

```

1 infrastructure infr // refers to infr.ginf
2
3 // Responder
4 process A {
5     send {"arrived", comp.a} @ (true);
6 }
7
8 process D {
9     loop {
10        receive (proc.x == "dissolve" && proc.y == comp.partner) {x, y} [
11            comp.rank := 2,
12            comp.exPartner := comp.partner,
13            comp.partner := -1
14        ];
15    }
16 }
17
18 process R {
19     loop {
20        receive (proc.x == "propose") {x, y, l, n};
21        spawn(Rprime)
22    }
23 }
24
25 process Rprime {
26     if (comp.lock == 0 && rank(proc.y) < comp.rank){
27         send{"accept", comp.idr, proc.n} @ (receiver.idi == proc.l) [comp.lock := 1];
28         receive {
29             case (comp.partner != -1 && proc.a == "ack" && proc.b == comp.idr && proc.c == proc.l){a, b, c}[
30                 comp.exPartner := comp.partner, comp.partner := proc.l
31             ]: {
32                 send{"dissolve", comp.idr} @ (receiver.idi == comp.exPartner) [
33                     comp.rank := rank(proc.y),
34                     comp.lock := 0
35                 ] print("M $comp.partner$ with W $comp.idr$ !");
36             }
37
38             case (comp.partner == -1 && proc.a == "ack" && proc.b == comp.idr && proc.c == proc.l){a, b, c}[
39                 comp.lock := 0, comp.partner := proc.l, comp.rank := rank(proc.y)
40             ] print("M $comp.partner$ with W $comp.idr$ !");: {
41             }
42
43             case (proc.a == "ack" && proc.b != comp.idr && proc.c == proc.l){a, b, c}[comp.lock := 0]: {
44             }
45         }
46     } else if (comp.lock == 0) {
47         set;
48     }
49 }
50
51 // Initiator
52 process I {
53     send{"propose", comp.a, comp.idi, comp.ref} @ (receiver.a == attributeForRank(comp.ref)) [comp.timer := 0];
54
55     loop {
56         if (comp.lock == 0 && !comp.success && comp.timer == comp.timeout){
57             send{"propose", comp.a, comp.idi, next(comp.ref)} @ (receiver.a == attributeForRank(next
58 (comp.ref))) [
59                 comp.timer := 0,
60                 comp.ref := next(comp.ref)
61             ];
62         } else if (comp.lock == 0 && comp.dissolve) {
63             send{"propose", comp.a, comp.idi, 0} @ (receiver.a == attributeForRank(0)) [
64                 comp.timer := 0,
65                 comp.ref := 0,
66                 comp.dissolve := false,
67                 comp.success := false,
68                 comp.rank := noRank(),
69                 comp.partner := -1
70             ];
71         } else if (comp.lock == 0 && comp.arrival && comp.bof <= comp.rank - 1 && comp.partner != -1) {
72             send{"dissolve", comp.idi} @ (receiver.idr == comp.partner) [
73                 comp.success := false,
74                 comp.ref := comp.bof,
75                 comp.bof := noRank(),
76                 comp.rank := noRank(),
77                 comp.exPartner := comp.partner,
78                 comp.partner := -1

```

```

78     ];
79
80     send{"propose", comp.a, comp.idi, comp.ref} @ (receiver.a == attributeForRank(comp.ref))
[comp.timer := 0];
81     }
82   }
83 }
84
85 process T {
86   loop {
87     waitfor(comp.timer < comp.timeout)[comp.timer := comp.timer + 1];
88     waitfor(100);
89   }
90 }
91
92 process N {
93   loop {
94     receive(proc.x == "accept" && (proc.z != comp.ref || comp.success)) {x, y, z};
95     spawn(Nprime)
96   }
97 }
98
99 process Nprime {
100  send{"ack", -1, comp.idi} @ (receiver.idr == proc.y);
101 }
102
103 process M {
104   loop {
105     receive {
106       case (!comp.success && proc.x == "accept" && proc.z == comp.ref){x, y, z}[
107         comp.lock := 1,
108         comp.success := true,
109         comp.rank := comp.ref,
110         comp.exPartner := comp.partner
111       ]:{
112         send{"ack", proc.y, comp.idi} @ (true) [
113           comp.lock := 0,
114           comp.partner := proc.y
115         ];
116       }
117
118       case (proc.x == "dissolve" && proc.y == comp.partner){x, y}[comp.dissolve := true]:{
119       }
120
121       case (proc.x == "arrived" && comp.bof >= rank(proc.y)){x,y}[
122         comp.arrival := true,
123         comp.bof := rank(proc.y)
124       ]:{
125       }
126     }
127   }
128 }
129
130 // Functions
131
132 function int next(int rank){
133   return (rank + 1) % noRank()
134 }
135
136 function int noRank(){
137   return 2
138 }
139
140 function int rank(int y){
141   return y
142 }
143
144 function int attributeForRank(int rank){
145   return rank
146 }
147
148 // Components
149
150 component {
151   a := 0, // attribute value, 0 or 1
152   partner := -1,
153   exPartner := -1,
154   idi := 0,

```

system.goat

```
155   ref := 0,
156   success := false,
157   arrival := false,
158   dissolve := false,
159   rank := 2,
160   bof := 2,
161   lock := 0,
162   timer := 0,
163   timeout := 20
164 } : I | T | M | N
165
166 environment E {
167   a := 0, // attribute value, 0 or 1
168   partner := -1,
169   exPartner := -1,
170   idr := 0,
171   rank := 2,
172   lock := 0
173 }
174 component E : A | R | D
```