Simple Protocol

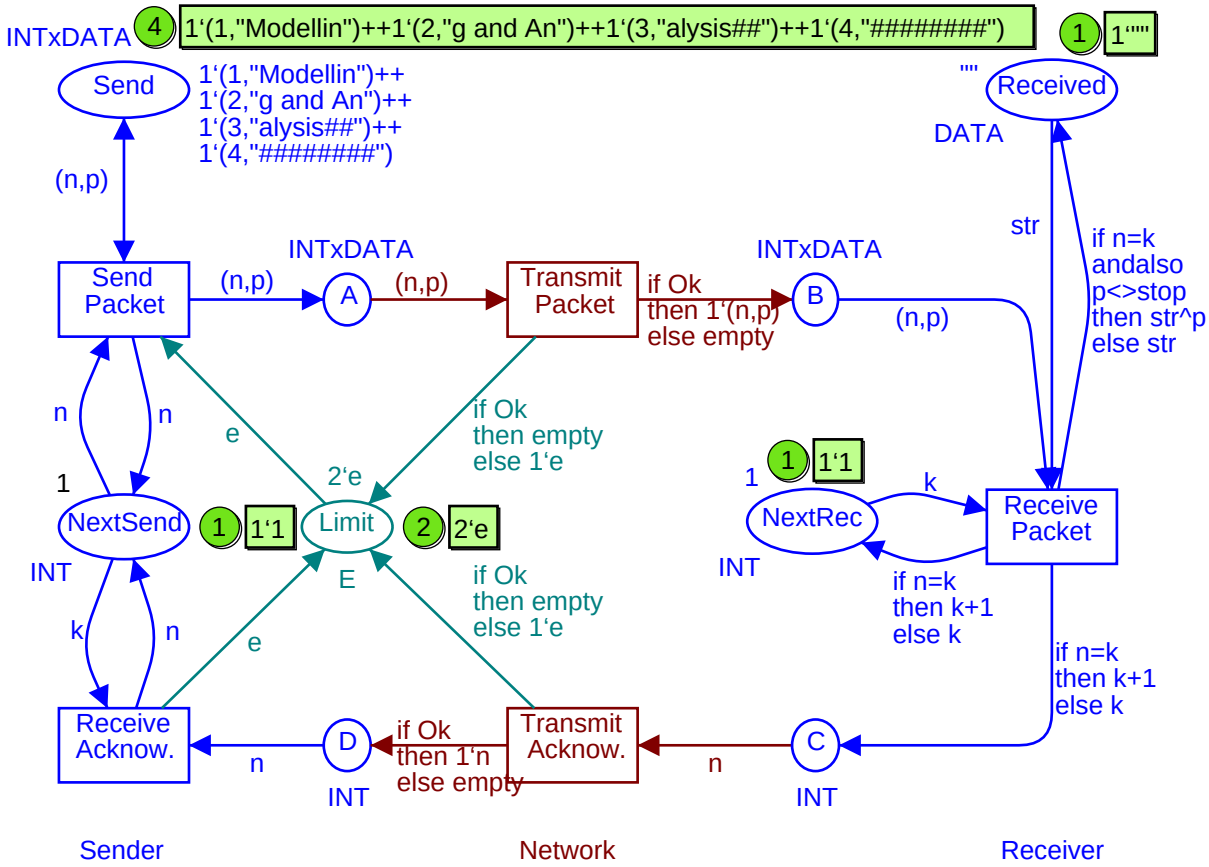
Abstract
This is a small toy example which is well-suited as an introduction to occurrence graphs. The analysis of the occurrence graph is described in great detail.

The CPN model describes a simple protocol by which a sender can transfer a number of packets to a receiver. The model is identical to the “Simple Protocol” presented in “Introductory Examples”(which we recommend to study before this example).

Developed and Maintained by:
Kurt Jensen, Aarhus University, Denmark (kjensen@daimi.au.dk).

CPN Model

In this example we study the O-graph for the simple protocol. To obtain a finite and reasonably small O-graph we make three modifications of the CP-net. First we reduce the number of packets from eight to four. Secondly, we introduce a new place to *Limit* the number of packets/acknowledgments which simultaneously can be at the network. Finally, we simplify the decision mechanism for transferring/loosing packets and acknowledgments. For O-graphs it does not make sense that packets are lost/transmitted with a certain probability. Hence, we replace the *Ok* function with a boolean variable *Ok*.

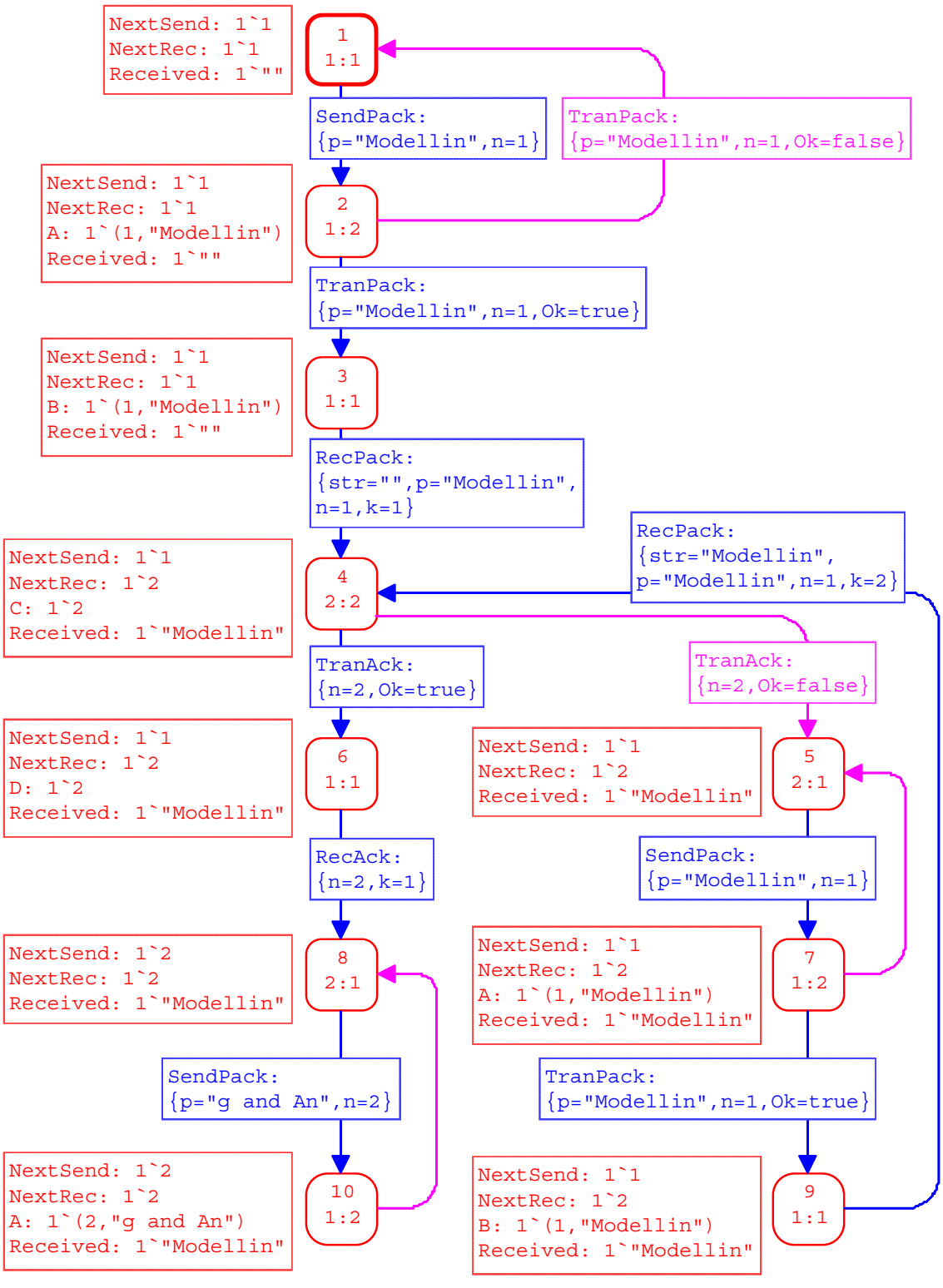



```
color INT = int;
color DATA = string;
color INTxDATA= product INT*DATA;
color E = with e;
var n,k: INT;
var p, str: DATA;
val stop = ”#########”;

color BOOL = bool;
var Ok: BOOL;
```

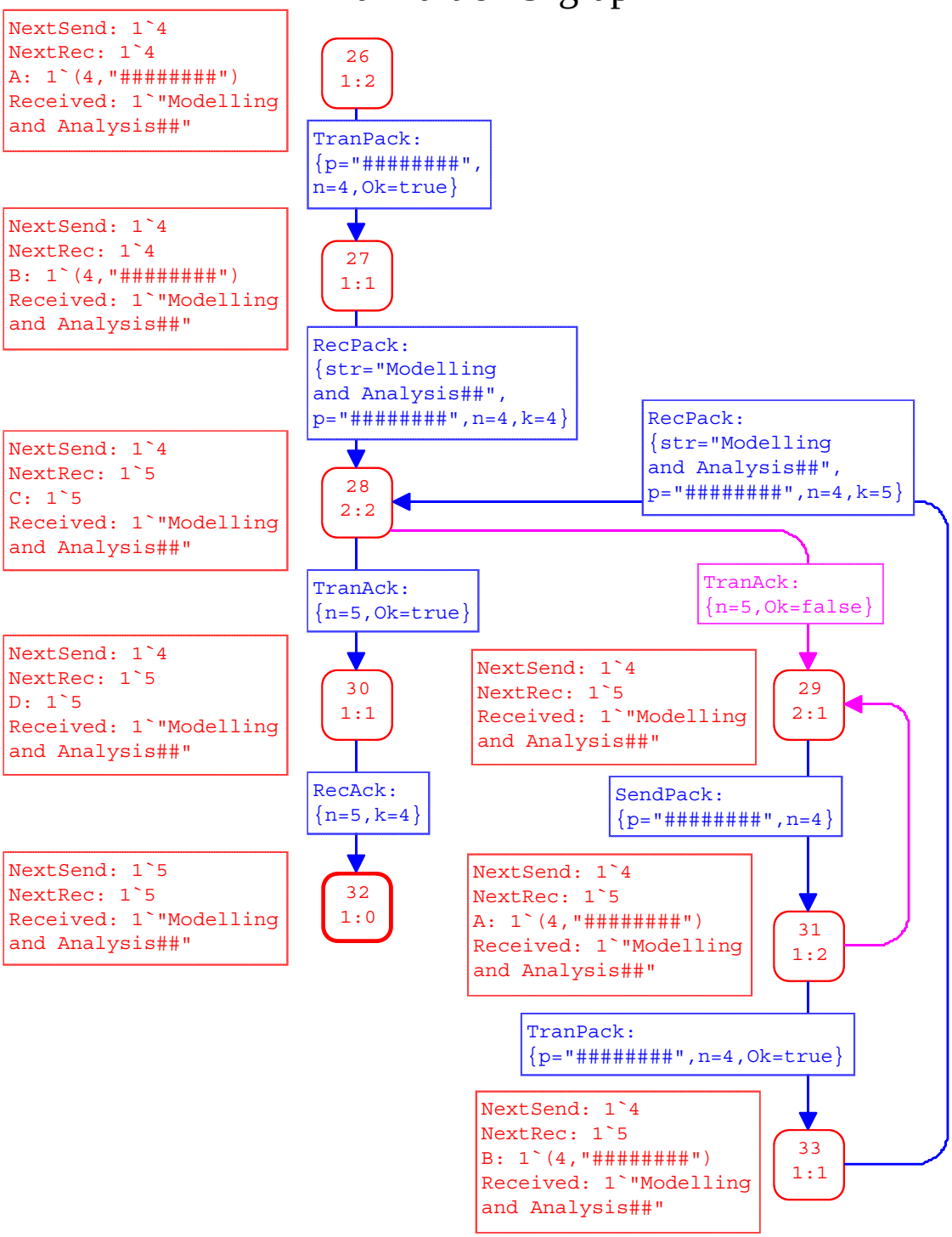
Having made the modifications described above, we are ready to construct O-graphs. Let us start with the situation where the initial marking of *Limit* is 1`e. This means that the network (i.e., the place *A*, *B*, *C* and *D*) contains at most one packet/acknowledgment at a time. Hence overtaking is impossible. The O–

Initial Part of O-graph



graph has 33 nodes and 44 arcs. The initial and final part of it is drawn below. The current version of CPN Tools does not include facilities for drawing O-graphs. From the O-graph we can see that the O-graph has a regular structure, in the sense that some patterns are repeated. The subgraph of nodes {4, 5, 7 and 9} has the same ”form” as the subgraph of nodes {28. 29. 31. 33}. The only difference is that the latter is “three packets ahead” of the former. If we construct the middle part of the occurrence graph, we will find two additional copies of the pattern.

Final Part of O-graph



Next let us investigate the more complex situation in which overtaking is possible. To do this, we construct an O-graph for the situation where the initial marking of *Limit* is 2`e. The new O-graph is considerably larger than the first one. The standard report looks as shown below.

From the statistics it can be seen that the O-graph has 428 nodes and 1130 arcs. It can also be seen that there are fewer strongly connected components than O-graph nodes. This means that the system has at least one non-trivial strongly connected component, and hence an infinite occurrence sequences exists. In other words, we cannot be sure that the simple protocol terminates – to achieve termination one usually limits the number of retransmissions.

```
 Statistics
--------------------------------
  Occurrence Graph
    Nodes:  428
    Arcs:   1130
    Secs:   8
    Status: Full

  Scc Graph
    Nodes:  182
    Arcs:   673
    Secs:   1
```

The integer bounds are as expected.

```
 Boundedness Properties
--------------------------------
  Best Integers Bounds
             Upper      Lower
  A            2          0
  B            2          0
  C            2          0
  D            2          0
  Limit        2          0
  NextRec      1          1
  NextSend     1          1
  Received     1          1
  Send         4          4
```

Also the multi-set bounds are as expected. The places *A* and *B* may contain all four different packets, while places *C* and *D* may contain all four possible acknowledgments. Remember that an acknowledgment always specifies the

number of the next packet to be sent (hence we never has an acknowledgment with value 1). The two counters *NextSend* and *NextRec* can take all values between one and five. The place *Received* may contain four different values – corresponding to the situations where we have received the data from zero, one, two or three packets (packet number four contains "########" which we never copy to *Received*). Finally, the place *Send* has identical upper and lower multi-set bounds. This means that the marking never changes.

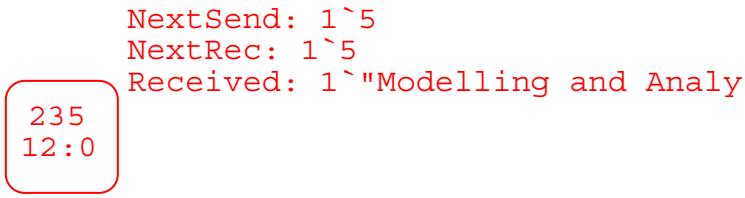
```
Best Upper Multi-set Bounds
A           2`(1,"Modellin")+
            2`(2,"g and An")+
            2`(3,"alysis##")+
            2`(4,"########")
B           2`(1,"Modellin")+
            2`(2,"g and An")+
            2`(3,"alysis##")+
            2`(4,"########")
C           2`2+ 2`3+ 2`4+ 2`5
D           2`2+ 2`3+ 2`4+ 2`5
Limit       2`e
NextRec     1`1+ 1`2+ 1`3+ 1`4+ 1`5
NextSend    1`1+ 1`2+ 1`3+ 1`4+ 1`5
Received    1`""+ 1`"Modellin"+
            1`"Modelling and An"+
        1`"Modelling and Analysis##"
Send        1`(1,"Modellin")+
            1`(2,"g and An")+
            1`(3,"alysis##")+
            1`(4,"########")+
Best Lower Multi-set Bounds
A           empty
B           empty
C           empty
D           empty
Limit       empty
NextRec     empty
NextSend    empty
Received    empty
Send        1`(1,"Modellin")+
            1`(2,"g and An")+
            1`(3,"alysis##")+
            1`(4,"########")+
```

The home and liveness properties are very interesting. They tell us that the system has exactly one dead marking M_{235} which also is a home marking:

```
 Home Properties
--------------------------------
  Home Markings:   [235]

 Liveness Properties
--------------------------------
  Dead Markings:   [235]
  Dead Transitions Instances: None
  Live Transitions Instances: None
```

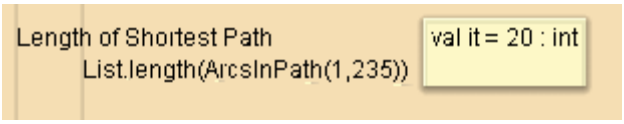
Marking M_{235} looks as shown below. It corresponds to the situation where all four packets has been successfully transmitted. The fact that M_{235} is dead tells us that the protocol is **partly correct** – if it terminates it terminates with the correct result. The fact that M_{235} is a home marking tells us that the protocol has the nice property that it never can reach a state from which it is impossible to terminate with the correct result.



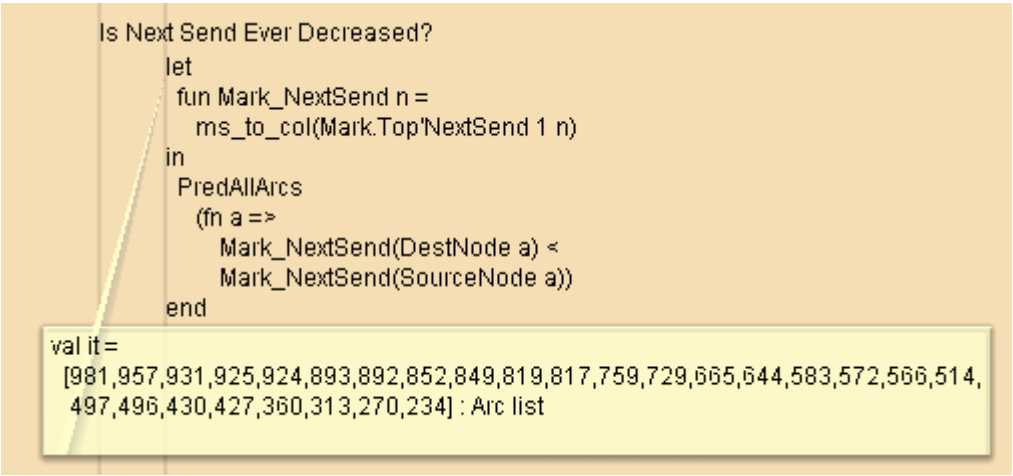
The fairness properties are as shown below.

```
 Fairness Properties
--------------------------------
  SendPack      Impartial
  TranPack      Impartial
  RecPack       No Fairness
  TranAck       No Fairness
  RecAck        No Fairness
```

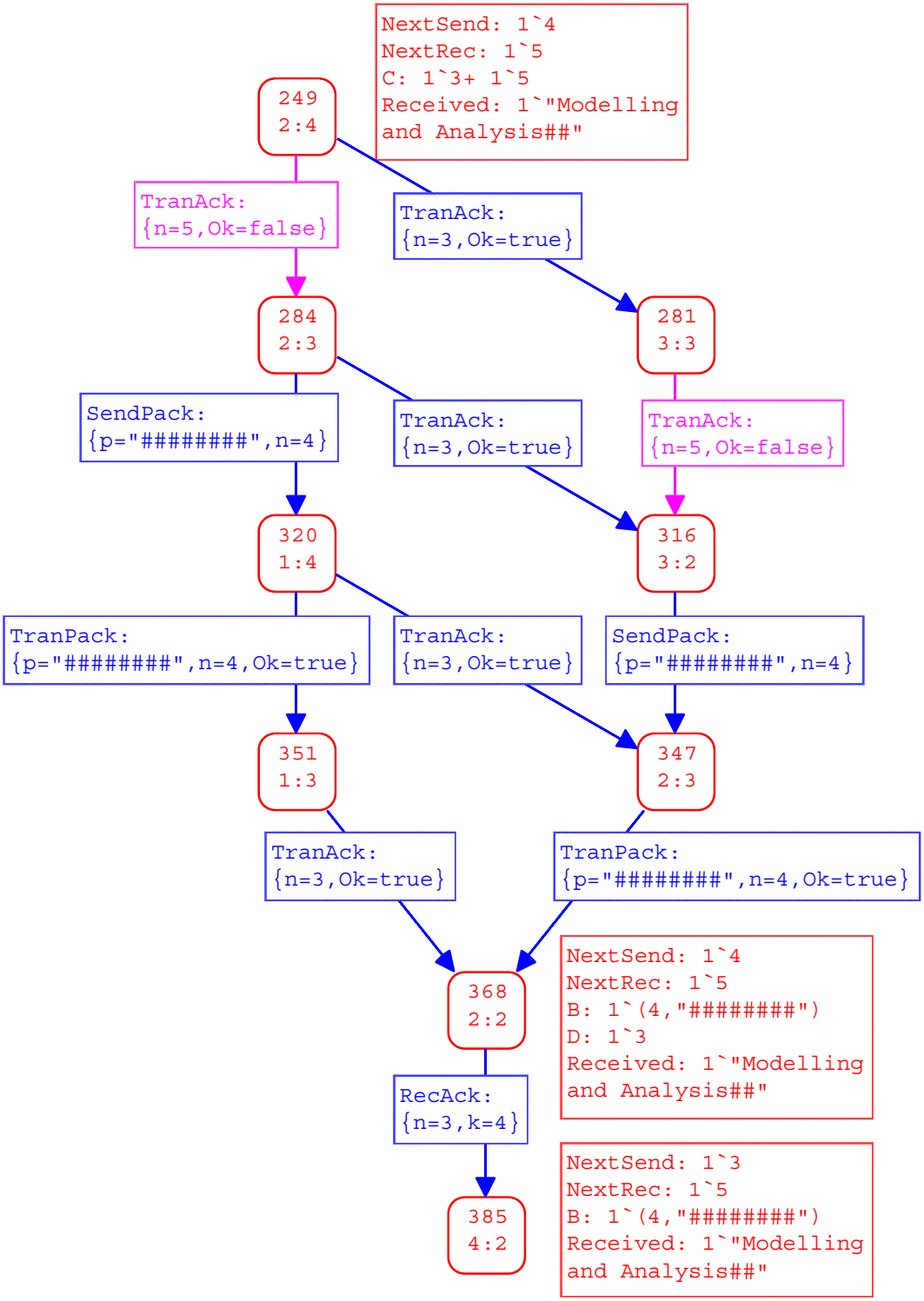
Above, we have seen that M_{235} is the desired final marking, and we have also seen that it can be reached from any reachable system state. Now let us investigate how fast it can be reached. To do this we ask the system to construct a path from the initial marking M_1 to M_{235}. By convention the system constructs a path with minimal length, and hence we see that at least 20 transitions must occur – in order to reach M_{235} from M_1. This is not surprising. We have four packets and to process a packet (plus the corresponding acknowledgment) we need one occurrence of each of the five transitions.



Next let us investigate the way in which we update the *NextSend* counter. One might expect that this counter always is increased (or left unchanged). However, the following query tells us that there are a number of occurrence graph arcs where *NextSend* actually is decreased. The result of the query is a list containing all those arcs where *NextSend* has a smaller value in the destination node than it has in the source node. The function *ms_to_col* maps a multi-set with one element into the element itself (e.g., 1`3 into 3).

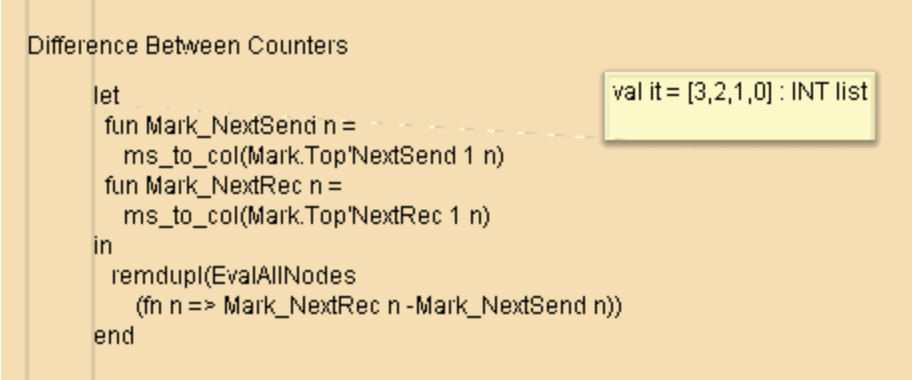


To investigate why *Next Send* is decreased, we examine the first arc in the result of the above query, i.e., arc number 981 (from node 368 to node 385). Recall that CPN Tools does not have facilities for drawing O-graphs, but standard queries can be used to examine all information associated with nodes and arcs in an O-graph. We also use standard queries to examine some of the nearest predecessors of node 368. After a few "backwards" steps we find marking M_{249}, which is of interest. In this marking *Next Send* has the value 4 while *Next Rec* has the value 5. Moreover, there is an acknowledgment with value 5 positioned



at place *C*. However, there is also an "old" acknowledgment with value 3 positioned at *C*. This acknowledgment must have been there quite a while. It was created when packet number 2 was successfully received, i.e., before receiving packets number 3 and 4. The old acknowledgment has been overtaken by several "younger" acknowledgments. However, it may still proceed and cause *NextSend* to be decreased to 3.

Another way to investigate the possible decrease of *NextSend*, is to ask how much *NextSend* can differ from the *NextRec*. This is done by means of the following query, which tells us that the difference can be 3, 2, 1 and 0. This result is consistent with our analysis above. *NextRec* can be at most 5, while *NextSend* is at least 1, but *NextSend* can never be reset to a value less than 2 – because we never have acknowledgments with value 1.



From our analysis above, it is quite obvious that an easy way to improve the simple protocol is to avoid decreasing *Next Send*. This can be done by modifying the arc expression of the arc from *Rec Ack* to *Next Send* – so that it becomes *Int.max(n,k)* instead of *n*. We encourage you to make this modification, construct a new O-graph and repeat the occurrence graph analysis – to convince yourself that the new protocol works as desired. You may also want to construct O-graphs where you allow the network to contain more than two packets/acknowledgments at a time. The size of the O-graphs are as follows:

Limit	Max	Nodes	Arcs
1	no	33	44
2	no	428	1130
3	no	3329	12825
4	no	19520	91220
1	yes	33	44
2	yes	293	764
3	yes	1829	6860
4	yes	9025	43124