AltaRica
Étude de quelques protocoles

Alain Griffault

14 octobre 2005

Table des matières

Table des figures

Chapitre 1

Le problème de l'ingénierie des protocoles

1.1 Intention

Établir une communication *fiable* entre deux processus. L'un est généralement appelé le producteur, et l'autre le consommateur. Une solution consiste à contruire une couche de communication comprenant un processus émetteur (dans lequel le producteur dépose ses messages) et un processus récepteur (duquel le consommateur reçoit les messages) reliés par un réseau.

1.2 Le code du protocole

L'emetteur exécute le programme suivant :

```
Emetteur {
  tant que vrai faire {
    recevoir(Producteur, message)
    transmettreSurement(Recepteur, message);
  }
}
```

Le récepteur exécute le programme suivant :

```
Recepteur {
  tant que vrai faire {
    recevoirSurement(Emetteur, message);
    emettre(Consommateur, message)
  }
}
```

1.3 Les spécifications

Toute solution doit satisfaire les propriétés suivantes :

P1 Le protocole ne doit pas perdre de message.

P2 Le protocole ne doit pas abimer de message.

P3 Le protocole ne doit pas dupliquer de message.

P4 Le protocole doit conserver l'ordonnancement des messages.

Afin de répondre à ces questions, il est généralement nécessaire de faire des hypothèses sur le type de communication et sur les qualités du réseau.

H1 Communications synchrones.

H2 Communications asynchrones.

H3 Réseau parfait.

H4 Réseau avec pertes unidirectionnelle.

H5 Réseau avec pertes bidirectionnelle.

H6 Réseau avec duplication unidirectionnelle.

H7 Réseau avec duplication bidirectionnelle.

H8 ...

1.4 L'étude réalisée

Nous allons vérifier si plusieurs protocoles de la littérature satisfont les propriétés enoncées en 1.3.
Pour cela, les algorithmes seront modélisés avec AltaRica et les propriétés seront vérifiées avec `acheck`.

Chapitre 2

Le protocole du bit alterné

2.1 Le code de l'algorithme

L'émetteur dans le protocole du bit alterné.

```
parite := true
while true do begin
              recevoir(message);            /* du producteur */
              repeter emettre(message,parite);
                      recevoir(ack,pariteAck);
              jusqu'a parite = pariteAck;
              parite := not parite;
              end;
```

Le récepteur dans le protocole du bit alterné.

```
pariteAck := true
while true do begin
              repeter recevoir(message,parite);
                      emettre(ack,parite);
              jusqu'a parite = pariteAck;
              emettre(message);               /* vers le consommateur */
              pariteAck := not pariteAck;
              end;
```

2.2 Le modèle ALTARICA des composants pour ABP

2.2.1 Le nœud EmetteurABP

2.2.2 Le source AltaRica

```
/** Conception du modèle.
 *  le passage de parametres par valeur n'est pas facile à décrire en AltaRica.
 *    - les valeurs sont codées dans le nom des événements
 *    - la boucle est dépliée pour différencier les valeurs des variables
**/

node EmetteurABP
  state
    co : [0,5];        // abstraction du Compteur Ordinal
  event
```

```
    receive_M, send_M_T, send_M_F, receive_A_T, receive_A_F;
  trans
    co=0                |- receive_M   -> co := 1;
    co=1                |- send_M_T    -> co := 2;
    co=2                |- receive_A_F -> co := 1;
    co=2                |- receive_A_T -> co := 3;
    co=3                |- receive_M   -> co := 4;
    co=4                |- send_M_F    -> co := 5;
    co=5                |- receive_A_F -> co := 0;
    co=5                |- receive_A_T -> co := 4;
  init
    co := 0;
edon
```

2.2.3 La sémantique

La sémantique en terme de système de transition est donnée par la figure 2.1

2.2.4 Le nœud RecepteurABP

2.2.5 Le source AltaRica

```
/** Conception du modèle.
 *  le passage de parametres par valeur n'est pas facile à décrire en AltaRica.
 *    - les valeurs sont codées dans le nom des événements
 *    - la boucle est dépliée pour différencier les valeurs des variables
**/

node RecepteurABP
  state
    co : [0,7];         // abstraction du Compteur Ordinal
  event
    send_M, send_A_T, send_A_F, receive_M_T, receive_M_F;
  trans
    co=0                |- receive_M_F -> co := 1;
    co=0                |- receive_M_T -> co := 2;
    co=1                |- send_A_F    -> co := 0;
    co=2                |- send_A_T    -> co := 3;
    co=3                |- send_M      -> co := 4;
    co=4                |- receive_M_F -> co := 6;
    co=4                |- receive_M_T -> co := 5;
    co=5                |- send_A_T    -> co := 4;
    co=6                |- send_A_F    -> co := 7;
    co=7                |- send_M      -> co := 0;
  init
    co := 0;
edon
```

2.2.6 La sémantique

La sémantique en terme de système de transition est donnée par la figure 2.2

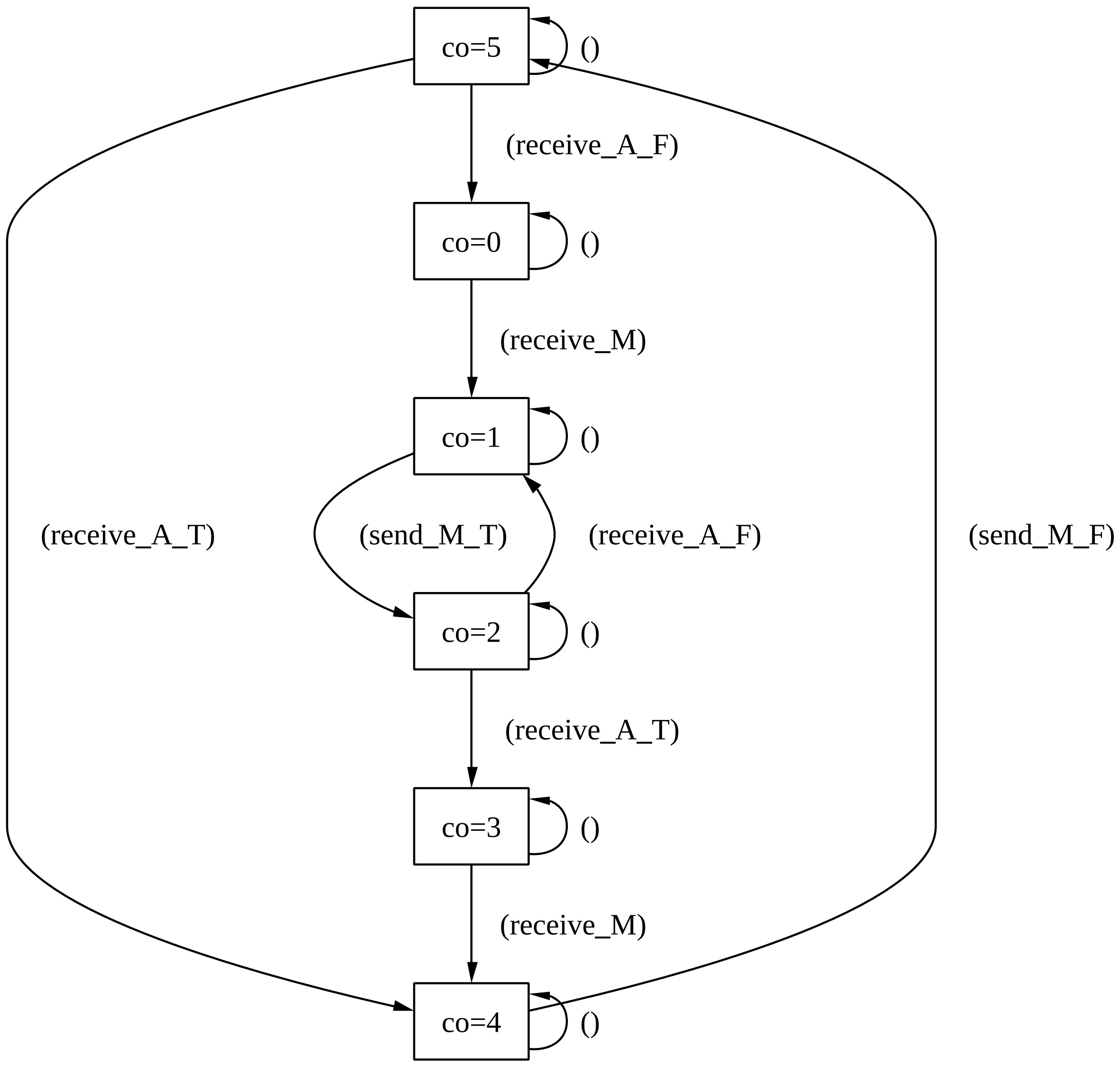

FIG. 2.1 – Le nœud EmetteurABP

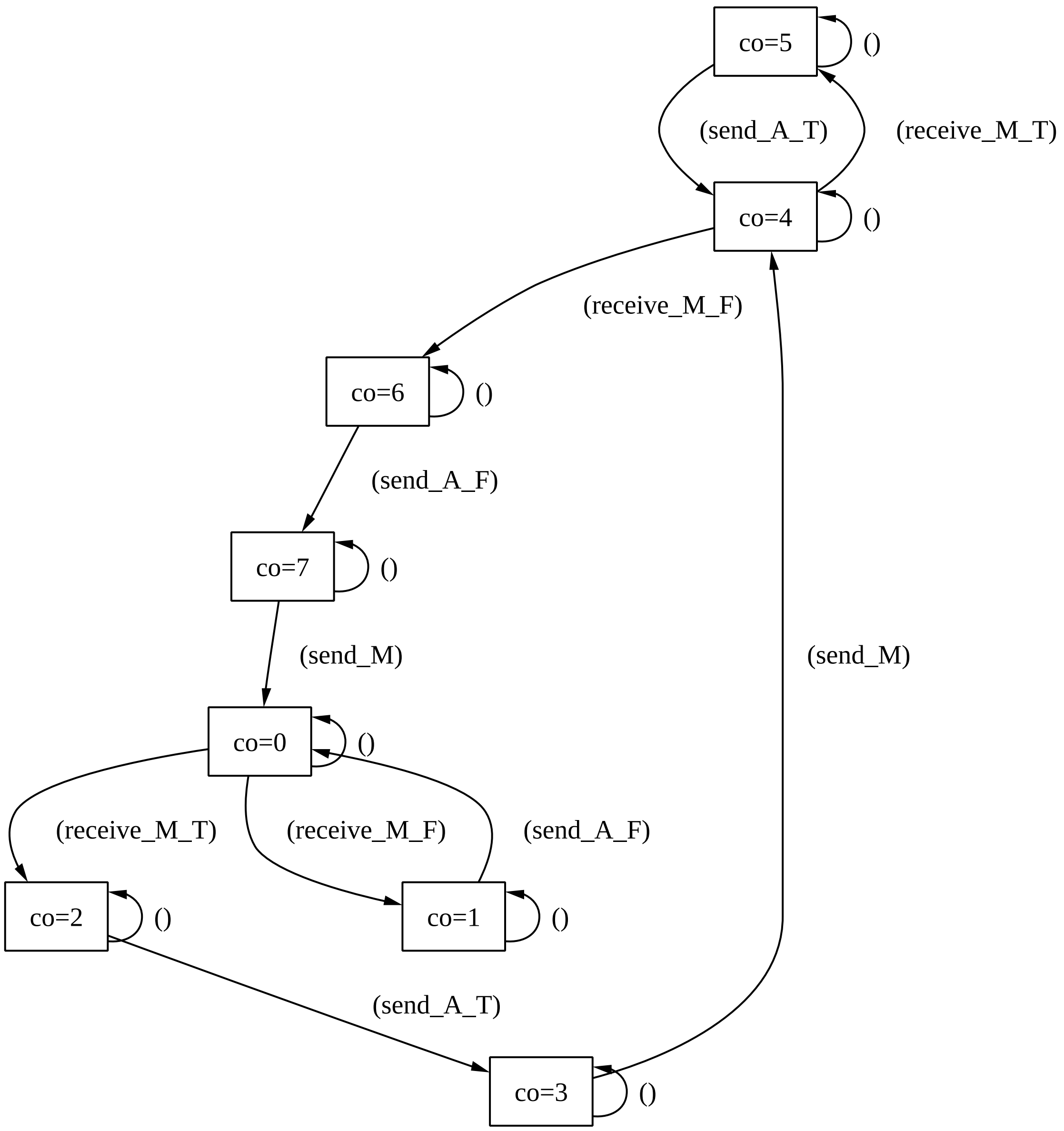

FIG. 2.2 – Le nœud RecepteurABP

2.3 L'étude

2.3.1 Le nœud ABPparfait

2.3.2 Le source AltaRica

```
/** Le modèle de communication est supposé synchrone.
 *    - la réception du message est simultanée à son émission
**/

node ABPparfait
  sub
    E : EmetteurABP;
    R : RecepteurABP;
  sync
    <E.send_M_T, R.receive_M_T>;
    <E.send_M_F, R.receive_M_F>;
    <E.receive_A_T, R.send_A_T>;
    <E.receive_A_F, R.send_A_F>;
edon
```

2.3.3 La sémantique

La sémantique en terme de système de transition est donnée par la figure 2.3

2.3.4 Les propriétés

Les spécifications

```
with ABPparfait do
 dot(any_s,any_t)          >  '$NODENAME.dot';
 dead                      := any_s - src(any_t - self_epsilon);
 notCFC                    := any_t - loop(any_t,any_t);
 show(all)                 >  '$NODENAME.prop';
 test(dead,0)              >  '$NODENAME.res';
 test(notCFC,0)            >> '$NODENAME.res';
done
```

Les résultats

```
/*
 * # state properties : 3
 *
 * initial = 1
 * any_s = 10
 * dead = 0
 *
 * # transition properties : 6
 *
 * self = 10
 * any_t = 22
 * self_epsilon = 10
 * notCFC = 0
 * epsilon = 10
 * not_deterministic = 0
```

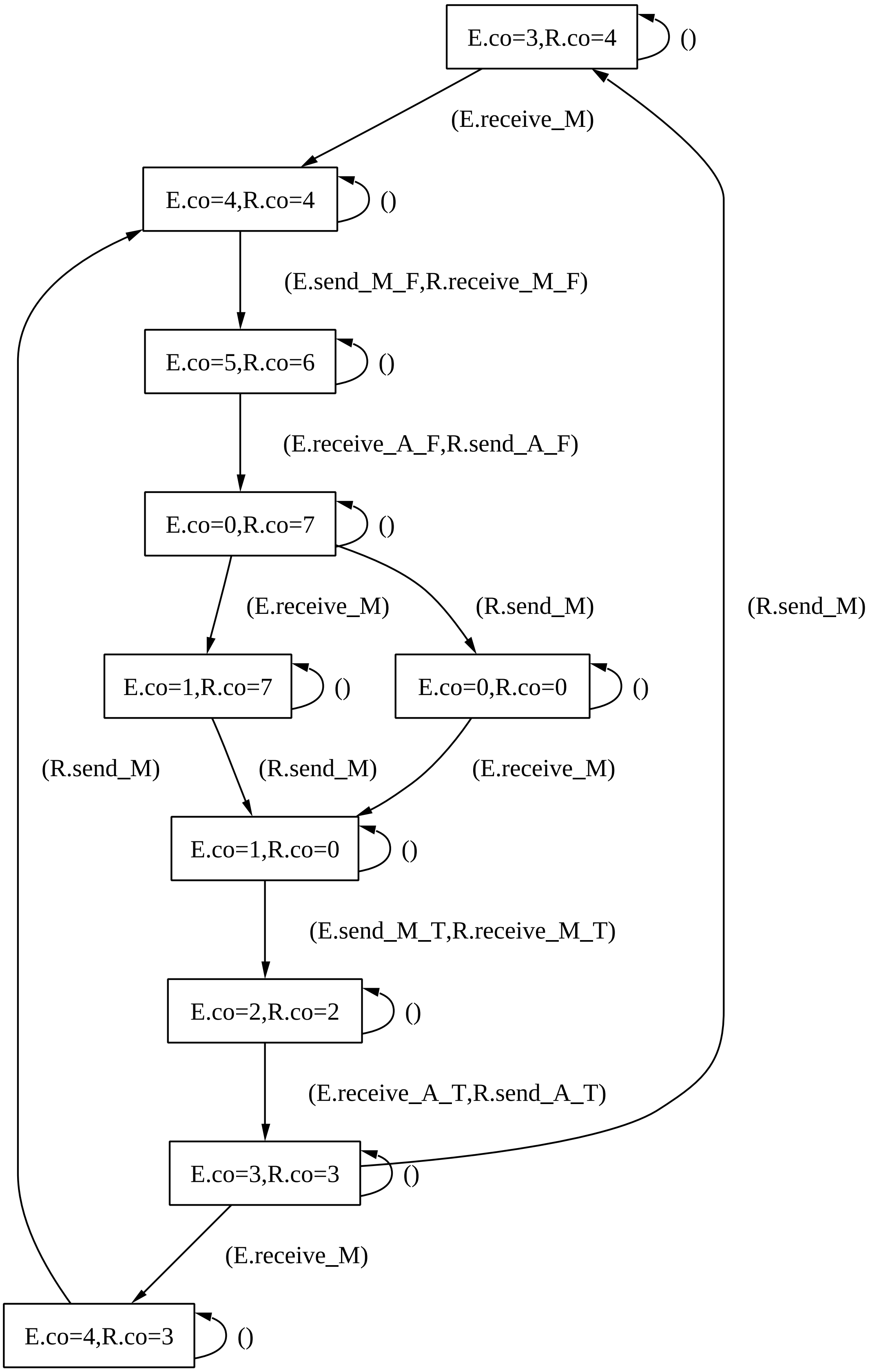

FIG. 2.3 – Le nœud ABPparfait

```
*/
```

Interprétations

```
TEST(dead=0) [PASSED]
TEST(notCFC=0) [PASSED]
```

2.3.5 Le nœud ABPperteUnidirectionnelle

2.3.6 Le source AltaRica

```
/** Le modèle de communication est supposé synchrone.
 *    - la réception du message est simultanée à son émission
 *    - ou bien elle n'a pas lieu en cas de perte
**/

node ABPperteUnidirectionnelle
  sub
    E : EmetteurABP;
    R : RecepteurABP;
  sync
    <E.send_M_T, R.receive_M_T>;
    <E.send_M_F, R.receive_M_F>;
    <E.receive_A_T, R.send_A_T>;
    <E.receive_A_F, R.send_A_F>;
    // les pertes de messages
    <E.send_M_T>;
    <E.send_M_F>;
edon
```

2.3.7 La sémantique

La sémantique en terme de système de transition est donnée par la figure 2.4

2.3.8 Les propriétés

Les spécifications

```
with ABPperteUnidirectionnelle do
 dot(any_s,any_t)         >  '$NODENAME.dot';
 dead                     := any_s - src(any_t - self_epsilon);
 notCFC                   := any_t - loop(any_t,any_t);
 show(all)                >  '$NODENAME.prop';
 test(dead,0)             >  '$NODENAME.res';
 test(notCFC,0)           >> '$NODENAME.res';
 bug  := trace(initial, any_t, dead);
 dot(src(bug)|tgt(bug),bug) > '$NODENAME_bug.dot';
done
```

Les résultats

```
/*
 * # state properties : 3
 *
 * any_s = 14
```

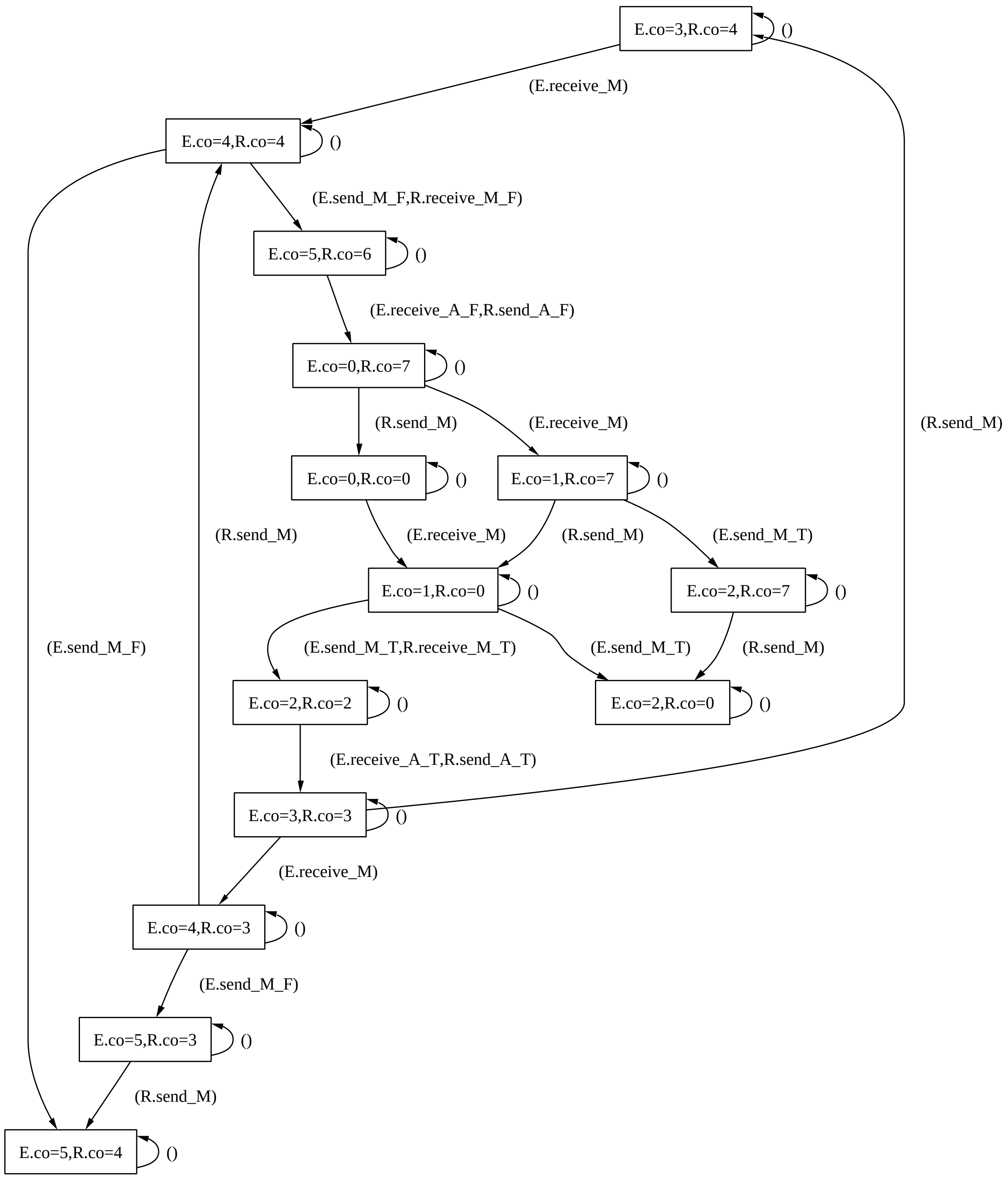

FIG. 2.4 – Le nœud ABPperteUnidirectionnelle

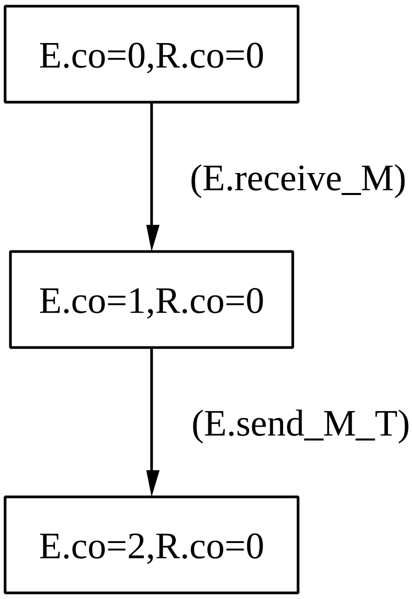

FIG. 2.5 – Un blocage dans le nœud ABPperteUnidirectionnelle

```
 * dead = 2
 * initial = 1
 *
 * # transition properties : 6
 *
 * self = 14
 * epsilon = 14
 * any_t = 32
 * self_epsilon = 14
 * notCFC = 6
 * not_deterministic = 0
 */
```

Interprétations

```
TEST(dead=0) [FAILED] actual size = 2
TEST(notCFC=0) [FAILED] actual size = 6
```

2.3.9 Le nœud ABPerreurUnidirectionnelle

2.3.10 Le source AltaRica

```
/** Le modèle de communication est supposé synchrone.
 *    - la réception du message est simultanée à son émission
 *    - le message reçu est correct ou bien erroné
**/

node ABPerreurUnidirectionnelle
  sub
    E : EmetteurABP;
```

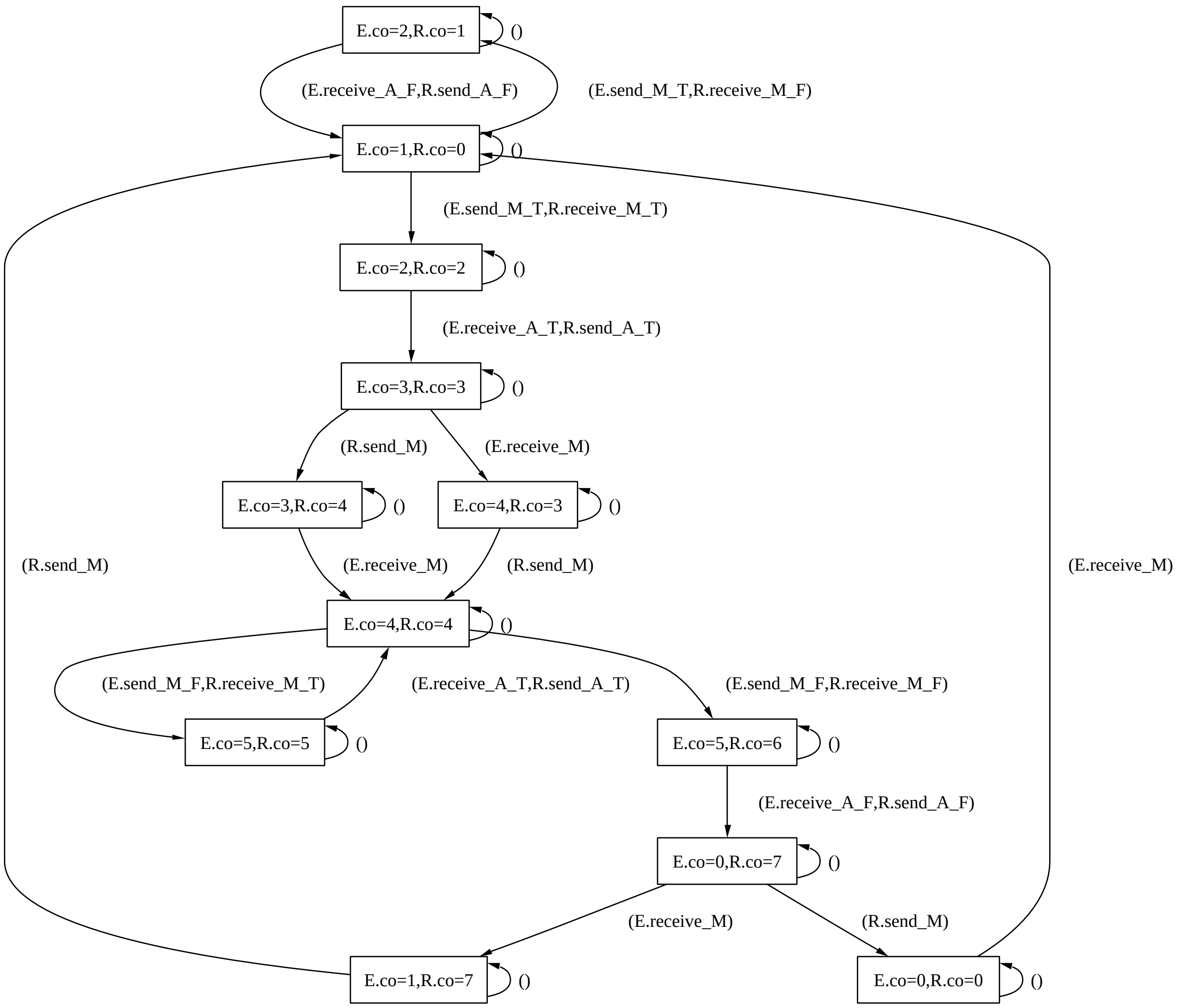

FIG. 2.6 – Le nœud ABPerreurUnidirectionnelle

```
    R : RecepteurABP;
  sync
    <E.send_M_T, R.receive_M_T>;
    <E.send_M_F, R.receive_M_F>;
    <E.receive_A_T, R.send_A_T>;
    <E.receive_A_F, R.send_A_F>;
    // les erreurs de transmission T->F ou F->T est possible
    <E.send_M_T, R.receive_M_F>;
    <E.send_M_F, R.receive_M_T>;
edon
```

2.3.11 La sémantique

La sémantique en terme de système de transition est donnée par la figure 2.6

2.3.12 Les propriétés

Les spécifications

```
with ABPerreurUnidirectionnelle do
 dot(any_s,any_t)          >  '$NODENAME.dot';
 dead                      := any_s - src(any_t - self_epsilon);
 notCFC                    := any_t - loop(any_t,any_t);
 show(all)                 >  '$NODENAME.prop';
 test(dead,0)              >  '$NODENAME.res';
 test(notCFC,0)            >> '$NODENAME.res';
done
```

Les résultats

```
/*
 * # state properties : 3
 *
 * any_s = 12
 * dead = 0
 * initial = 1
 *
 * # transition properties : 6
 *
 * epsilon = 12
 * not_deterministic = 0
 * any_t = 28
 * self_epsilon = 12
 * notCFC = 0
 * self = 12
 */
```

Interprétations

```
TEST(dead=0) [PASSED]
TEST(notCFC=0) [PASSED]
```

2.4 Conclusion

Dans cette étude, nous avons montré que : Sous l'hypothèse de communications synchrones (**H1**), le protocole du bit alterné vérifie les propriétés suivantes :

1. **P1, P2, P3, sans blocage** si le réseau est parfait.
2. **P1, P2, P3, sans blocage** si un seul sens du réseau abime les messages.
3. **P1, P2, P3 avec risque de blocage** si un seul sens du réseau perd les messages.