



# Introduction

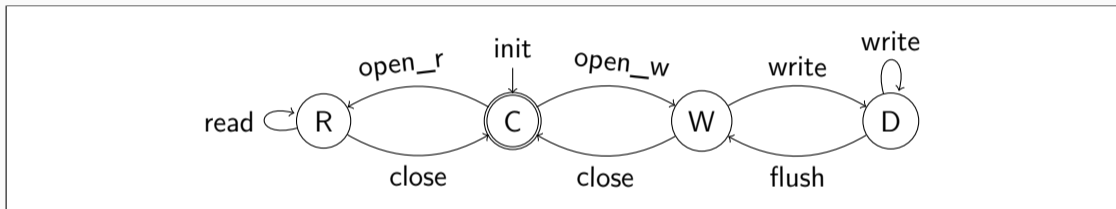


+ ACSL



Robert E. Strom and Shaula Yemini. Typestate: A programming language concept for enhancing software reliability. *IEEE Trans. Software Eng.*, 12(1):157171, 1986.  
[doi:10.1109/TSE.1986.6312929](https://doi.org/10.1109/TSE.1986.6312929)

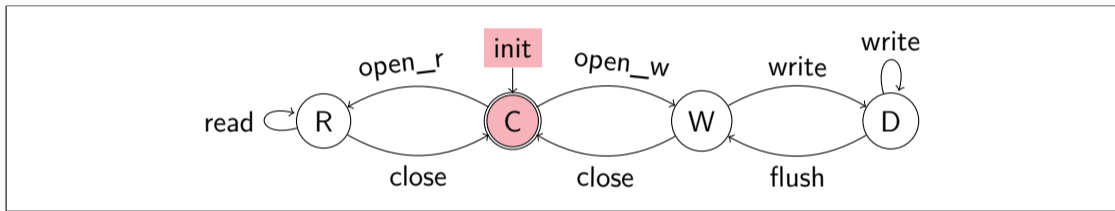
C: Closed, R: Read-mode, W: Write-mode, D: Draft



```

1 FILE f;
2 init(&f);
3 open_w(&f);
4 write(&f);
5 close(&f);
  
```

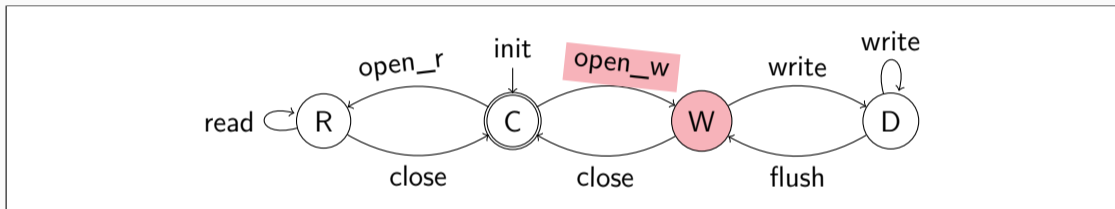
C: Closed, R: Read-mode, W: Write-mode, D: Draft



```

1 FILE f;
2 init(&f);
3 open_w(&f);
4 write(&f);
5 close(&f);
  
```

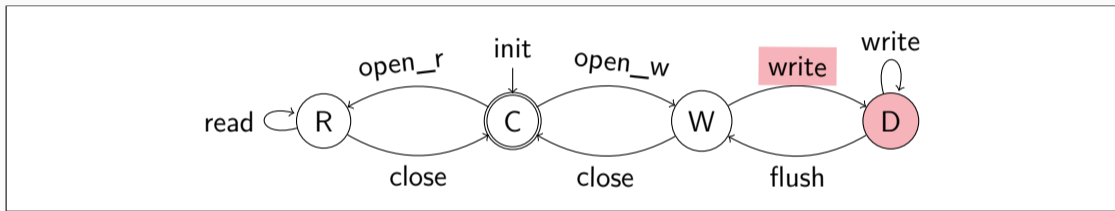
C: Closed, R: Read-mode, W: Write-mode, D: Draft



```

1 FILE f;
2 init(&f);
3 open_w(&f);
4 write(&f);
5 close(&f);
  
```

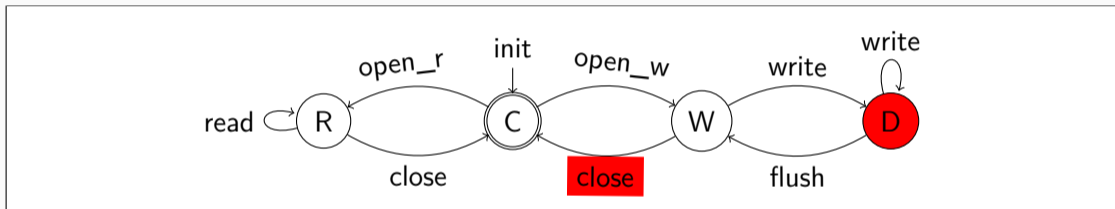
C: Closed, R: Read-mode, W: Write-mode, D: Draft



```

1 FILE f;
2 init(&f);
3 open_w(&f);
4 write(&f);
5 close(&f);
  
```

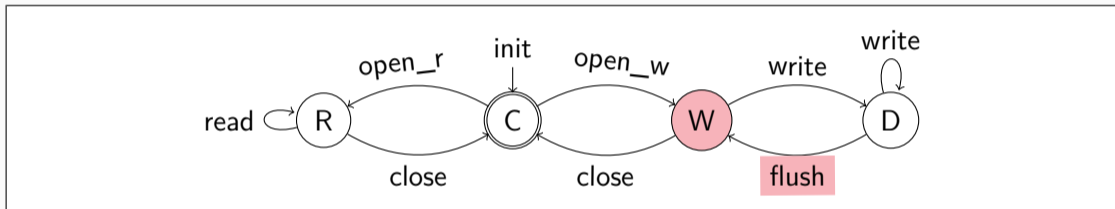
C: Closed, R: Read-mode, W: Write-mode, D: Draft



```

1 FILE f;
2 init(&f);
3 open_w(&f);
4 write(&f);
5 close(&f);
  
```

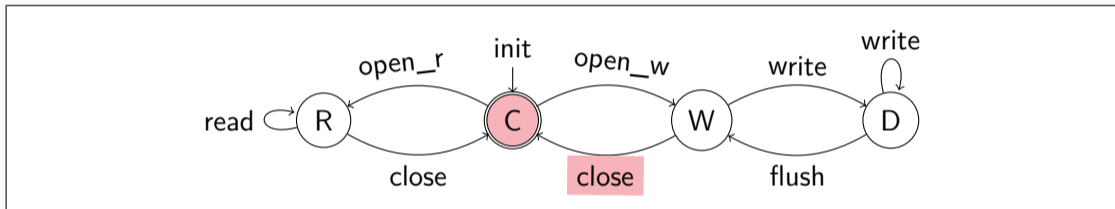
C: Closed, R: Read-mode, W: Write-mode, D: Draft



```

1  FILE f;
2  init(&f);
3  open_w(&f);
4  write(&f);
5  flush(&f);
6  close(&f);
  
```

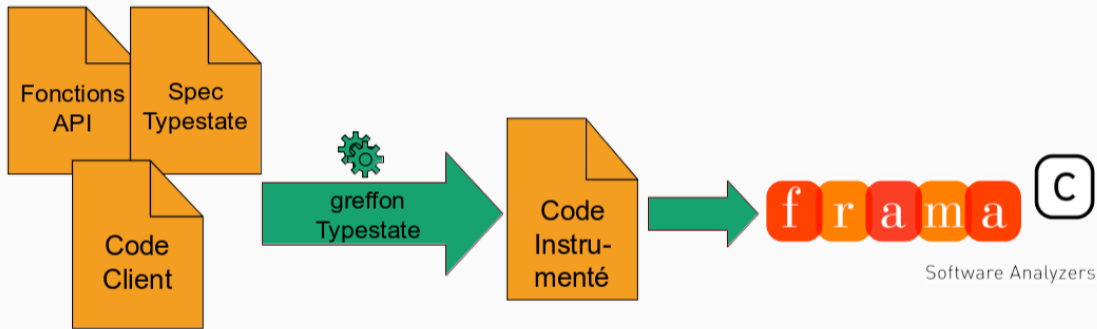
C: Closed, R: Read-mode, W: Write-mode, D: Draft

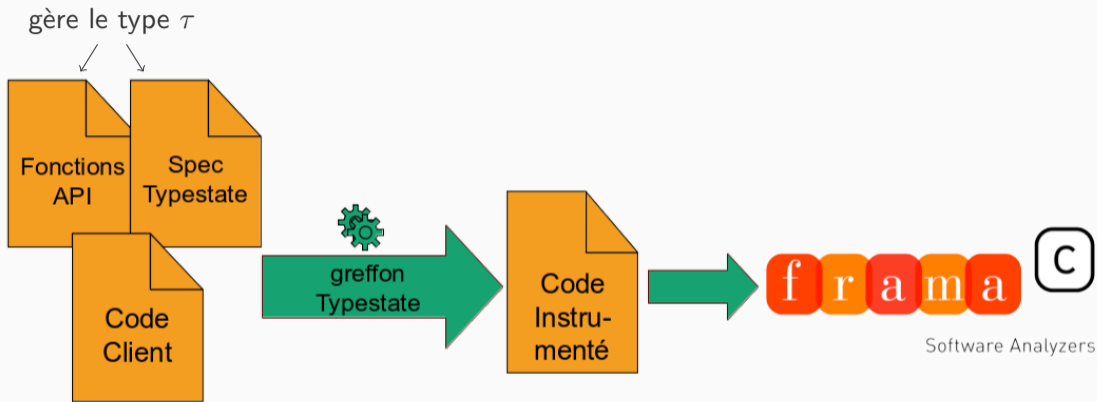


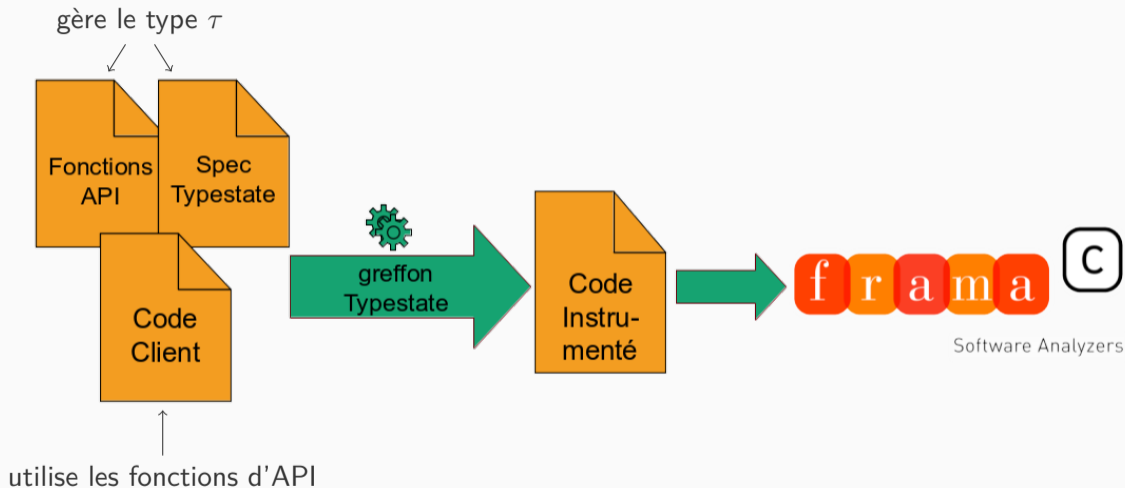
```

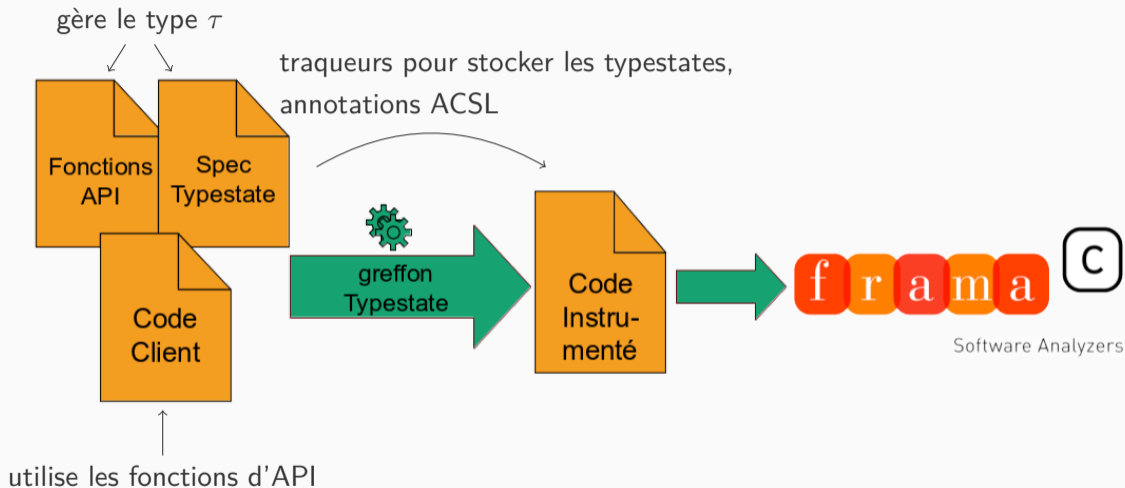
1  FILE f;
2  init(&f);
3  open_w(&f);
4  write(&f);
5  flush(&f);
6  close(&f);
  
```

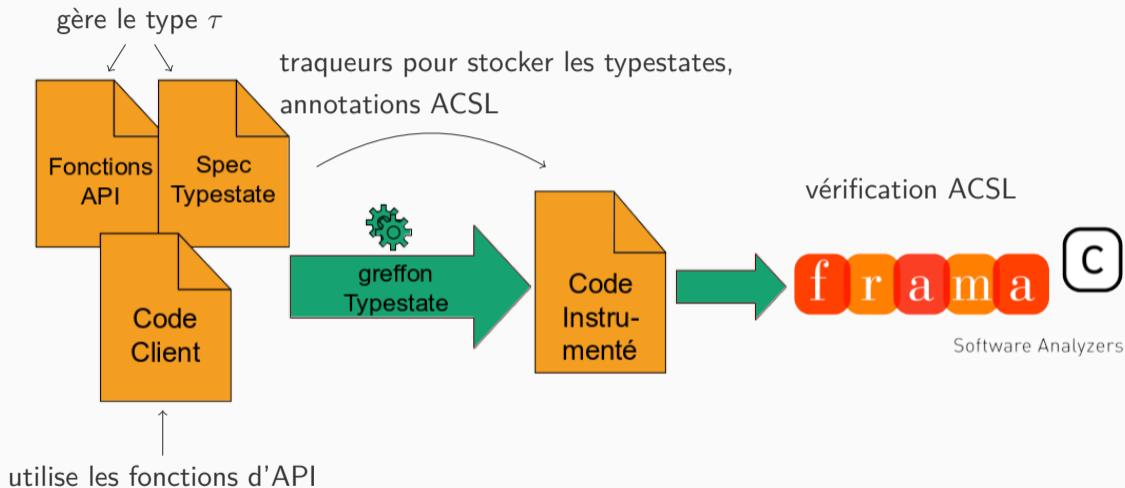
# Greffon Typestates

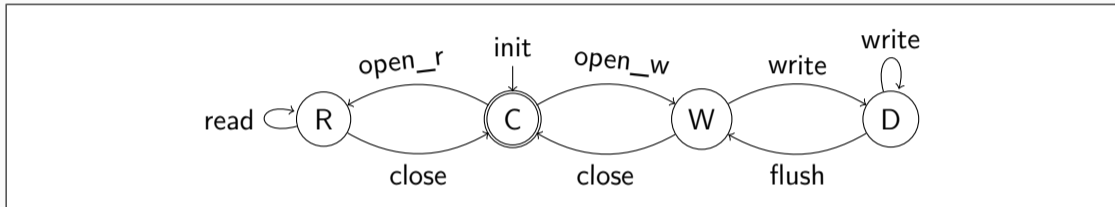












```

1  \def, (FILE) fd == C | R | W | D;
2  \final{C};
3  \trans, f==Undef, {init}, f==C;
4  \trans, f==C, {open_w}, f==W;
5  \trans, f==C, {open_r}, f==R;
6  \trans, f==R, {read}, f==R;
  
```

```

7  \trans, f==W, {write}, f==D;
8  \trans, f==D, {write}, f==D;
9  \trans, f==D, {flush}, f==W;
10 \trans, f==W, {close}, f==C;
11 \trans, f==R, {close}, f==C;
  
```

## Type des traqueurs

```
1 | enum ts_fd {Undef = 0, C = 1, R = 2, W = 3, D = 4};
```

## Type des traqueurs

```
1 enum ts_fd {Undef = 0, C = 1, R = 2, W = 3, D = 4};
```

## Spécification tpestate

```
1 {f:W} close {f:C}
2 {f:R} close {f:C}
```

## Fonction d'origine

```
1 FILE close(FILE *f)
```

## Fonction instrumentée

```
1 /*@
2   requires *tr_f == W || *tr_f == R;
3
4   behavior ts_0:
5     assumes *tr_f == W;
6     ensures *tr_f == C;
7
8   behavior ts_1:
9     assumes *tr_f == R;
10    ensures *tr_f == C;
11    complete behaviors;
12    disjoint behaviors;
13 */
14 FILE close(FILE *f)
15 /*@ ghost (enum fd \ghost *tr_f)*/
```

## Type des traqueurs

```
1 enum ts_fd {Undef = 0, C = 1, R = 2, W = 3, D = 4};
```

## Spécification tpestate

```
1 {f:W} close {f:C}
2 {f:R} close {f:C}
```

## Fonction d'origine

```
1 FILE close(FILE *f)
```

## Fonction instrumentée

```
1 /*@
2   requires *tr_f == W || *tr_f == R;
3
4   behavior ts_0:
5     assumes *tr_f == W;
6     ensures *tr_f == C;
7
8   behavior ts_1:
9     assumes *tr_f == R;
10    ensures *tr_f == C;
11    complete behaviors;
12    disjoint behaviors;
13 */
14 FILE close(FILE *f)
15 /*@ ghost (enum fd \ghost *tr_f)*/
```

## Type des traqueurs

```
1 enum ts_fd {Undef = 0, C = 1, R = 2, W = 3, D = 4};
```

## Spécification tpestate

```
1 {f:W} close {f:C}
2 {f:R} close {f:C}
```

## Fonction d'origine

```
1 FILE close(FILE *f)
```

## Fonction instrumentée

```
1 /*@
2   requires *tr_f == W || *tr_f == R;
3
4   behavior ts_0:
5     assumes *tr_f == W;
6     ensures *tr_f == C;
7
8   behavior ts_1:
9     assumes *tr_f == R;
10    ensures *tr_f == C;
11  complete behaviors;
12  disjoint behaviors;
13 */
14 FILE close(FILE *f)
15 /*@ ghost (enum fd \ghost *tr_f)*/
```

## Type des traqueurs

```
1 | enum ts_fd {Undef = 0, C = 1, R = 2, W = 3, D = 4};
```

## Spécification tpestate

```
1 | {f:W} close {f:C}
2 | {f:R} close {f:C}
```

## Fonction d'origine

```
1 | FILE close(FILE *f)
```

## Fonction instrumentée

```
1 | /*@
2 |   requires *tr_f == W || *tr_f == R;
3 |
4 |   behavior ts_0:
5 |     assumes *tr_f == W;
6 |     ensures *tr_f == C;
7 |
8 |   behavior ts_1:
9 |     assumes *tr_f == R;
10 |    ensures *tr_f == C;
11 |   complete behaviors;
12 |   disjoint behaviors;
13 | */
14 | FILE close(FILE *f)
15 | /*@ ghost (enum fd \ghost *tr_f)*/
```

## Type des traqueurs

```
1 | enum ts_fd {Undef = 0, C = 1, R = 2, W = 3, D = 4};
```

## Spécification tpestate

```
1 | {f:W} close {f:C}
2 | {f:R} close {f:C}
```

## Fonction d'origine

```
1 | FILE close(FILE *f)
```

## Fonction instrumentée

```
1 | /*@
2 |   requires *tr_f == W || *tr_f == R;
3 |
4 |   behavior ts_0:
5 |     assumes *tr_f == W;
6 |     ensures *tr_f == C;
7 |
8 |   behavior ts_1:
9 |     assumes *tr_f == R;
10 |    ensures *tr_f == C;
11 |   complete behaviors;
12 |   disjoint behaviors;
13 | */
14 | FILE close(FILE *f)
15 |   /*@ ghost (enum fd \ghost *tr_f)*/
```

## Code-client d'origine

```

1  int main(){
2      FILE f; FILE *p;
3      p = & f;
4      init(p);
5      open_w(p);
6      write(p);
7      flush(p);
8      close(& f);
9      return 0;
10 }
11

```

## Code-client instrumenté

```

1  int main(){
2      /*@ ghost enum fd tr_f;
3          enum fd \ghost *tr_p; */
4      FILE f; FILE* p;
5      /*@ ghost tr_f = 0; */
6      p = & f;
7      //@ ghost tr_p = & tr_f;
8      init(p) /*@ ghost (tr_p)*/;
9      open_w(p) /*@ ghost (tr_p)*/;
10     write(p) /*@ ghost (tr_p)*/;
11     flush(p) /*@ ghost (tr_p)*/;
12     close(& f) /*@ ghost (& tr_f)*/;
13     //@ assert tr_f == C;
14     return 0;
15 }
16

```

## Code-client d'origine

```

1  int main(){
2  FILE f; FILE *p;
3  p = & f;
4  init(p);
5  open_w(p);
6  write(p);
7  flush(p);
8  close(& f);
9  return 0;
10 }
11

```

## Code-client instrumenté

```

1  int main(){
2  /*@ ghost enum fd tr_f;
3  enum fd \ghost *tr_p; */
4  FILE f; FILE* p;
5  /*@ ghost tr_f = 0; */
6  p = & f;
7  //@ ghost tr_p = & tr_f;
8  init(p) /*@ ghost (tr_p)*;/
9  open_w(p) /*@ ghost (tr_p)*;/
10 write(p) /*@ ghost (tr_p)*;/
11 flush(p) /*@ ghost (tr_p)*;/
12 close(& f) /*@ ghost (& tr_f)*;/
13 //@ assert tr_f == C;
14 return 0;
15 }
16

```

## Code-client d'origine

```

1  int main(){
2      FILE f; FILE *p;
3      p = & f;
4      init(p);
5      open_w(p);
6      write(p);
7      flush(p);
8      close(& f);
9      return 0;
10 }
11

```

## Code-client instrumenté

```

1  int main(){
2      /*@ ghost enum fd tr_f;
3          enum fd \ghost *tr_p; */
4      FILE f; FILE* p;
5      /*@ ghost tr_f = 0; */
6      p = & f;
7      /*@ ghost tr_p = & tr_f;
8          init(p) /*@ ghost (tr_p)*/;
9          open_w(p) /*@ ghost (tr_p)*/;
10         write(p) /*@ ghost (tr_p)*/;
11         flush(p) /*@ ghost (tr_p)*/;
12         close(& f) /*@ ghost (& tr_f)*/;
13         /*@ assert tr_f == C;
14         return 0;
15     }
16

```

## Code-client d'origine

```

1  int main(){
2      FILE f; FILE *p;
3      p = & f;
4      init(p);
5      open_w(p);
6      write(p);
7      flush(p);
8      close(& f);
9      return 0;
10 }
11

```

## Code-client instrumenté

```

1  int main(){
2      /*@ ghost enum fd tr_f;
3          enum fd \ghost *tr_p; */
4      FILE f; FILE* p;
5      /*@ ghost tr_f = 0; */
6      p = & f;
7      //@ ghost tr_p = & tr_f;
8      init(p) /*@ ghost (tr_p)*/;
9      open_w(p) /*@ ghost (tr_p)*/;
10     write(p) /*@ ghost (tr_p)*/;
11     flush(p) /*@ ghost (tr_p)*/;
12     close(& f) /*@ ghost (& tr_f)*/;
13     //@ assert tr_f == C;
14     return 0;
15 }
16

```

## Code-client d'origine

```

1  int main(){
2      FILE f; FILE *p;
3      p = & f;
4      init(p);
5      open_w(p);
6      write(p);
7      flush(p);
8      close(& f);
9      return 0;
10 }
11

```

## Code-client instrumenté

```

1  int main(){
2      /*@ ghost enum fd tr_f;
3          enum fd \ghost *tr_p; */
4      FILE f; FILE* p;
5      /*@ ghost tr_f = 0; */
6      p = & f;
7      //@ ghost tr_p = & tr_f;
8      init(p) /*@ ghost (tr_p)*/;
9      open_w(p) /*@ ghost (tr_p)*/;
10     write(p) /*@ ghost (tr_p)*/;
11     flush(p) /*@ ghost (tr_p)*/;
12     close(& f) /*@ ghost (& tr_f)*/;
13     //@ assert tr_f == C;
14     return 0;
15 }
16

```

## Appel de FRAMA-C en ligne de commande

```
1 | frama-c -typestates flush.c -then-last -wp -wp-rte -wp-smoke-tests
```

## Sortie de FRAMA-C

```
1 | [kernel] Parsing flush.c (with preprocessing)
2 | [wp] Running WP plugin...
3 | [wp] 34 goals scheduled
4 | [wp] Proved goals:    34 / 34
5 |   Qed:                16 (3ms-2ms-6ms)
6 |   Alt-Ergo 2.6.0:    18 (0.50ms-6ms-24ms)
7 |   Smoke Tests:      10 / 10
```

## Appel de FRAMA-C en ligne de commande

```
1 | frama-c -typestates flush.c -then-last -wp -wp-rte -wp-smoke-tests
```

## Sortie de FRAMA-C

```
1 | [kernel] Parsing flush.c (with preprocessing)
2 | [wp] Running WP plugin...
3 | [wp] 34 goals scheduled
4 | [wp] Proved goals:    34 / 34
5 |   Qed:                16 (3ms-2ms-6ms)
6 |   Alt-Ergo 2.6.0:    18 (0.50ms-6ms-24ms)
7 |   Smoke Tests:       10 / 10
```

## Appel de FRAMA-C en ligne de commande

```
1 | frama-c -typestates flush.c -then-last -wp -wp-rte -wp-smoke-tests
```

## Sortie de FRAMA-C

```
1 | [kernel] Parsing flush.c (with preprocessing)
2 | [wp] Running WP plugin...
3 | [wp] 34 goals scheduled
4 | [wp] Proved goals:    34 / 34
5 |   Qed:                16 (3ms-2ms-6ms)
6 |   Alt-Ergo 2.6.0:    18 (0.50ms-6ms-24ms)
7 |   Smoke Tests:      10 / 10
```

## Appel de FRAMA-C en ligne de commande

```
1 | frama-c -typestates flush.c -then-last -wp -wp-rte -wp-smoke-tests
```

## Sortie de FRAMA-C

```
1 | [kernel] Parsing flush.c (with preprocessing)
2 | [wp] Running WP plugin...
3 | [wp] 34 goals scheduled
4 | [wp] Proved goals: 34 / 34
5 |   Qed:                16 (3ms-2ms-6ms)
6 |   Alt-Ergo 2.6.0:    18 (0.50ms-6ms-24ms)
7 |   Smoke Tests:      10 / 10
```

## Appel de FRAMA-C en ligne de commande

```
1 | frama-c -typestates flush.c -then-last -wp -wp-rte -wp-smoke-tests
```

## Sortie de FRAMA-C

```
1 | [kernel] Parsing flush.c (with preprocessing)
2 | [wp] Running WP plugin...
3 | [wp] 34 goals scheduled
4 | [wp] Proved goals:    34 / 34
5 |   Qed:                16 (3ms-2ms-6ms)
6 |   Alt-Ergo 2.6.0:    18 (0.50ms-6ms-24ms)
7 |   Smoke Tests:      10 / 10
```

Wp (vérification déductive), EVA (interprétation abstraite), E-ACSL (vérification à l'exécution)

## Mode d'ouverture en paramètre

```
1 | int open(FILE *f, char mode);  
2 |  
3 | \trans, \guard(mode=='R'), f==C, {open}, f==R;  
4 | \trans, \guard(mode=='W'), f==C, {open}, f==W;
```

## Mode d'ouverture en paramètre

```

1 | int open(FILE *f, char mode);
2 |
3 | \trans, \guard(mode=='R'), f==C, {open}, f==R;
4 | \trans, \guard(mode=='W'), f==C, {open}, f==W;
  
```

## Code d'erreur en valeur de retour

```

1 | \trans, \guard(mode=='R'), f==C, {open}, \post(\result!=0), f==C;
2 | \trans, \guard(mode=='W'), f==C, {open}, \post(\result!=0), f==C;
3 | \trans, \guard(mode=='R'), f==C, {open}, \post(\result==0), f==R;
4 | \trans, \guard(mode=='W'), f==C, {open}, \post(\result==0), f==W;
  
```

## Mode d'ouverture en paramètre

```

1 | int open(FILE *f, char mode);
2 |
3 | \trans, \guard(mode=='R'), f==C, {open}, f==R;
4 | \trans, \guard(mode=='W'), f==C, {open}, f==W;
  
```

## Code d'erreur en valeur de retour

```

1 | \trans, \guard(mode=='R'), f==C, {open}, \post(\result!=0), f==C;
2 | \trans, \guard(mode=='W'), f==C, {open}, \post(\result!=0), f==C;
3 | \trans, \guard(mode=='R'), f==C, {open}, \post(\result==0), f==R;
4 | \trans, \guard(mode=='W'), f==C, {open}, \post(\result==0), f==W;
  
```

## Mode d'ouverture en paramètre

```

1 | int open(FILE *f, char mode);
2 |
3 | \trans, \guard(mode=='R'), f==C, {open}, f==R;
4 | \trans, \guard(mode=='W'), f==C, {open}, f==W;
  
```

## Code d'erreur en valeur de retour

```

1 | \trans, \guard(mode=='R'), f==C, {open}, \post(\result!=0), f==C;
2 | \trans, \guard(mode=='W'), f==C, {open}, \post(\result!=0), f==C;
3 | \trans, \guard(mode=='R'), f==C, {open}, \post(\result==0), f==R;
4 | \trans, \guard(mode=='W'), f==C, {open}, \post(\result==0), f==W;
  
```

```

1  /*@ requires (*tr_f == C && (mode == 'R')) || (*tr_f == C && (mode == 'W'));
2
3  behavior ts_0:
4      assumes *tr_f == C && (mode == 'R');
5      ensures trans_complete: (\result == 0) || (\result != 0);
6      ensures trans_disjoint: !((\result == 0)) || !((\result != 0));
7      ensures (\result == 0) ==> *tr_f == R;
8      ensures (\result != 0) ==> *tr_f == C;
9
10 behavior ts_1:
11     assumes *tr_f == C && (mode == 'W');
12     [...]
13
14 complete behaviors ts_0, ts_1;
15 disjoint behaviors ts_0, ts_1;
16 */
17 int open(FILE *f, char mode) /*@ ghost (enum ts_fd \ghost *tr_f) */;
  
```

```

1  /*@ requires (*tr_f == C && (mode == 'R')) || (*tr_f == C && (mode == 'W'));
2
3  behavior ts_0:
4      assumes *tr_f == C && (mode == 'R');
5      ensures trans_complete: (\result == 0) || (\result != 0);
6      ensures trans_disjoint: !((\result == 0)) || !((\result != 0));
7      ensures (\result == 0) ==> *tr_f == R;
8      ensures (\result != 0) ==> *tr_f == C;
9
10 behavior ts_1:
11     assumes *tr_f == C && (mode == 'W');
12     [...]
13
14 complete behaviors ts_0, ts_1;
15 disjoint behaviors ts_0, ts_1;
16 */
17 int open(FILE *f, char mode) /*@ ghost (enum ts_fd \ghost *tr_f) */;
    
```

```

1  /*@ requires (*tr_f == C && (mode == 'R')) || (*tr_f == C && (mode == 'W'));
2
3  behavior ts_0:
4      assumes *tr_f == C && (mode == 'R');
5      ensures trans_complete: (\result == 0) || (\result != 0);
6      ensures trans_disjoint: !((\result == 0)) || !((\result != 0));
7      ensures (\result == 0) ==> *tr_f == R;
8      ensures (\result != 0) ==> *tr_f == C;
9
10 behavior ts_1:
11     assumes *tr_f == C && (mode == 'W');
12     [...]
13
14 complete behaviors ts_0, ts_1;
15 disjoint behaviors ts_0, ts_1;
16 */
17 int open(FILE *f, char mode) /*@ ghost (enum ts_fd \ghost *tr_f) */;
  
```

```

1 | \trans, \guard(mode=='R'), f==C, {open}, \post(\result==0), f==R;
2 | \trans, \guard(mode=='R'), f==C, {open}, \post(\result!=0), f==C;
    
```

### Code-client d'origine

```

1 | int main() {
2 |     FILE f; int res;
3 |     init(&f);
4 |     res = open(&f, 'R');
5 |     if(res==0) {
6 |         read(&f);
7 |         close(&f);
8 |     }
9 | }
    
```

### Code-client instrumenté

```

1 | int main(){
2 |     /*@ ghost enum fd tr_f;
3 |     FILE f; int res;
4 |     /*@ ghost tr_f = 0; */
5 |     init(& f) /*@ ghost (& tr_f)*/;
6 |     res=open(& f, 'R') /*@ ghost (& tr_f) */;
7 |     if (res == 0) {
8 |         read(& f) /*@ ghost (& tr_f) */;
9 |         close(& f) /*@ ghost (& tr_f) */;
10 |     }
11 |     //@ assert tr_f == C;
12 |     return 0;
13 | }
    
```

```

1 | \trans, \guard(mode=='R'), f==C, {open}, \post(\result==0), f==R;
2 | \trans, \guard(mode=='R'), f==C, {open}, \post(\result!=0), f==C;
    
```

### Code-client d'origine

```

1 | int main() {
2 |     FILE f; int res;
3 |     init(&f);
4 |     res = open(&f, 'R');
5 |     if(res==0) {
6 |         read(&f);
7 |         close(&f);
8 |     }
9 | }
    
```

### Code-client instrumenté

```

1 | int main(){
2 |     /*@ ghost enum fd tr_f;
3 |     FILE f; int res;
4 |     /*@ ghost tr_f = 0; */
5 |     init(& f) /*@ ghost (& tr_f)*/;
6 |     res=open(& f, 'R') /*@ ghost (& tr_f) */;
7 |     if (res == 0) {
8 |         read(& f) /*@ ghost (& tr_f) */;
9 |         close(& f) /*@ ghost (& tr_f) */;
10 |     }
11 |     //@ assert tr_f == C;
12 |     return 0;
13 | }
    
```

```

1 | \trans, \guard(mode=='R'), f==C, {open}, \post(\result==0), f==R;
2 | \trans, \guard(mode=='R'), f==C, {open}, \post(\result!=0), f==C;
    
```

### Code-client d'origine

```

1 | int main() {
2 |     FILE f; int res;
3 |     init(&f);
4 |     res = open(&f, 'R');
5 |     if(res==0) {
6 |         read(&f);
7 |         close(&f);
8 |     }
9 | }
    
```

### Code-client instrumenté

```

1 | int main(){
2 |     /*@ ghost enum fd tr_f;
3 |     FILE f; int res;
4 |     /*@ ghost tr_f = 0; */
5 |     init(& f) /*@ ghost (& tr_f)*/;
6 |     res=open(& f, 'R') /*@ ghost (& tr_f) */;
7 |     if (res == 0) {
8 |         read(& f) /*@ ghost (& tr_f) */;
9 |         close(& f) /*@ ghost (& tr_f) */;
10 |    }
11 |    //@ assert tr_f == C;
12 |    return 0;
13 | }
    
```

```

1 | \trans, \guard(mode=='R'), f==C, {open}, \post(\result==0), f==R;
2 | \trans, \guard(mode=='R'), f==C, {open}, \post(\result!=0), f==C;
    
```

### Code-client d'origine

```

1 | int main() {
2 |     FILE f; int res;
3 |     init(&f);
4 |     res = open(&f, 'R');
5 |     if(res==0) {
6 |         read(&f);
7 |         close(&f);
8 |     }
9 | }
    
```

### Code-client instrumenté

```

1 | int main(){
2 |     /*@ ghost enum fd tr_f;
3 |     FILE f; int res;
4 |     /*@ ghost tr_f = 0; */
5 |     init(& f) /*@ ghost (& tr_f)*/;
6 |     res=open(& f, 'R') /*@ ghost (& tr_f) */;
7 |     if (res == 0) {
8 |         read(& f) /*@ ghost (& tr_f) */;
9 |         close(& f) /*@ ghost (& tr_f) */;
10 |    }
11 |    //@ assert tr_f == C;
12 |    return 0;
13 | }
    
```

```

1 | \trans, \guard(mode=='R'), f==C, {open}, \post(\result==0), f==R;
2 | \trans, \guard(mode=='R'), f==C, {open}, \post(\result!=0), f==C;
    
```

### Code-client d'origine

```

1 | int main() {
2 |     FILE f; int res;
3 |     init(&f);
4 |     res = open(&f, 'R');
5 |     if(res==0) {
6 |         read(&f);
7 |         close(&f);
8 |     }
9 | }
    
```

### Code-client instrumenté

```

1 | int main(){
2 |     /*@ ghost enum fd tr_f;
3 |     FILE f; int res;
4 |     /*@ ghost tr_f = 0; */
5 |     init(& f) /*@ ghost (& tr_f)*/;
6 |     res=open(& f, 'R') /*@ ghost (& tr_f) */;
7 |     if (res == 0) {
8 |         read(& f) /*@ ghost (& tr_f) */;
9 |         close(& f) /*@ ghost (& tr_f) */;
10 |    }
11 |    //@ assert tr_f == C;
12 |    return 0;
13 | }
    
```

```

1 | \trans, \guard(mode=='R'), f==C, {open}, \post(\result==0), f==R;
2 | \trans, \guard(mode=='R'), f==C, {open}, \post(\result!=0), f==C;

```

### Code-client d'origine

```

1 | int main() {
2 |     FILE f; int res;
3 |     init(&f);
4 |     res = open(&f, 'R');
5 |     if(res==0) {
6 |         read(&f);
7 |         close(&f);
8 |     }
9 | }

```

### Code-client instrumenté

```

1 | int main(){
2 |     /*@ ghost enum fd tr_f;
3 |     FILE f; int res;
4 |     /*@ ghost tr_f = 0; */
5 |     init(& f) /*@ ghost (& tr_f)*/;
6 |     res=open(& f, 'R') /*@ ghost (& tr_f) */;
7 |     if (res == 0) {
8 |         read(& f) /*@ ghost (& tr_f) */;
9 |         close(& f) /*@ ghost (& tr_f) */;
10 |     }
11 |     //@ assert tr_f == C;
12 |     return 0;
13 | }

```

```

1 | \trans, \guard(mode=='R'), f==C, {open}, \post(\result==0), f==R;
2 | \trans, \guard(mode=='R'), f==C, {open}, \post(\result!=0), f==C;
    
```

### Code-client d'origine

```

1 | int main() {
2 |     FILE f; int res;
3 |     init(&f);
4 |     res = open(&f, 'R');
5 |     if(res==0) {
6 |         read(&f);
7 |         close(&f);
8 |     }
9 | }
    
```

### Code-client instrumenté

```

1 | int main(){
2 |     /*@ ghost enum fd tr_f;
3 |     FILE f; int res;
4 |     /*@ ghost tr_f = 0; */
5 |     init(& f) /*@ ghost (& tr_f)*/;
6 |     res=open(& f, 'R') /*@ ghost (& tr_f) */;
7 |     if (res == 0) {
8 |         read(& f) /*@ ghost (& tr_f) */;
9 |         close(& f) /*@ ghost (& tr_f) */;
10 |     }
11 |     //@ assert tr_f == C;
12 |     return 0;
13 | }
    
```

```

1 | \trans, \guard(mode=='R'), f==C, {open}, \post(\result==0), f==R;
2 | \trans, \guard(mode=='R'), f==C, {open}, \post(\result!=0), f==C;
    
```

### Code-client d'origine

```

1 | int main() {
2 |     FILE f; int res;
3 |     init(&f);
4 |     res = open(&f, 'R');
5 |     if(res==0) {
6 |         read(&f);
7 |         close(&f);
8 |     }
9 | }
    
```

### Code-client instrumenté

```

1 | int main(){
2 |     /*@ ghost enum fd tr_f;
3 |     FILE f; int res;
4 |     /*@ ghost tr_f = 0; */
5 |     init(& f) /*@ ghost (& tr_f)*/;
6 |     res=open(& f, 'R') /*@ ghost (& tr_f) */;
7 |     if (res == 0) {
8 |         read(& f) /*@ ghost (& tr_f) */;
9 |         close(& f) /*@ ghost (& tr_f) */;
10 |     }
11 |     /*@ assert tr_f == C;
12 |     return 0;
13 | }
    
```

## Spécification

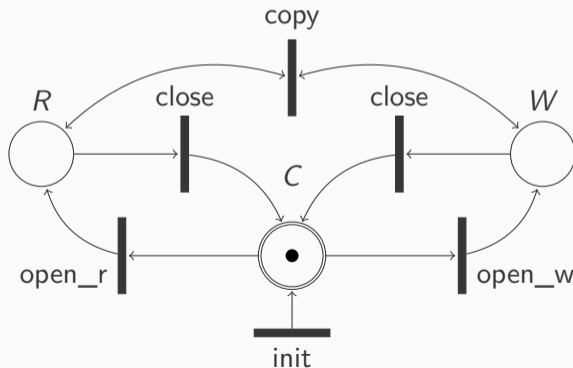
```
1 | \trans, src==R, tgt==W, {copy}, src==R, tgt==W;
```

## Spécification

```
1 | \trans, src==R, tgt==W, {copy}, src==R, tgt==W;
```

## Code-client

```
1 | int f, f2;
2 | init(&f);
3 | open_r(&f);
4 | init(&f2);
5 | open_w(&f2);
6 | copy(&f, &f2);
7 | close(&f);
8 | close(&f2);
```

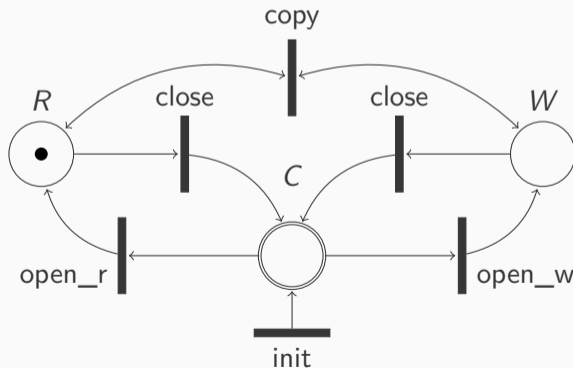


## Spécification

```
1 | \trans, src==R, tgt==W, {copy}, src==R, tgt==W;
```

## Code-client

```
1 | int f, f2;
2 | init(&f);
3 | open_r(&f);
4 | init(&f2);
5 | open_w(&f2);
6 | copy(&f, &f2);
7 | close(&f);
8 | close(&f2);
```

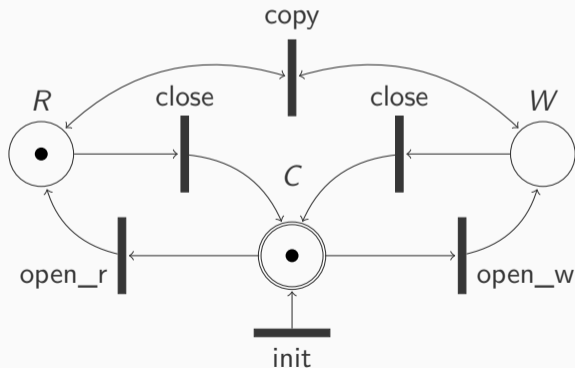


## Spécification

```
1 | \trans, src==R, tgt==W, {copy}, src==R, tgt==W;
```

## Code-client

```
1 | int f, f2;
2 | init(&f);
3 | open_r(&f);
4 | init(&f2);
5 | open_w(&f2);
6 | copy(&f, &f2);
7 | close(&f);
8 | close(&f2);
```

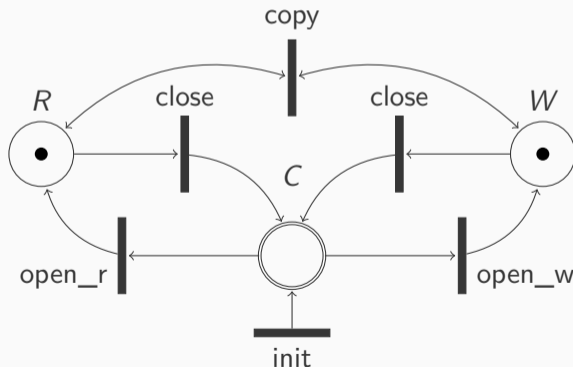


## Spécification

```
1 | \trans, src==R, tgt==W, {copy}, src==R, tgt==W;
```

## Code-client

```
1 | int f, f2;
2 | init(&f);
3 | open_r(&f);
4 | init(&f2);
5 | open_w(&f2);
6 | copy(&f, &f2);
7 | close(&f);
8 | close(&f2);
```

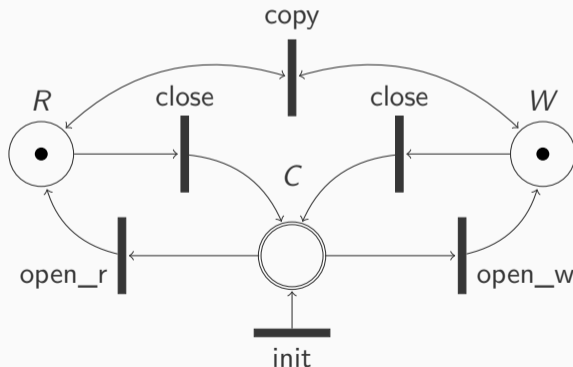


## Spécification

```
1 | \trans, src==R, tgt==W, {copy}, src==R, tgt==W;
```

## Code-client

```
1 | int f, f2;
2 | init(&f);
3 | open_r(&f);
4 | init(&f2);
5 | open_w(&f2);
6 | copy(&f, &f2);
7 | close(&f);
8 | close(&f2);
```

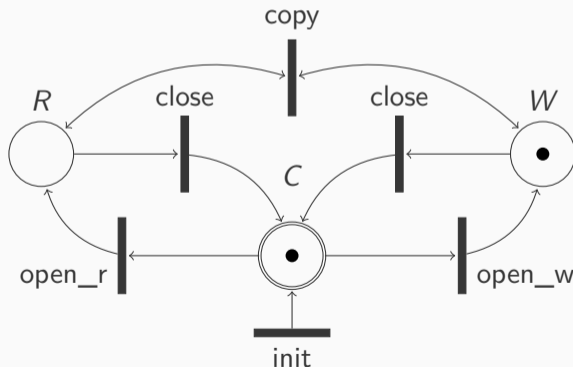


## Spécification

```
1 | \trans, src==R, tgt==W, {copy}, src==R, tgt==W;
```

## Code-client

```
1 | int f, f2;
2 | init(&f);
3 | open_r(&f);
4 | init(&f2);
5 | open_w(&f2);
6 | copy(&f, &f2);
7 | close(&f);
8 | close(&f2);
```

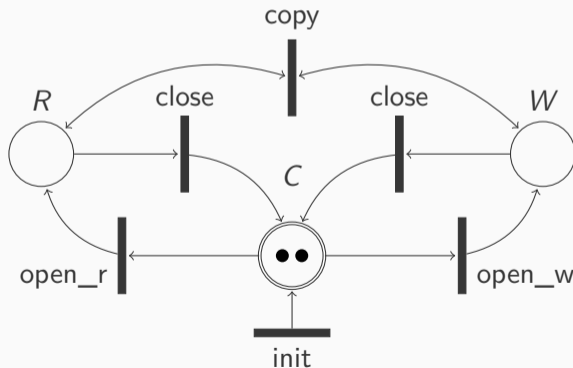


## Spécification

```
1 | \trans, src==R, tgt==W, {copy}, src==R, tgt==W;
```

## Code-client

```
1 | int f, f2;
2 | init(&f);
3 | open_r(&f);
4 | init(&f2);
5 | open_w(&f2);
6 | copy(&f, &f2);
7 | close(&f);
8 | close(&f2);
```



# Conclusion

## Limitations

- > Les objets ne sont modifiables qu'à travers les fonctions de l'API
- > Les transitions typestate doivent être déterministes
- > Allocation dynamique compatible uniquement avec l'analyse par EVA

## Limitations

- > Les objets ne sont modifiables qu'à travers les fonctions de l'API
- > Les transitions typestate doivent être déterministes
- > Allocation dynamique compatible uniquement avec l'analyse par EVA

## Perspectives

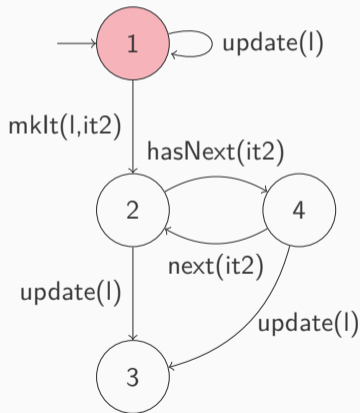
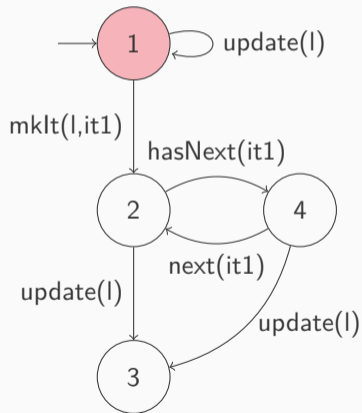
- > Enrichir études de cas (sockets, itérateurs ...)
- > Interaction plus fine avec EVA (domaine abstrait, partitionnement)
- > Bonne formation spécification (réseaux de Petri)
- > Typestates d'objets en interaction

**Merci de votre attention  
Des questions ?**

```

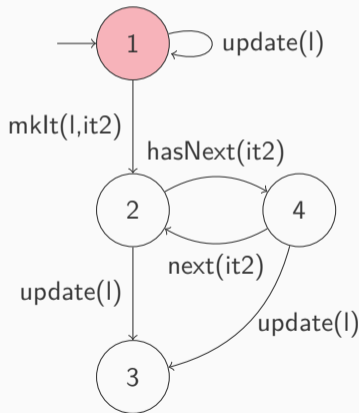
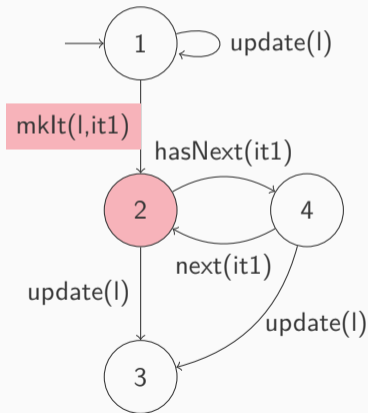
1  Iterator it1 = list_begin(l);
2  Iterator it2 = list_begin(l);
3  if (hasNext(it1)) {add(1,l)};

```



```

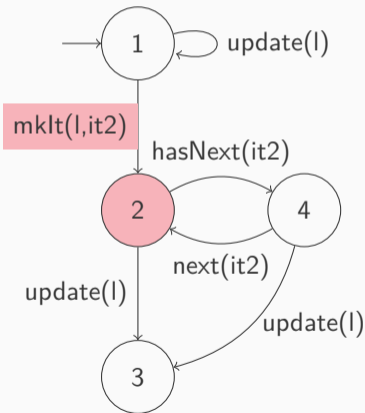
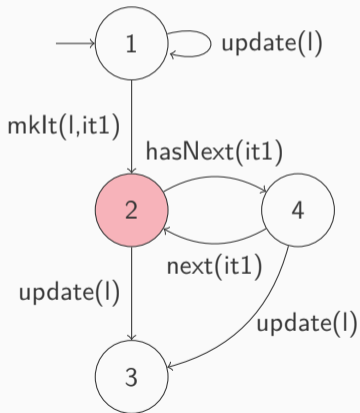
1  Iterator it1 = list_begin(l);
2  Iterator it2 = list_begin(l);
3  if (hasNext(it1)) {add(1,l)};
  
```



```

1  Iterator it1 = list_begin(l);
2  Iterator it2 = list_begin(l);
3  if (hasNext(it1)) {add(1,l)};

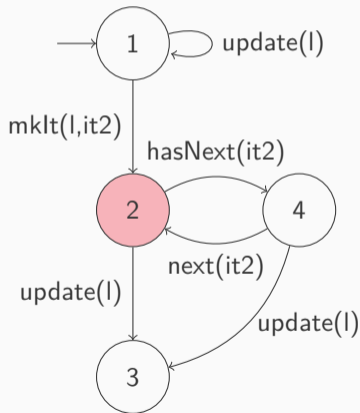
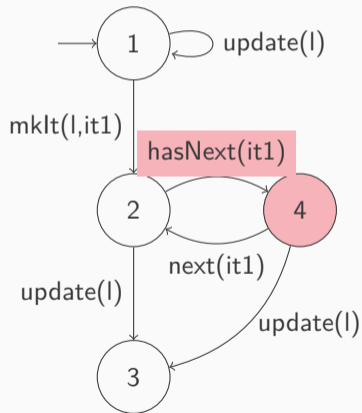
```



```

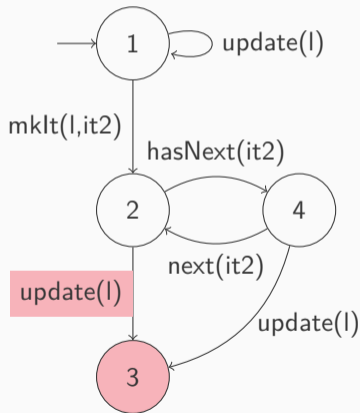
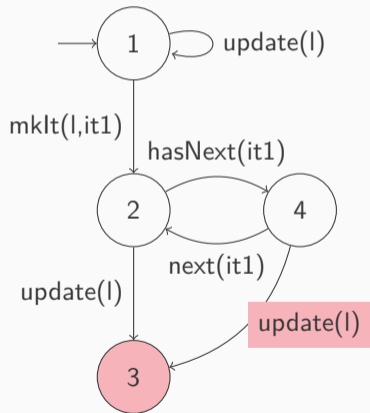
1  Iterator it1 = list_begin(l);
2  Iterator it2 = list_begin(l);
3  if (hasNext(it1)) {add(1,l)};

```



```

1  Iterator it1 = list_begin(l);
2  Iterator it2 = list_begin(l);
3  if (hasNext(it1)) {add(1,l)};
  
```



## Spécification Typestate

```

1 | {f:W} close {f:C}
2 | {f:R} close {f:C}

```

## Fonction d'origine

```

1 | FILE close(FILE *f) {
2 |     ... //update f
3 |     return 0;
4 | }

```

## Fonction instrumentée

```

1 | /*@
2 |     requires *tr_f == W || *tr_f == R;
3 |
4 |     behavior ts_0:
5 |         assumes *tr_f == W;
6 |         ensures *tr_f == C;
7 |
8 |     behavior ts_1:
9 |         assumes *tr_f == R;
10 |        ensures *tr_f == C;
11 |    complete behaviors;
12 |    disjoint behaviors;
13 | */
14 | FILE close(FILE *f)
15 |     /*@ ghost (enum fd \ghost *tr_f) */ {
16 |     ... //update f
17 |     /*@ ghost
18 |     if (*tr_f == W) {
19 |         *tr_f = C;
20 |     } else if (*tr_f == R) {
21 |         *tr_f = C;
22 |     }; */
23 |     return 0;
24 | }

```

## Spécification Typestate

```

1 | {f:W} close {f:C}
2 | {f:R} close {f:C}
  
```

## Fonction d'origine

```

1 | FILE close(FILE *f) {
2 |     ... //update f
3 |     return 0;
4 | }
  
```

## Fonction instrumentée

```

1 | /*@
2 |     requires *tr_f == W || *tr_f == R;
3 |
4 |     behavior ts_0:
5 |         assumes *tr_f == W;
6 |         ensures *tr_f == C;
7 |
8 |     behavior ts_1:
9 |         assumes *tr_f == R;
10 |        ensures *tr_f == C;
11 |    complete behaviors;
12 |    disjoint behaviors;
13 | */
14 | FILE close(FILE *f)
15 |     /*@ ghost (enum fd \ghost *tr_f)*/{
16 |     ... //update f
17 |     /*@ ghost
18 |     if (*tr_f == W) {
19 |         *tr_f = C;
20 |     } else if (*tr_f == R) {
21 |         *tr_f = C;
22 |     }; */
23 |     return 0;
24 | }
  
```

## Spécification Typestate

```

1 | {f:W} close {f:C}
2 | {f:R} close {f:C}

```

## Fonction d'origine

```

1 | FILE close(FILE *f) {
2 |     ... //update f
3 |     return 0;
4 | }

```

## Fonction instrumentée

```

1 | /*@
2 |     requires *tr_f == W || *tr_f == R;
3 |
4 |     behavior ts_0:
5 |         assumes *tr_f == W;
6 |         ensures *tr_f == C;
7 |
8 |     behavior ts_1:
9 |         assumes *tr_f == R;
10 |        ensures *tr_f == C;
11 |    complete behaviors;
12 |    disjoint behaviors;
13 | */
14 | FILE close(FILE *f)
15 |     /*@ ghost (enum fd \ghost *tr_f) */ {
16 |     ... //update f
17 |     /*@ ghost
18 |     if (*tr_f == W) {
19 |         *tr_f = C;
20 |     } else if (*tr_f == R) {
21 |         *tr_f = C;
22 |     }; */
23 |     return 0;
24 | }

```

## Spécification Typestate

```

1 {f:W} close {f:C}
2 {f:R} close {f:C}

```

## Fonction d'origine

```

1 FILE close(FILE *f) {
2     ... //update f
3     return 0;
4 }

```

## Fonction instrumentée

```

1 /*@
2     requires *tr_f == W || *tr_f == R;
3
4     behavior ts_0:
5         assumes *tr_f == W;
6         ensures *tr_f == C;
7
8     behavior ts_1:
9         assumes *tr_f == R;
10        ensures *tr_f == C;
11
12    complete behaviors;
13    disjoint behaviors;
14 */
15 FILE close(FILE *f)
16     /*@ ghost (enum fd \ghost *tr_f) */ {
17     ... //update f
18     /*@ ghost
19     if (*tr_f == W) {
20         *tr_f = C;
21     } else if (*tr_f == R) {
22         *tr_f = C;
23     }; */
24     return 0;
25 }

```

## Spécification Typestate

```

1 | {f:W} close {f:C}
2 | {f:R} close {f:C}

```

## Fonction d'origine

```

1 | FILE close(FILE *f) {
2 |     ... //update f
3 |     return 0;
4 | }

```

## Fonction instrumentée

```

1 | /*@
2 |     requires *tr_f == W || *tr_f == R;
3 |
4 |     behavior ts_0:
5 |         assumes *tr_f == W;
6 |         ensures *tr_f == C;
7 |
8 |     behavior ts_1:
9 |         assumes *tr_f == R;
10 |        ensures *tr_f == C;
11 |    complete behaviors;
12 |    disjoint behaviors;
13 | */
14 | FILE close(FILE *f)
15 |     /*@ ghost (enum fd \ghost *tr_f) */ {
16 |     ... //update f
17 |     /*@ ghost
18 |     if (*tr_f == W) {
19 |         *tr_f = C;
20 |     } else if (*tr_f == R) {
21 |         *tr_f = C;
22 |     }; */
23 |     return 0;
24 | }

```

```

1  int main(){
2  FILE f1, f2;
3  init(&f1);
4  init(&f2);
5  FILE t[2] = {f1, f2};
6  /*@
7   loop assigns i,t[0..1];
8   loop invariant 0<=i<=2;
9   loop ts_invariant \forallall integer j; i<=j<2 ==> \ts(t[j])==C;
10  loop ts_invariant \forallall integer j; 0<=j<i ==> \ts(t[j])==R;
11  loop variant 3-i;
12  */
13  for (int i = 0; i < 2; i++){
14      open(&t[i], 'R');
15  }
16  /*@
17   loop assigns i,t[0..1];
18   loop invariant 0<=i<=2;
19   loop ts_invariant \forallall integer j; i<=j<2 ==> \ts(t[j])==R;
20   loop ts_invariant \forallall integer j; 0<=j<i ==> \ts(t[j])==C;
21   loop variant 3-i;
22  */
23  for (int i = 0; i < 1; i++){
24      close(&t[i]);
25  }
26  return 0;
27  }

```