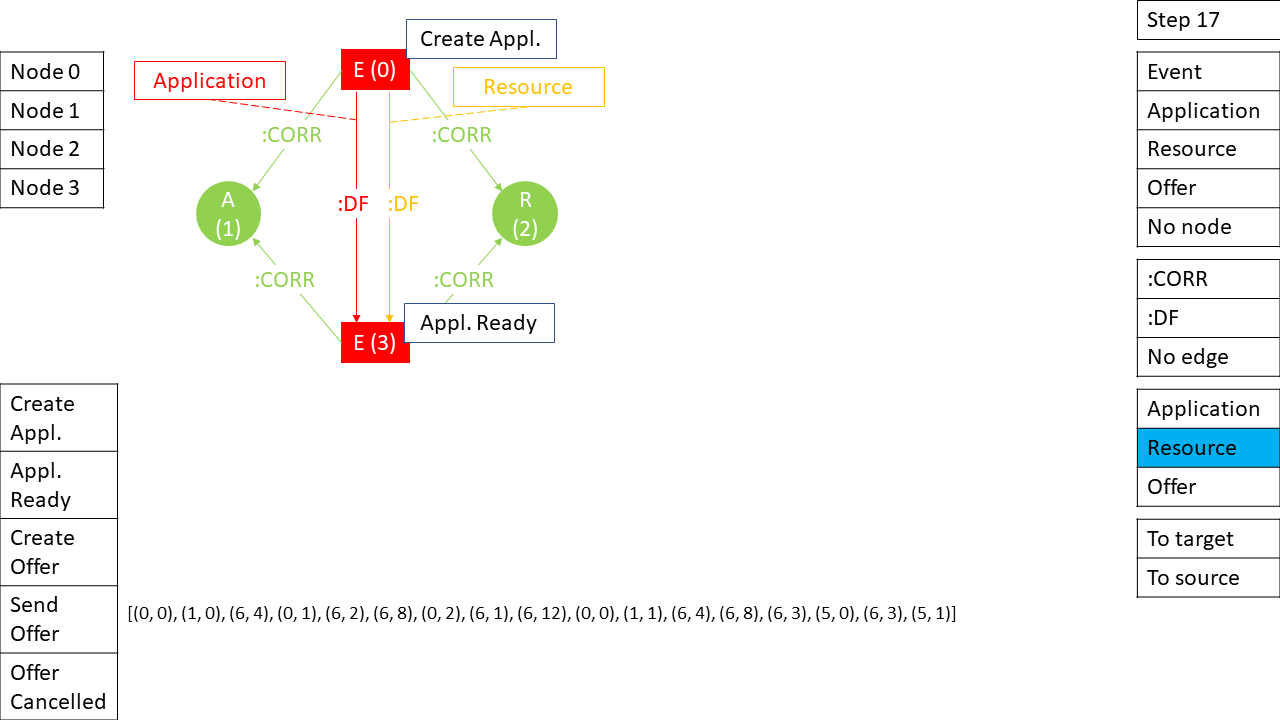
## Step 15 (5 – Correlation attribute, 0 – Application)



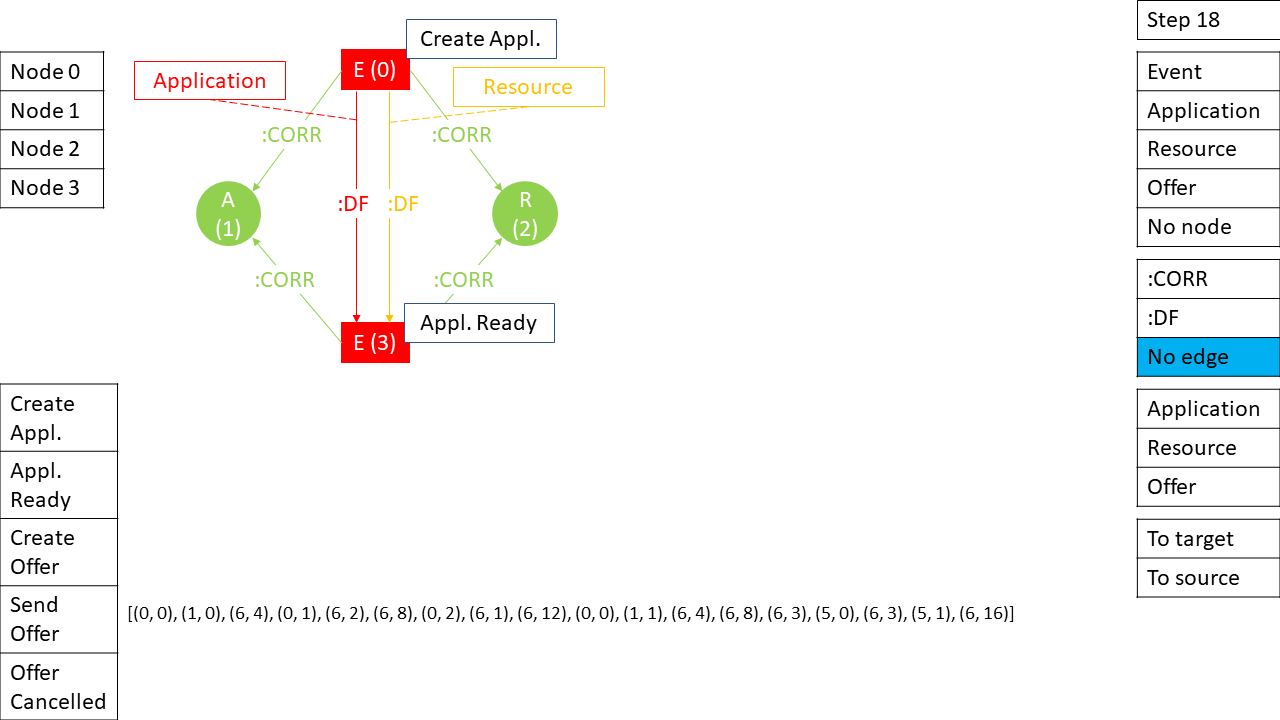
## Step 16 (6 – Joint Edge space, 3 – :DF)



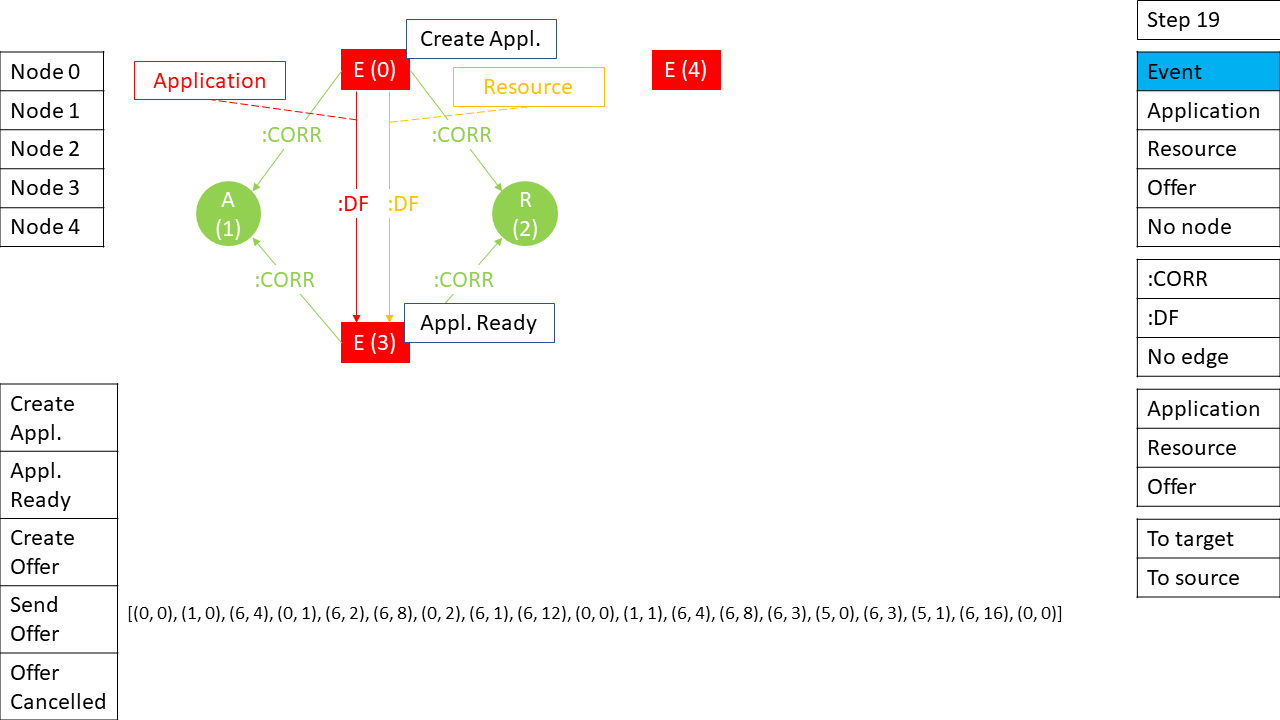
## Step 17 (5 – Correlation attribute, 1 – Resource)



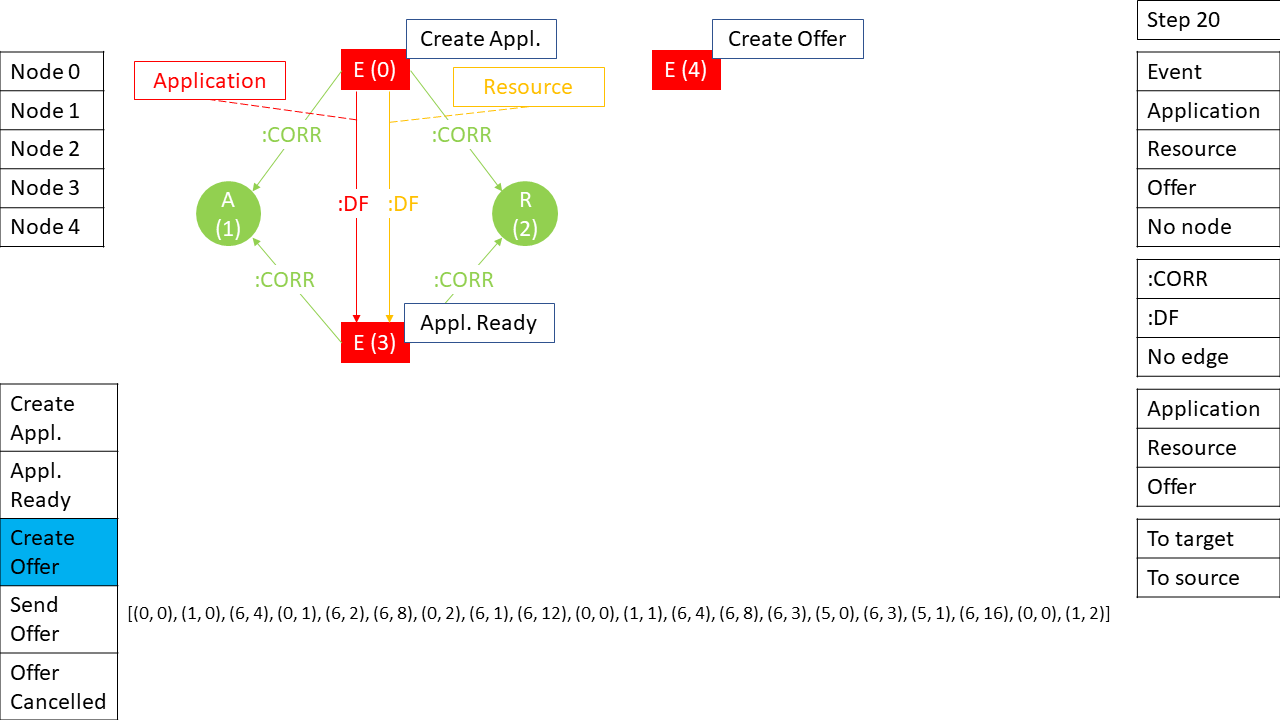
## Step 18 (6 – Joint edge space, 0 – No edge)



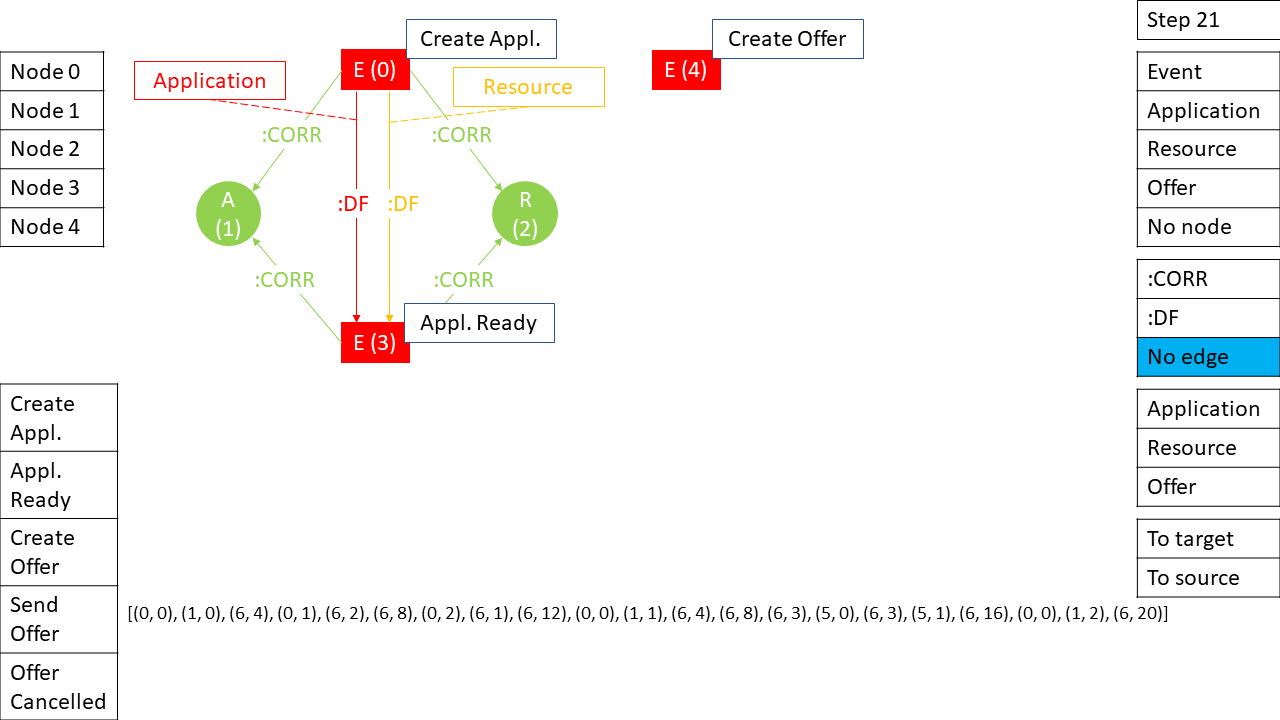
## Step 19 (0 – Node, 0 – Event)



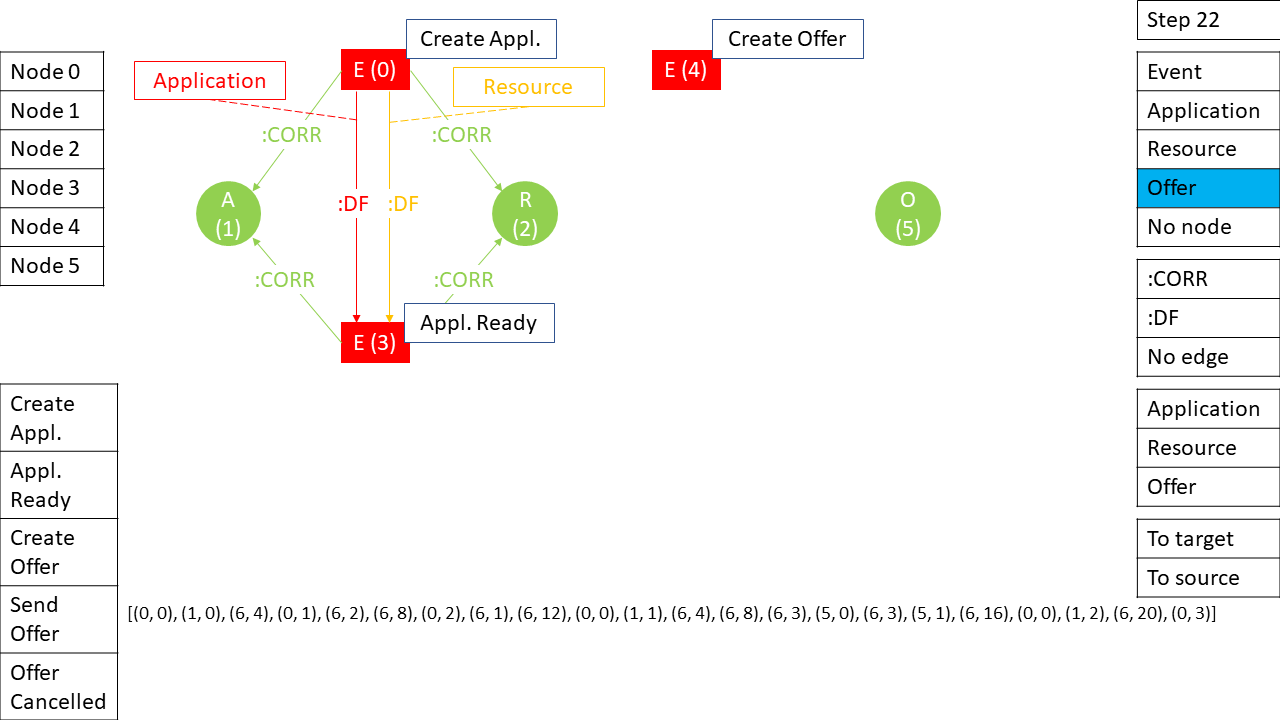
## Step 20 (1 – Activity attribute, 2 – Create Offer)



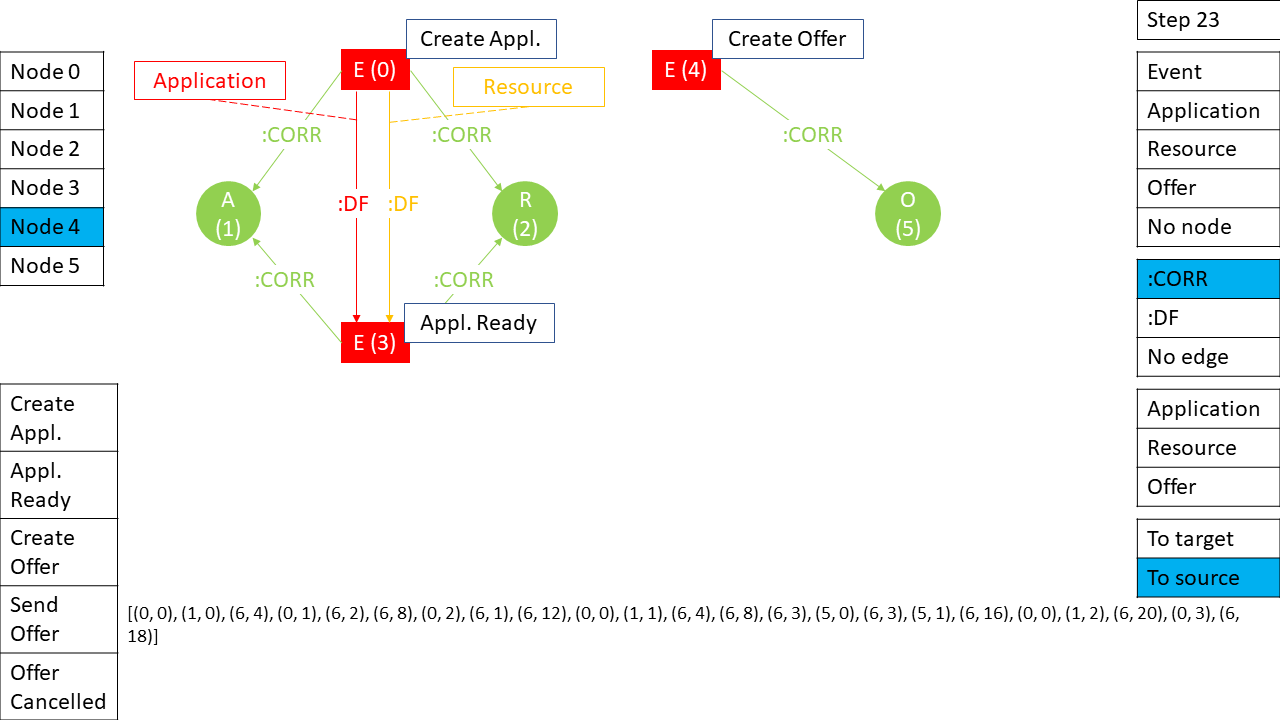
## Step 21 (6 – Joint edge space, 20 – No edge)



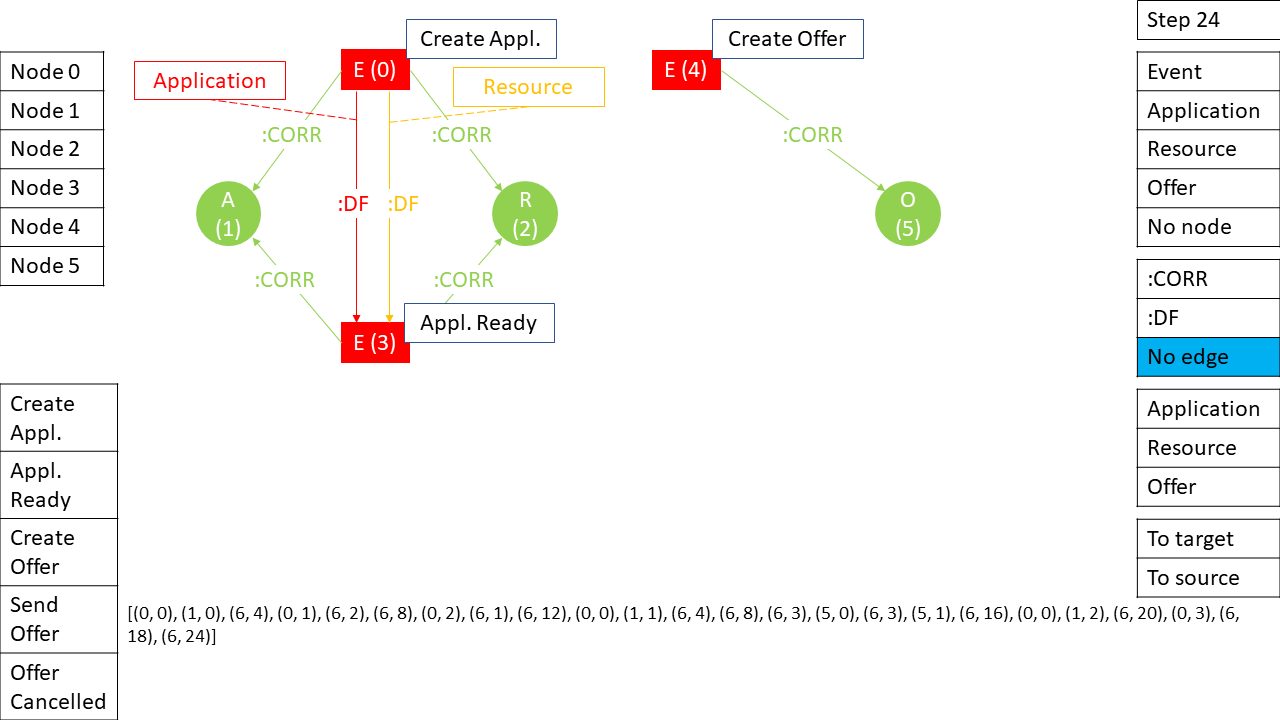
## Step 22 (0 – Node, 3 – Offer)



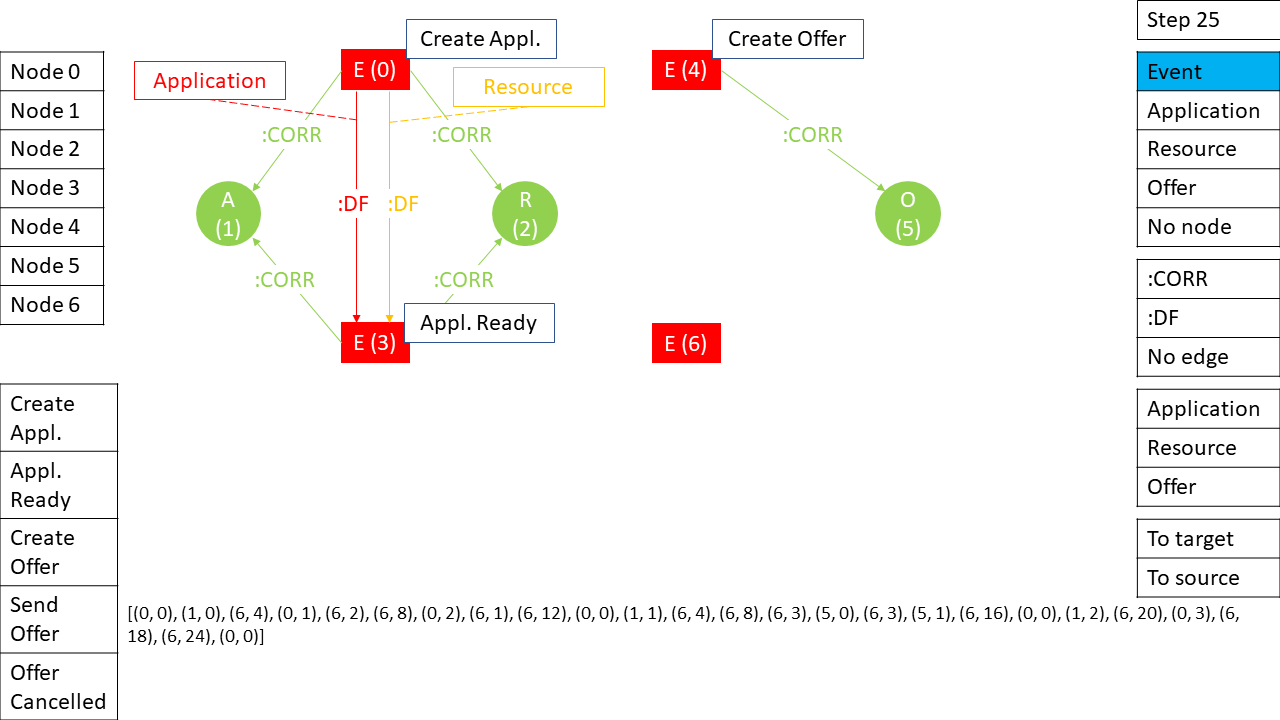
## Step 23 (6 – Joint edge space, 18 – :CORR)



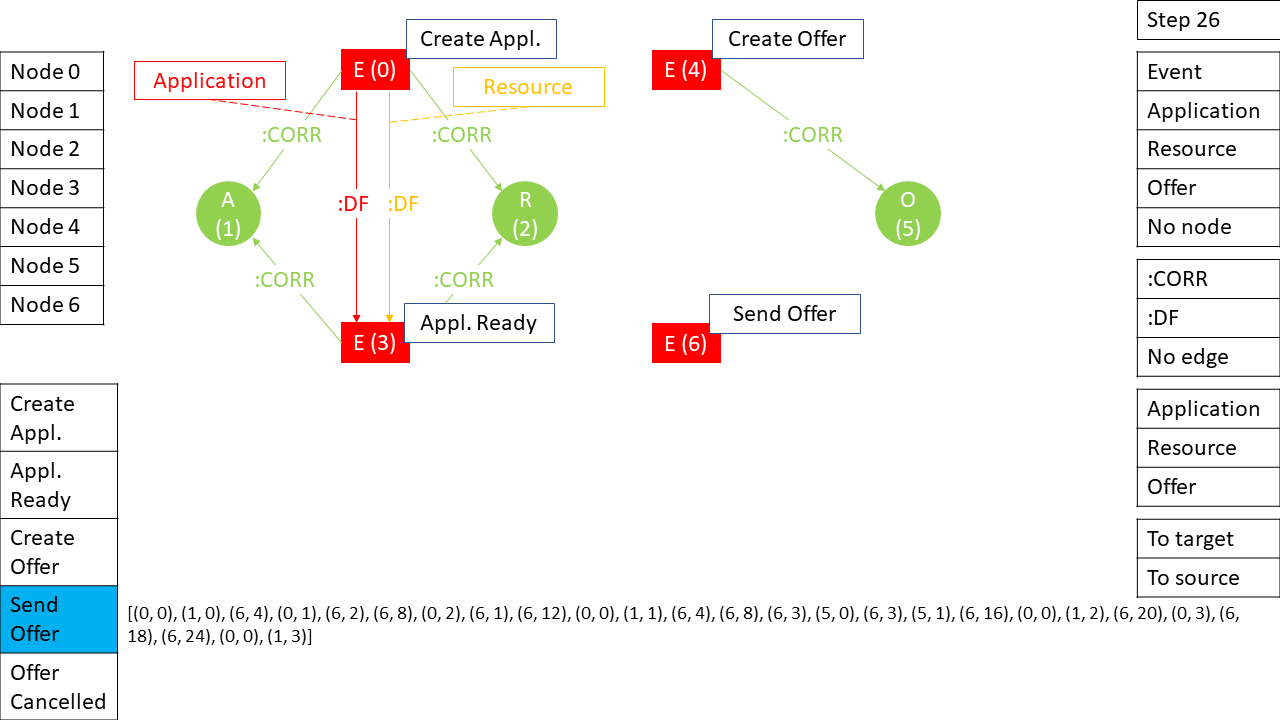
## Step 24 (6 – Joint edge space, 24 – No edge)



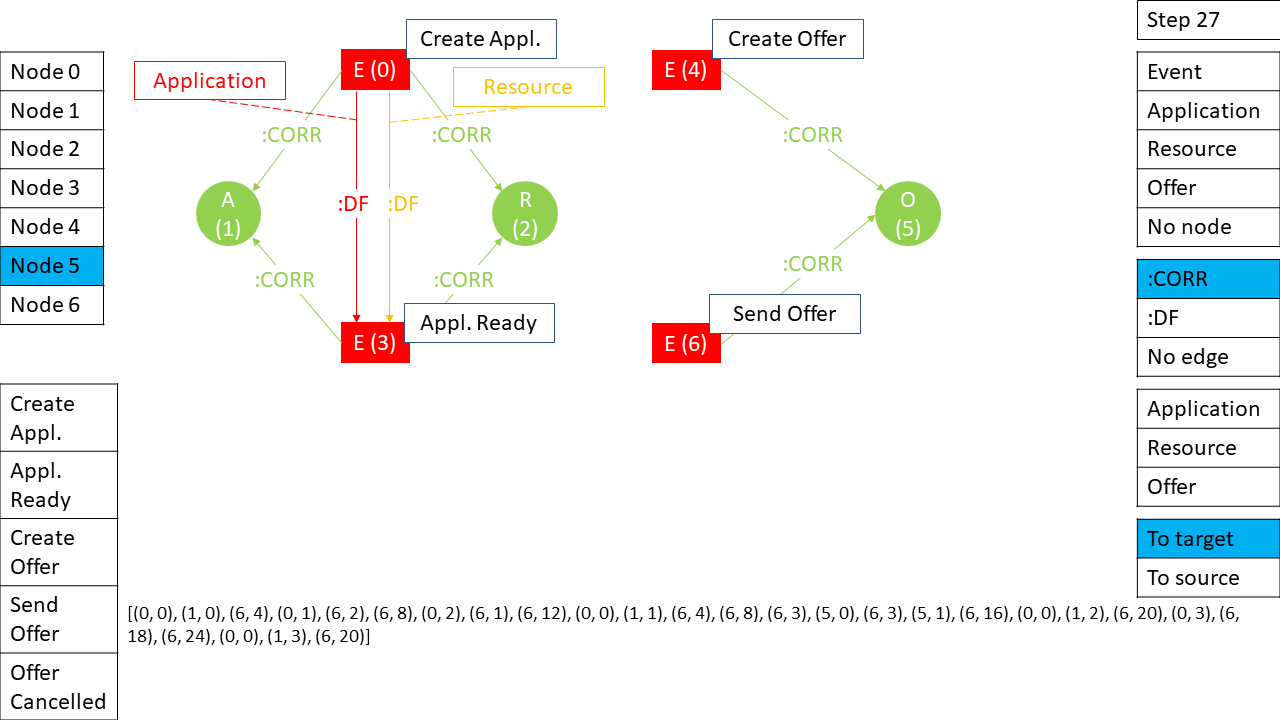
## Step 25 (0 – Node, 0 – Event)



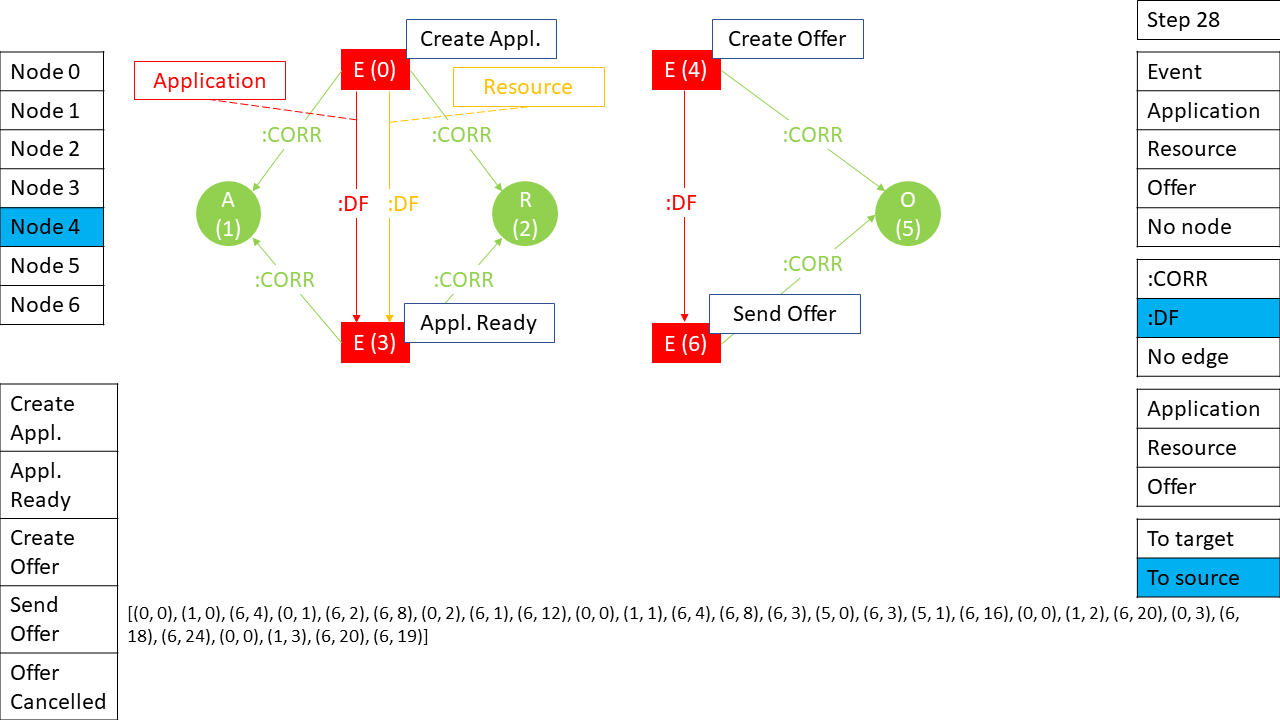
## Step 26 (1 – Activity attribute, 3 – Send Offer)



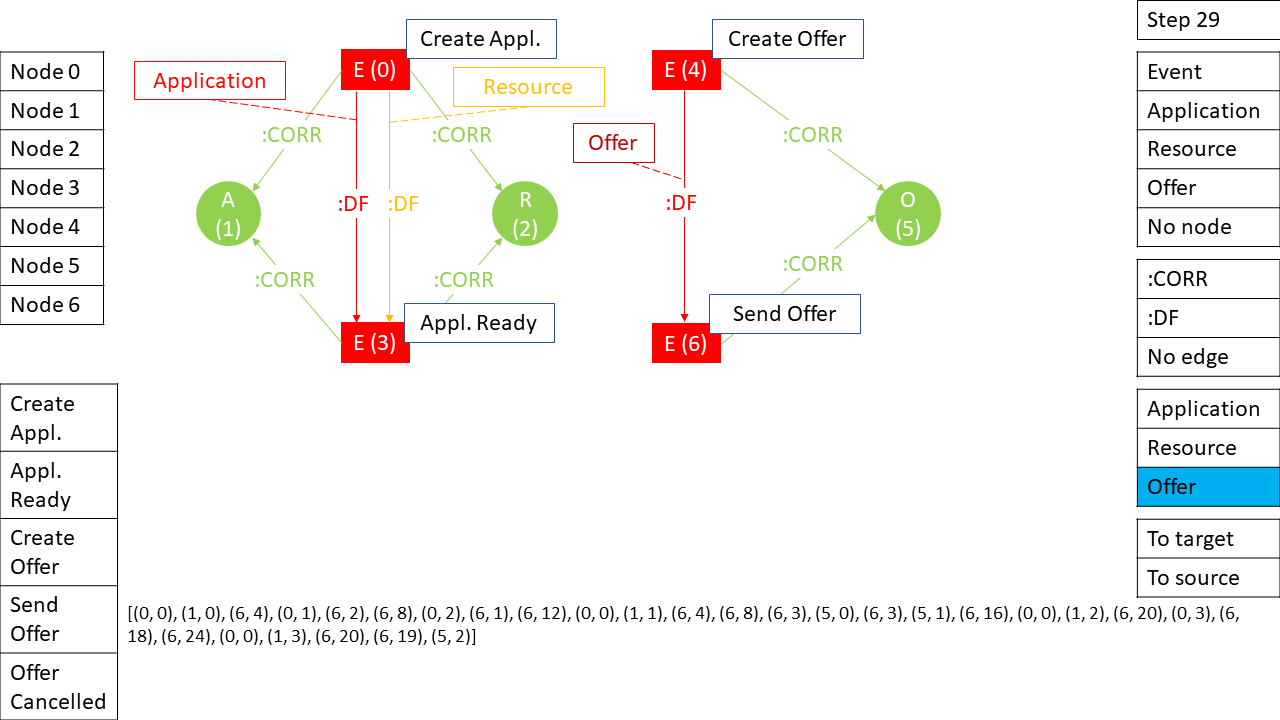
## Step 27 (6 – Joint edge space, 20 – :CORR)



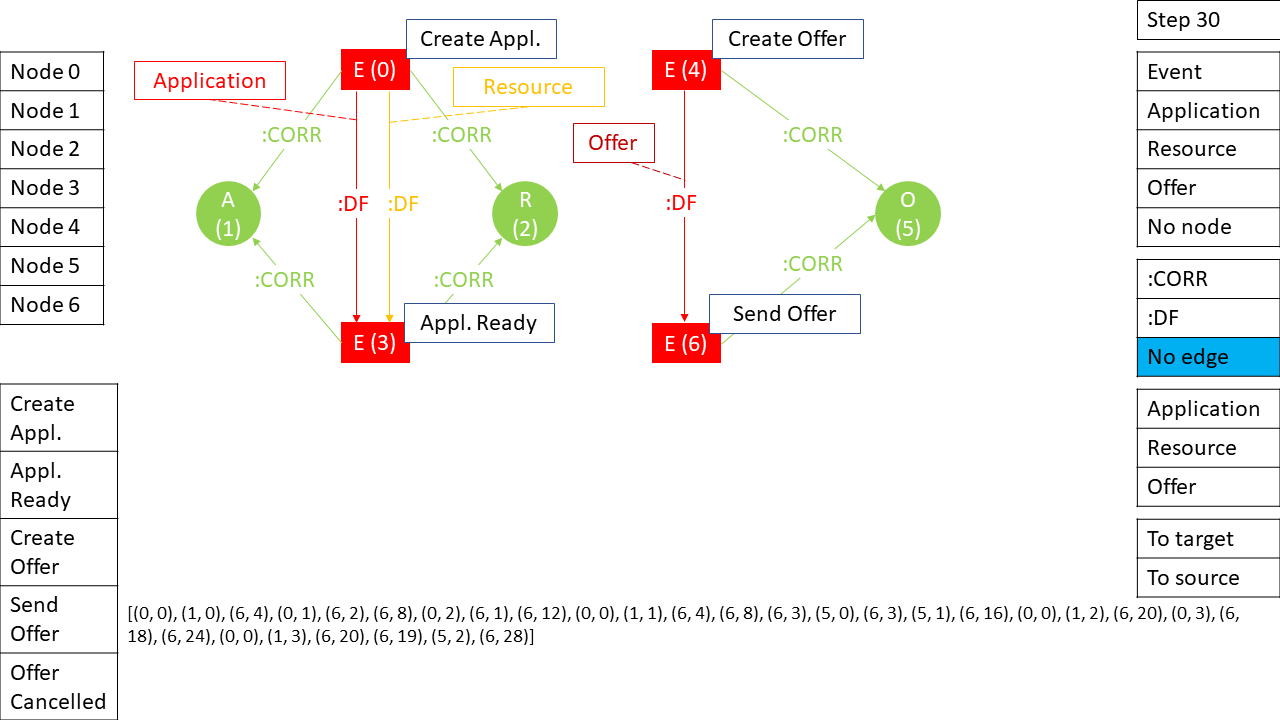
## Step 28 (6 – Joint edge space, 19 – :DF)



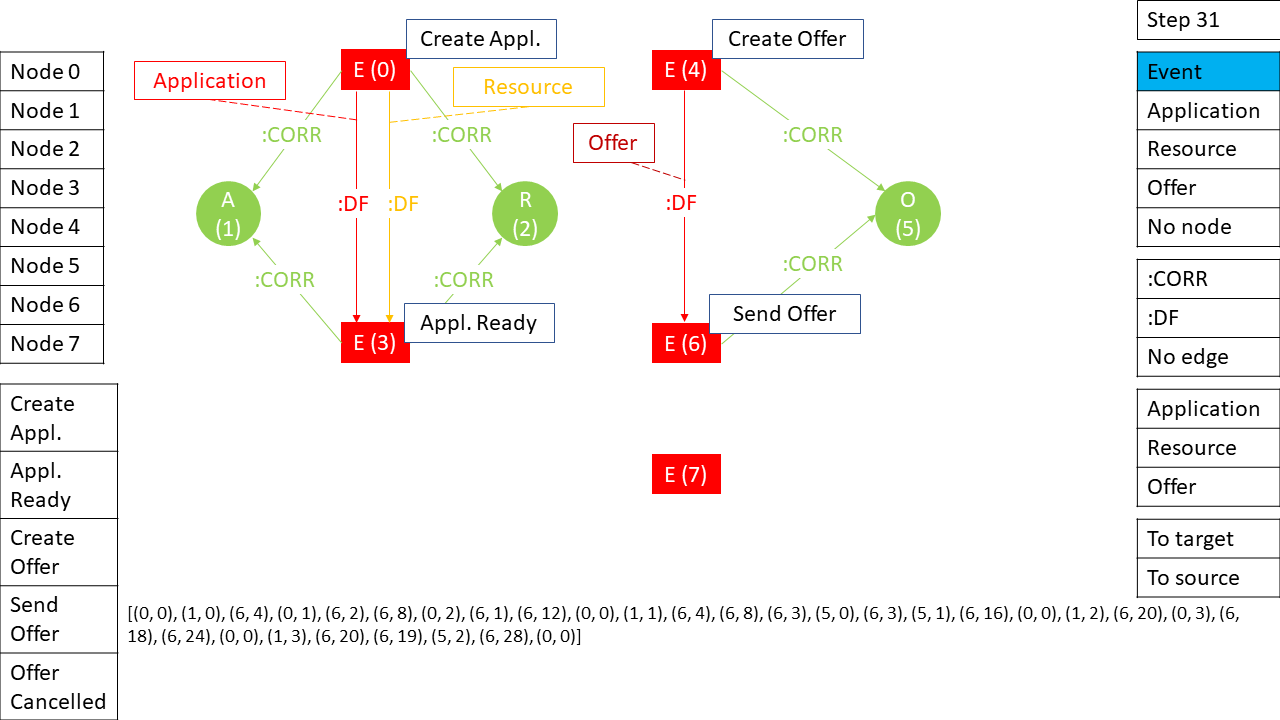
## Step 29 (5 – Correlation attribute, 2 – Offer)



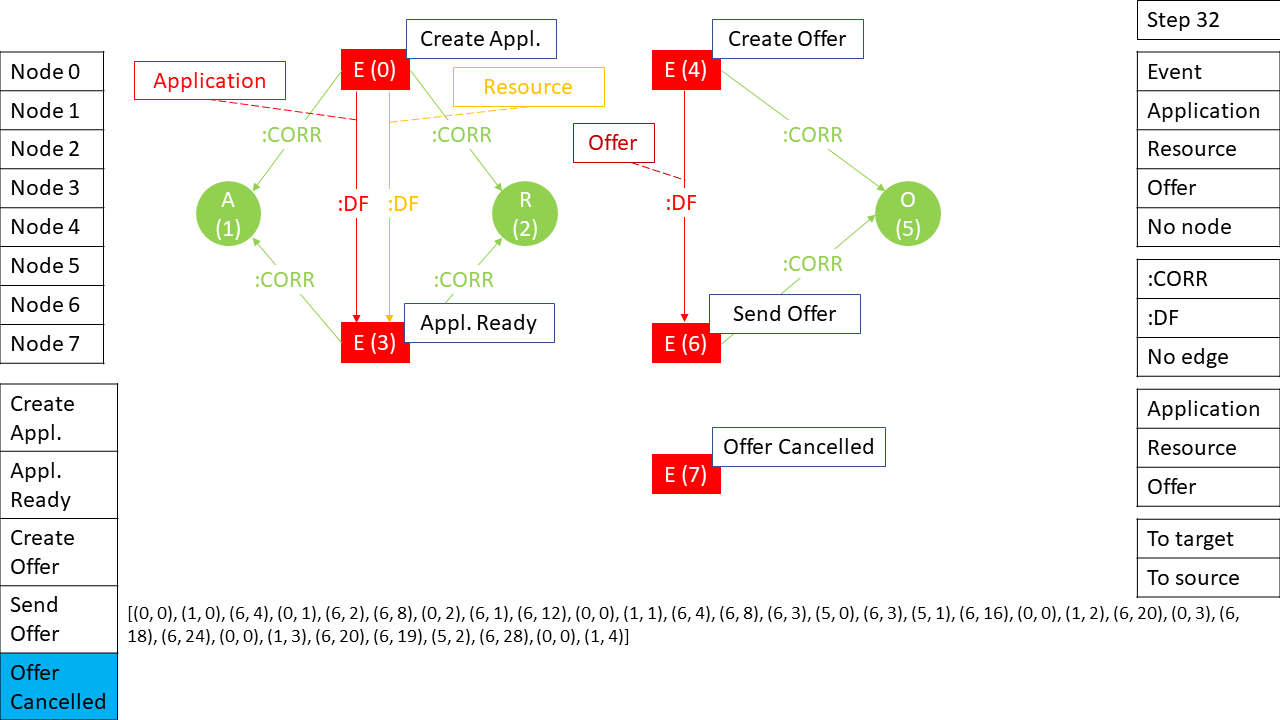
## Step 30 (6 – Joint edge space, 28 – No edge)



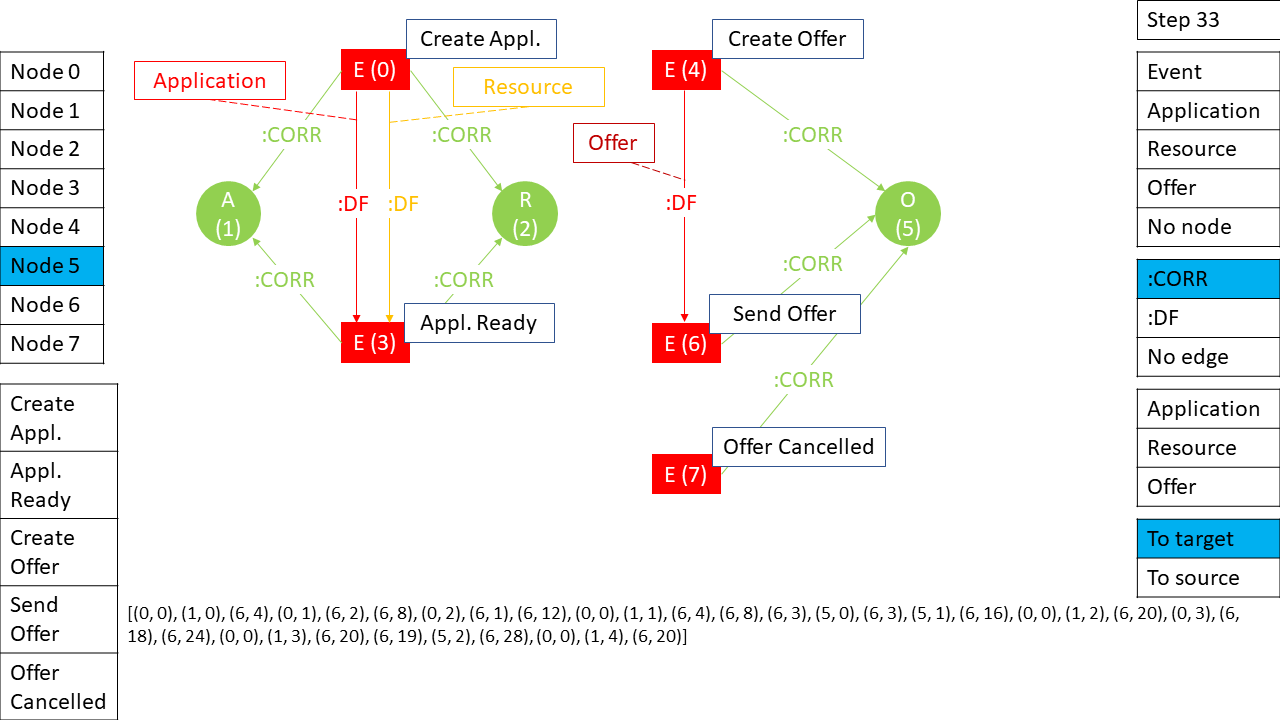
## Step 31 (0 – Node, 0 – Event)



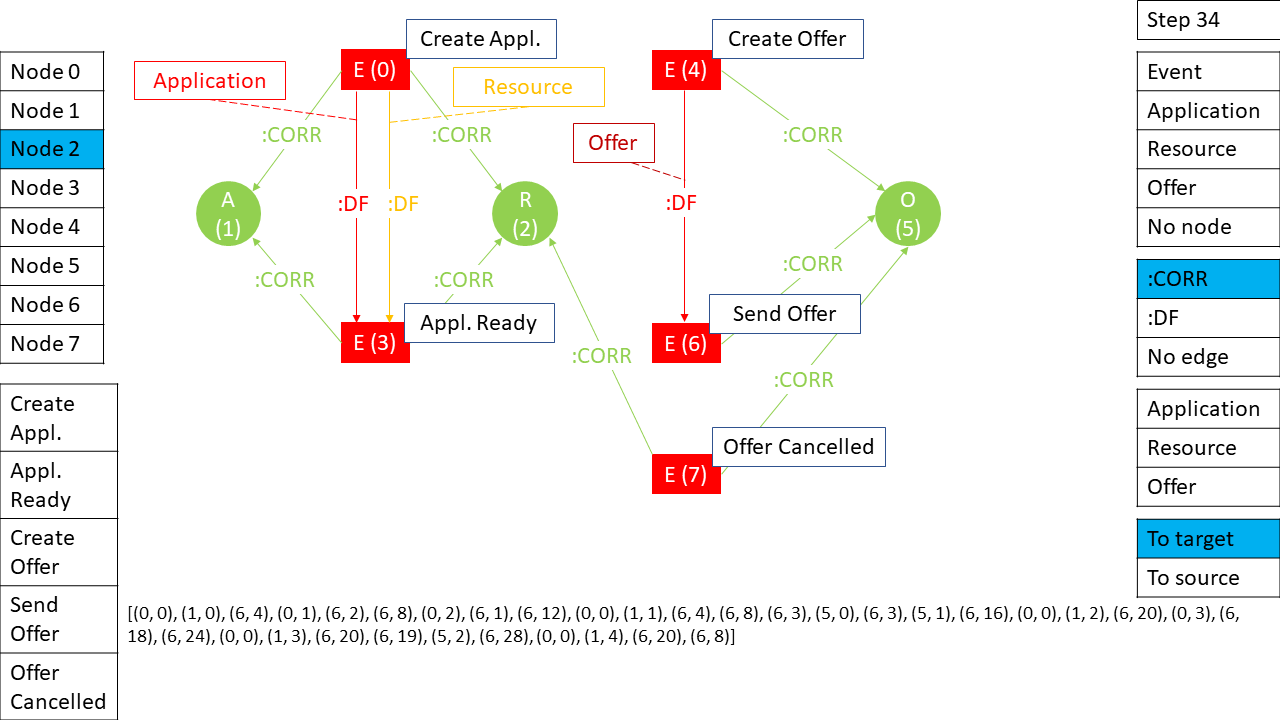
## Step 32 (1 – Activity attribute, 4 – Offer Cancelled)



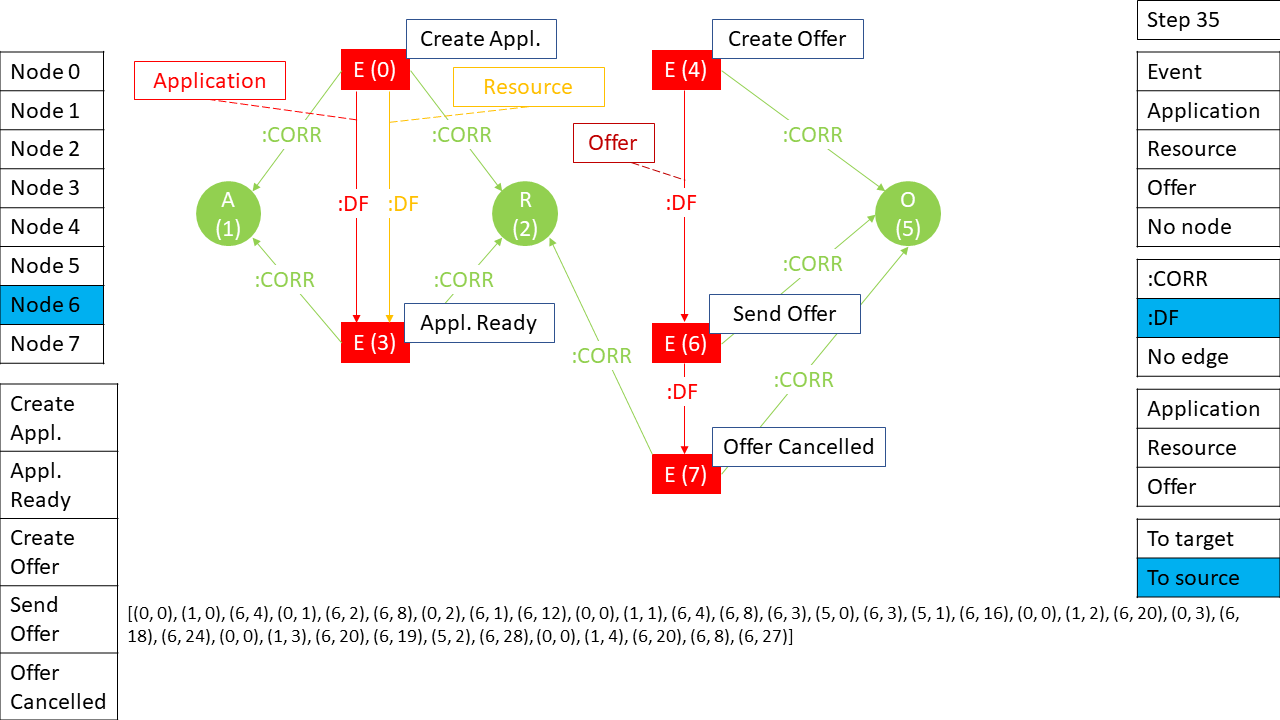
## Step 33 (6 – Joint edge space, 20 – :CORR)



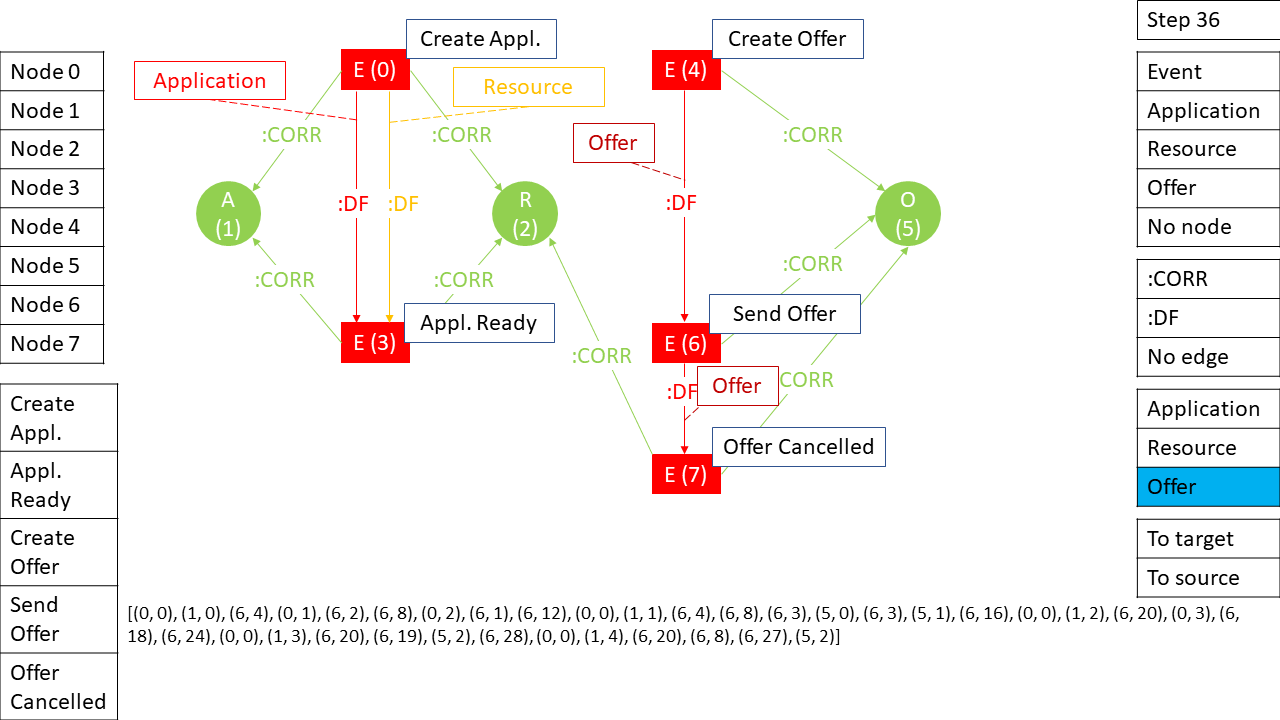
## Step 34 (6 – Joint edge space, 8 – :CORR)



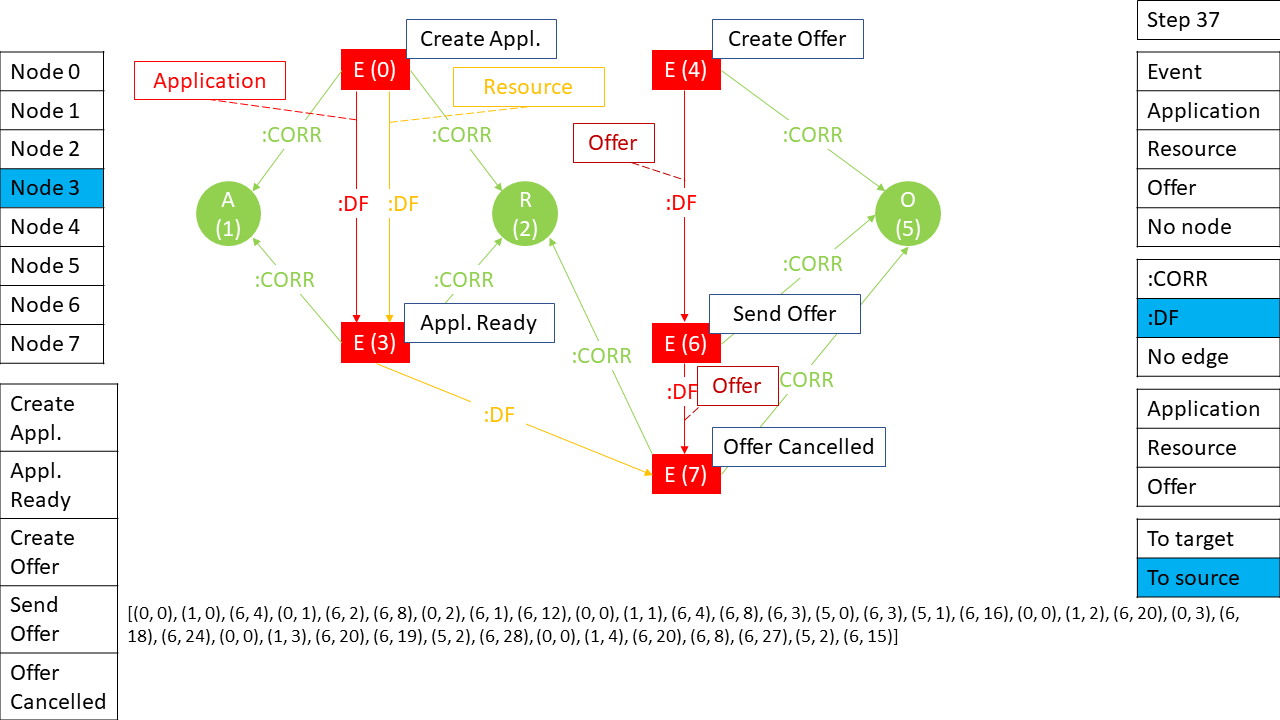
## Step 35 (6 – Joint edge space, 27 – :DF)



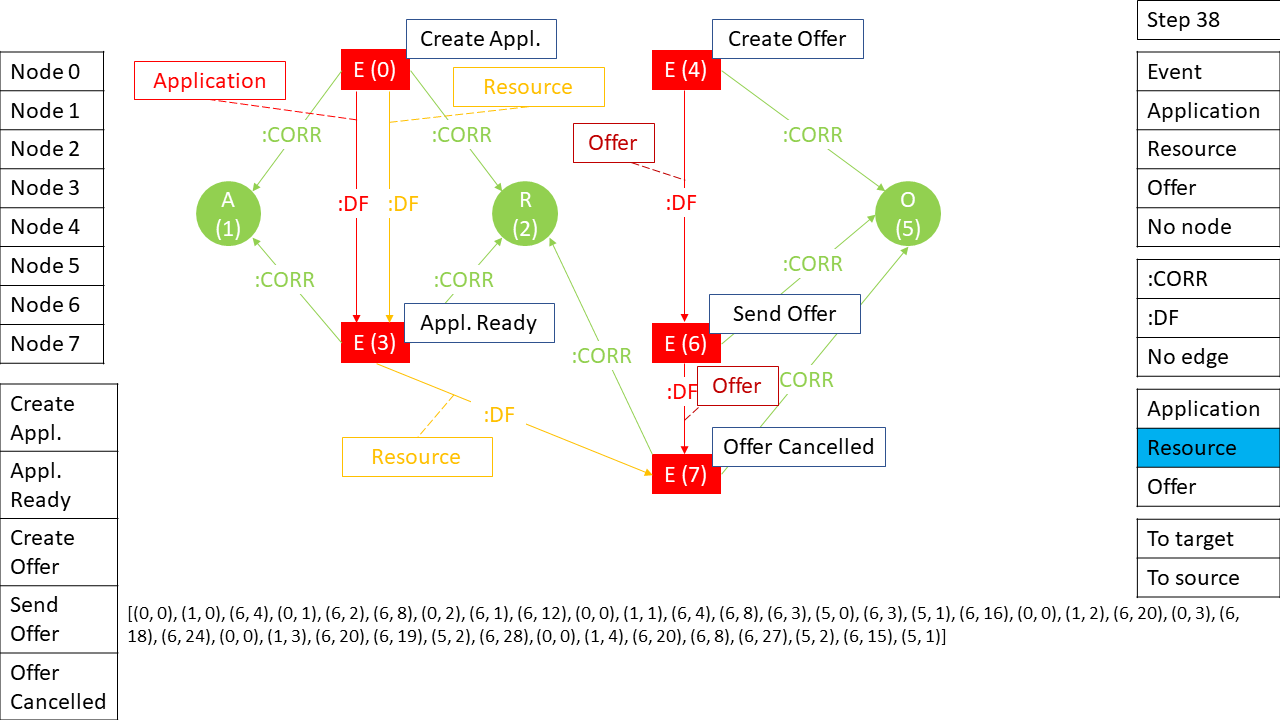
## Step 36 (5 – Correlation attribute, 2 – Offer)



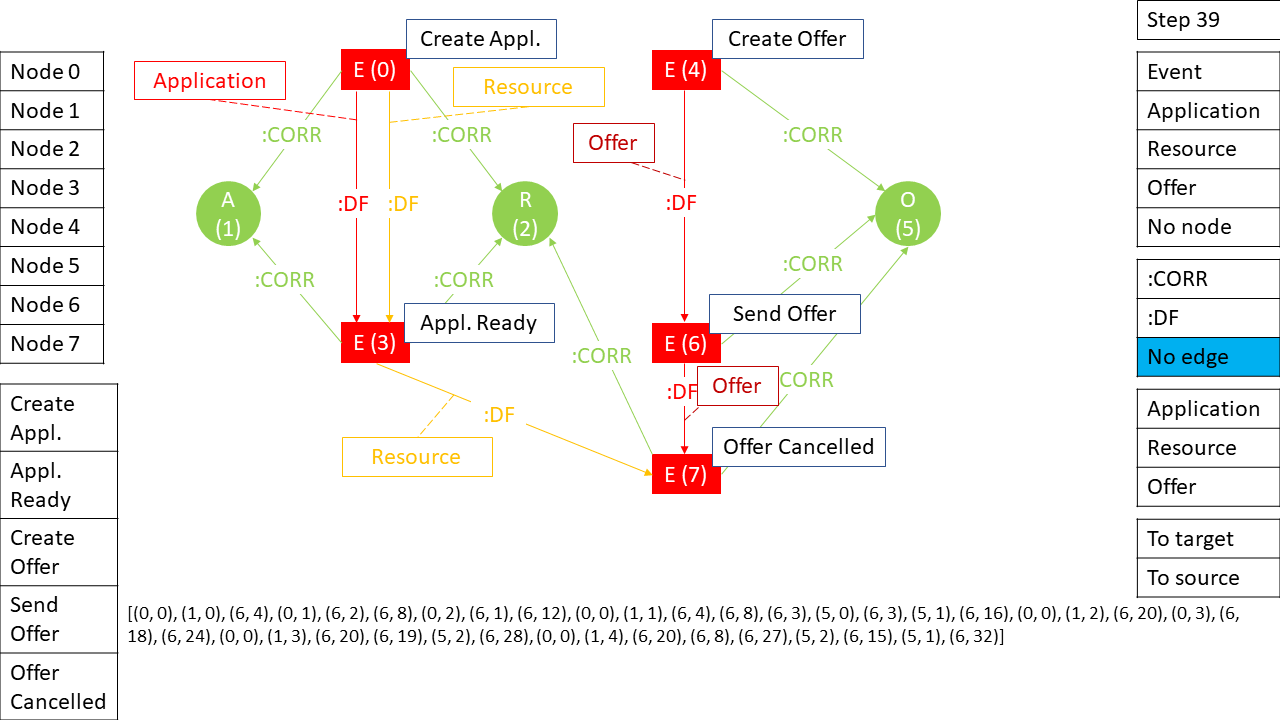
## Step 37 (6 – Joint edge space, 15 – :DF)



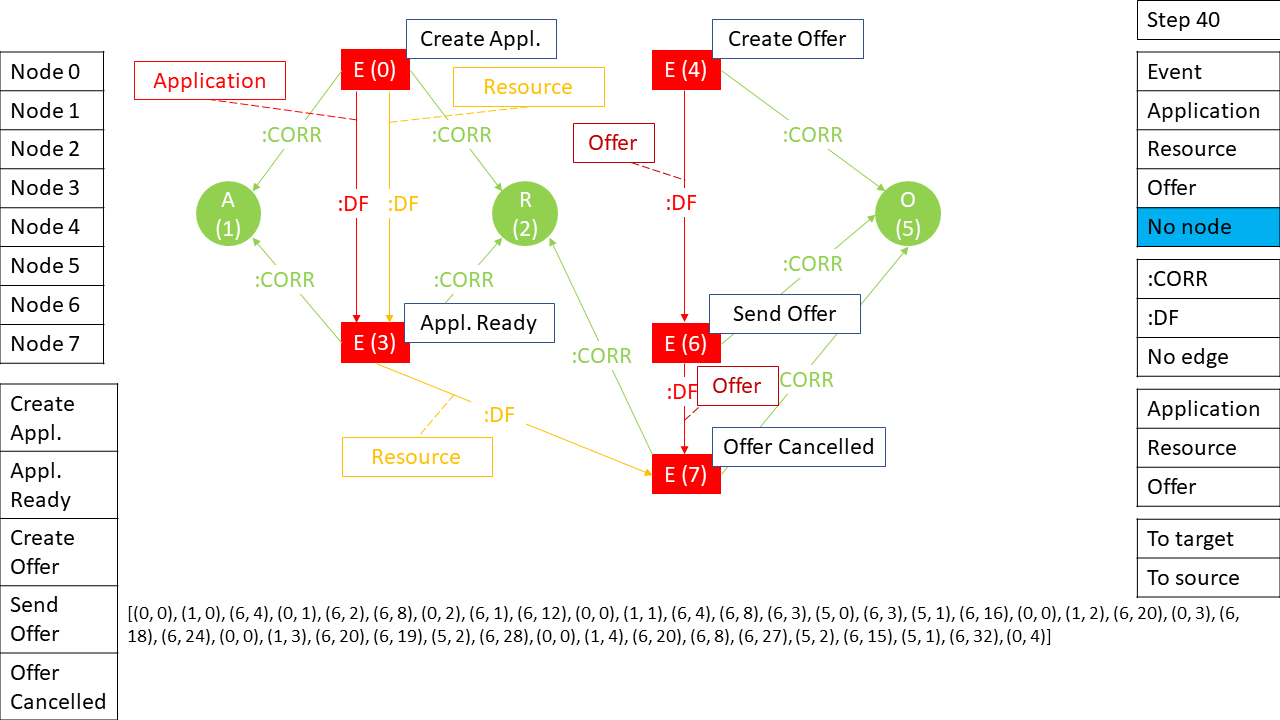
## Step 38 (5 – Correlation attribute, 1 – Resource)



## Step 39 (6 – Joint edge space, 32 – No edge)



## Step 40 (0 – Node, 4 – No node)



The aforementioned algorithm has some challenges. Due to the growing size of the graph there are scalability issues and representation learning difficulties:

* Scalability: Memory consumption of the graph
* Representation: the Node list is increasing so the cardinality of Joint edge space is also increasing making more challenging to represent that joint probability distribution.

The need of pruning the graph by time is expected. The solution for that is the matter of future research. Some pruning candidates:

* Keeping a moving window containing fixed number of nodes and corresponding edges.
* Dropping Event nodes and corresponding edges which are older than a time threshold.
* Dropping Entities and edges which are considered as *closed/completed* (see page 297 in (Fahland, Process Mining over Multiple Behavioral Dimensions with Event Knowledge Graphs, 2022) to understand terminology) at a given point in time.
  + Event nodes which end-up having no further correlation are also deleted.
* Solutions of graph stream mining can be adopted (András Benczúr, 2021)

# Code

Our code is at GitHub as a PyTorch example in a fork of the DGL project (Ketykó, 2022).

It follows (Mufei Li, 2022). (Mufei Li, 2022) implements (Yujia Li, 2018) for homogenous graph generation as visualized in Figure 7.

Diagram

Description automatically generated

Figure 7 Homogenous graph generation sequence

Our example graph is encoded into a list at <https://github.com/ketyi/dgl/blob/671b1aae72d1c79a09e2eb07e187c9f7876770d7/examples/pytorch/r-dgmg/heterograph.py#L24> for details of encoding see the previous example of steps.

The metagraph is defined at <https://github.com/ketyi/dgl/blob/671b1aae72d1c79a09e2eb07e187c9f7876770d7/examples/pytorch/r-dgmg/configure.py#L27>

The metagraph object is built by instantiation the Metagraph class (<https://github.com/ketyi/dgl/blob/671b1aae72d1c79a09e2eb07e187c9f7876770d7/examples/pytorch/r-dgmg/model.py#L10>).

The implementation needs to be continued with the development of message passing which incorporates two additional components:

1. The df-edge’s Correlation attribute
2. The Event node’s Activity attribute

There is no prior art for that yet. Basically, (Yujia Li, 2018) and (Mufei Li, 2022) already differ a bit on message passing. Furthermore, we have to generalize it further to a relational approach as depicted in (Michael Schlichtkrull, 2018). The basic implementation idea of this relational approach is described by DGL in (Message Passing on Heterogeneous Graph, 2022). Eq. (1) in (Yujia Li, 2018) defines message passing as:

where is the source node vector, is the target node vector, and is the edge attribute vector. However, the message function by (Mufei Li, 2022) at <https://github.com/ketyi/dgl/blob/b2a0f8174e17ba475b9d7232dbec77aa4a4860e7/examples/pytorch/r-dgmg/model.py#L109> only passes . Anyways, this has to be refactored to introduce an extra level of aggregation hierarchy due to different relation types (Message Passing on Heterogeneous Graph, 2022).

# Bibliography

András Benczúr, F. B. (2021). Tutorial on graph stream analytics. *DEBS '21.*

*Deep Graph Library*. (2022). Retrieved from https://www.dgl.ai/

Fahland, D. (2022). *Event Graph of BPI Challenge 2017*. Retrieved from https://doi.org/10.4121/14169584

Fahland, D. (2022). Process Mining over Multiple Behavioral Dimensions with Event Knowledge Graphs. In *Process Mining Handbook.*

Ketykó, I. (2022). *R-DGMG*. Retrieved from https://github.com/ketyi/dgl/tree/master/examples/pytorch/r-dgmg

Leskovec, J. (2022). *Stanford CS224W: Machine Learning with Graphs*. Retrieved from https://www.youtube.com/playlist?list=PLoROMvodv4rPLKxIpqhjhPgdQy7imNkDn

*Message Passing on Heterogeneous Graph*. (2022). Retrieved from https://docs.dgl.ai/en/latest/guide/message-heterograph.html

Michael Schlichtkrull, T. N. (2018). Modeling Relational Data with Graph Convolutional Networks. *ESWC.*

Mufei Li, L. Y. (2022). *Generative Models of Graphs - DGL Tutorial*. Retrieved from https://docs.dgl.ai/en/latest/tutorials/models/3\_generative\_model/5\_dgmg.html

*PyTorch Geometric*. (2022). Retrieved from https://www.pyg.org/

Yujia Li, O. V. (2018). Learning Deep Generative Models of Graphs. *ICML.*