Το εργαλείο UPPAAL

Στην ενότητα αυτή θα μελετηθούν τα εξής θέματα:

Εισαγωγή στο εργαλείο UPPAAL

- *Γλώσσα Μοντελοποίησης*
- *Ο προσομοιωτής*
- *Ο επαληθευτής*

Εισαγωγή στην UPPAAL

- Το εργαλείο UPPAAL αποτελεί ένα μοντελοελεγκτή μέσω του οποίου μπορούμε να μοντελοποιούμε, να προσομοιώνουμε και να ελέγχουμε τη συμπεριφορά συστημάτων πραγματικού χρόνου
  - Ελεγκτές πραγματικού χρόνου
  - Πρωτόκολλα επικοινωνίας
  - Εφαρμογές πολυμέσων

- Αναπτύχθηκε από
  - Uppsala University + Aalborg University = UPPAAL

- Πρώτη εκδοχή το 1995

- Διατίθεται από: http://www.uppaal.org

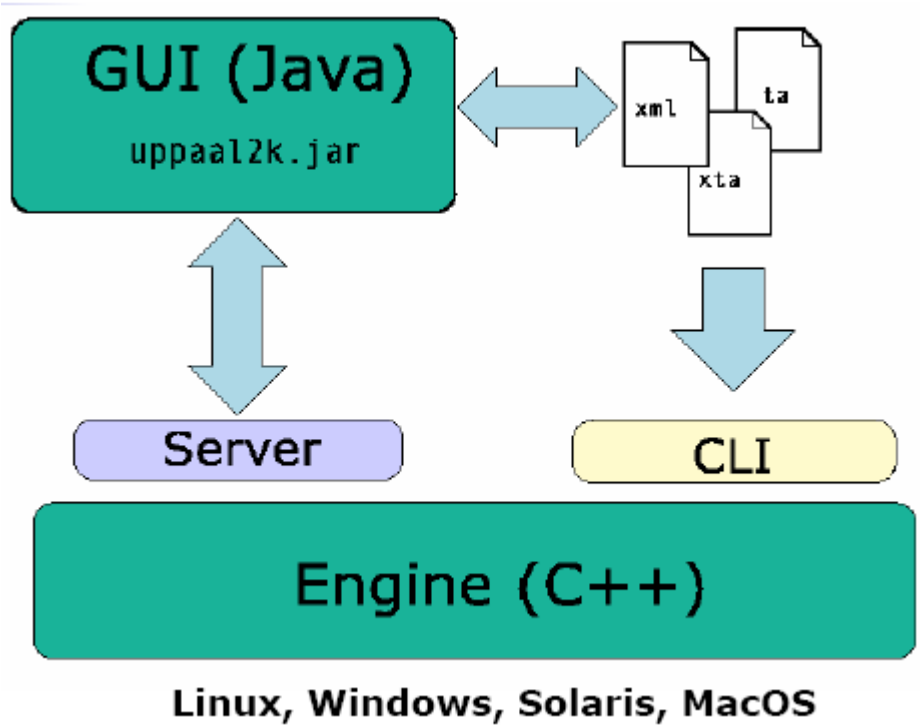
Εισαγωγή στο εργαλείο UPPAAL

- Το εργαλείο UPPAAL λαμβάνει ως είσοδο συστήματα Χρονικών Αυτομάτων τα οποία συνοδεύονται από
  - Ακέραιες μεταβλητές
  - Δομές δεδομένων
  - Κανάλια Επικοινωνίας
  - Κρίσιμες συμπεριφορές

- Υποστηρίζεται ο μοντελοέλεγχος ιδιοτήτων διατυπωμένων στη χρονική επέκταση του λογισμού CTL.

Αρχιτεκτονική Συστήματος

- Περιλαμβάνει:
  - Γραφική διεπιφάνεια
  - Προσομοιωτή
  - Μοντελοελεγκτή



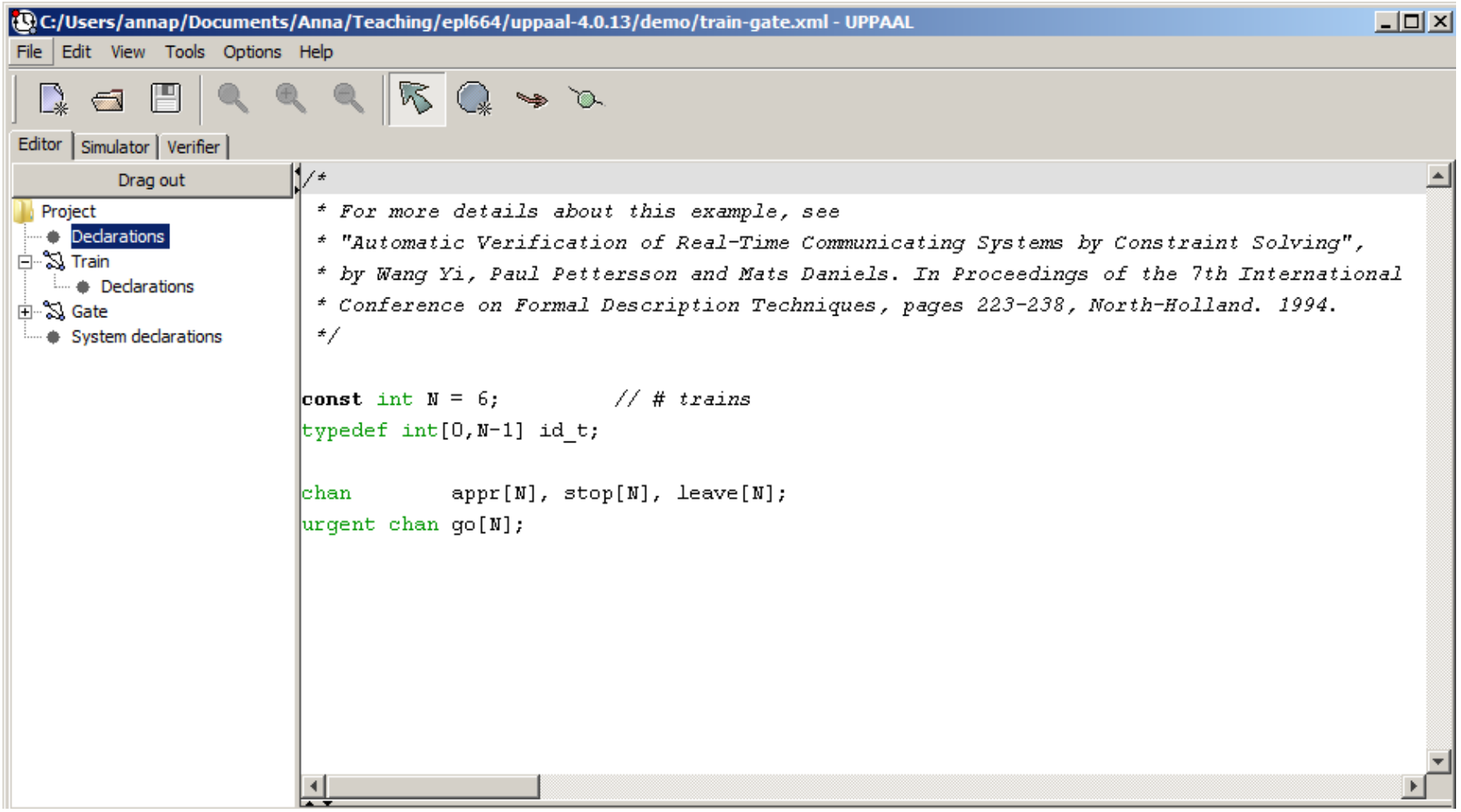
Μοντέλα UPPAAL

- Ένα μοντέλο στην UPPAAL περιλαμβάνει ένα δίκτυο από αυτόματα η περιγραφή των οποίων χωρίζεται σε τρία τμήματα:
  - Οι καθολικές και τοπικές δηλώσεις
  - Οι κλάσεις των αυτομάτων
  - Ο ορισμός του συστήματος

- Το επόμενο παράδειγμα προέρχεται από το Uppaal release (φάκελος demo).

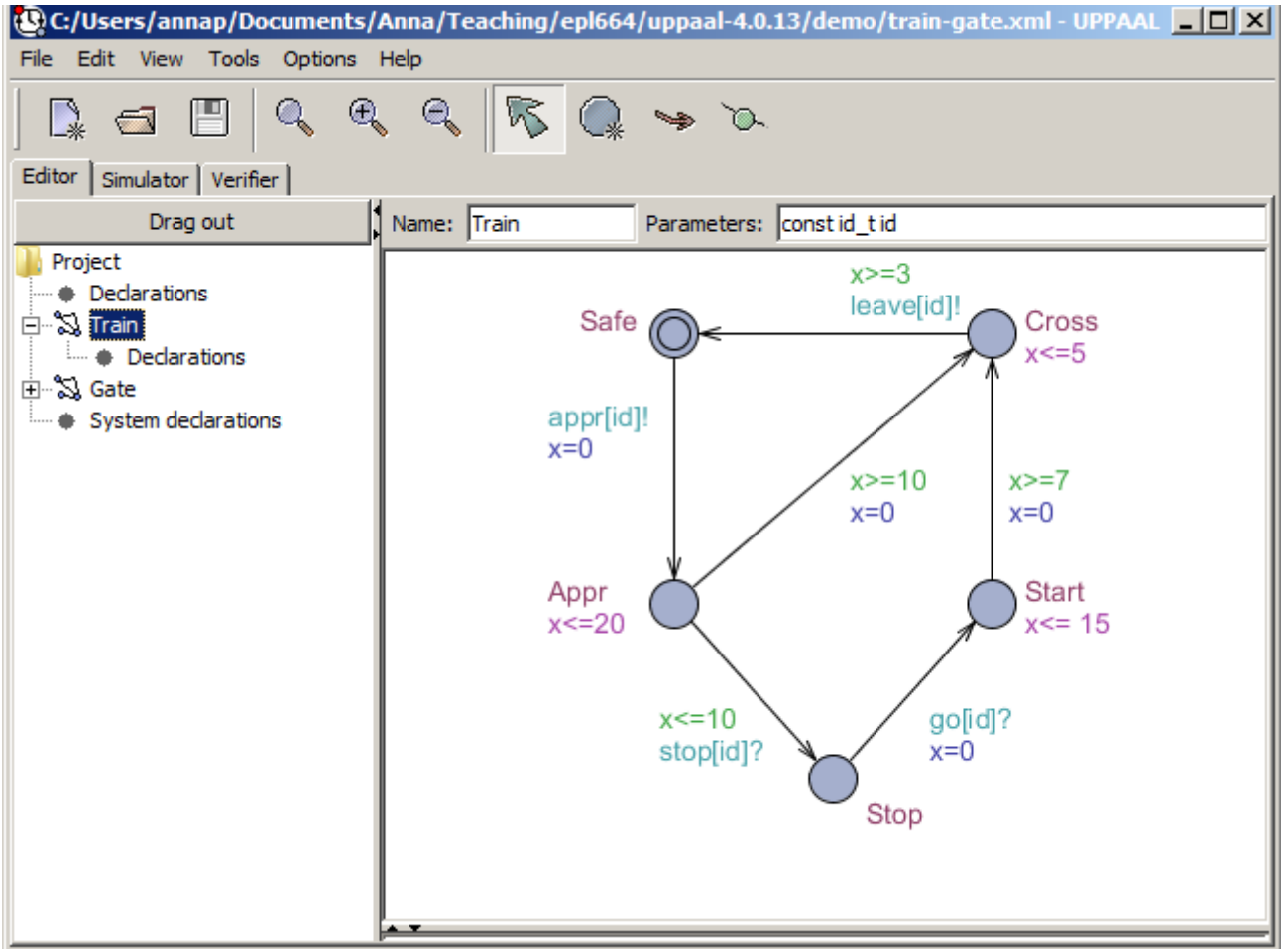
Παράδειγμα

- Καθολικές Δηλώσεις



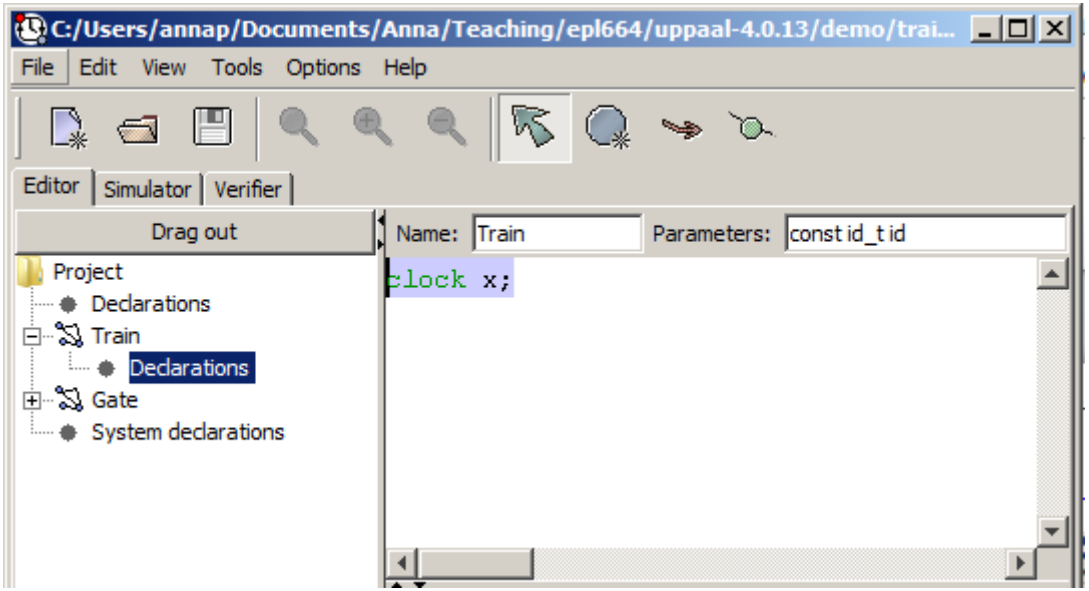
Παράδειγμα (συν.)

- Κλάση αυτόματου Train



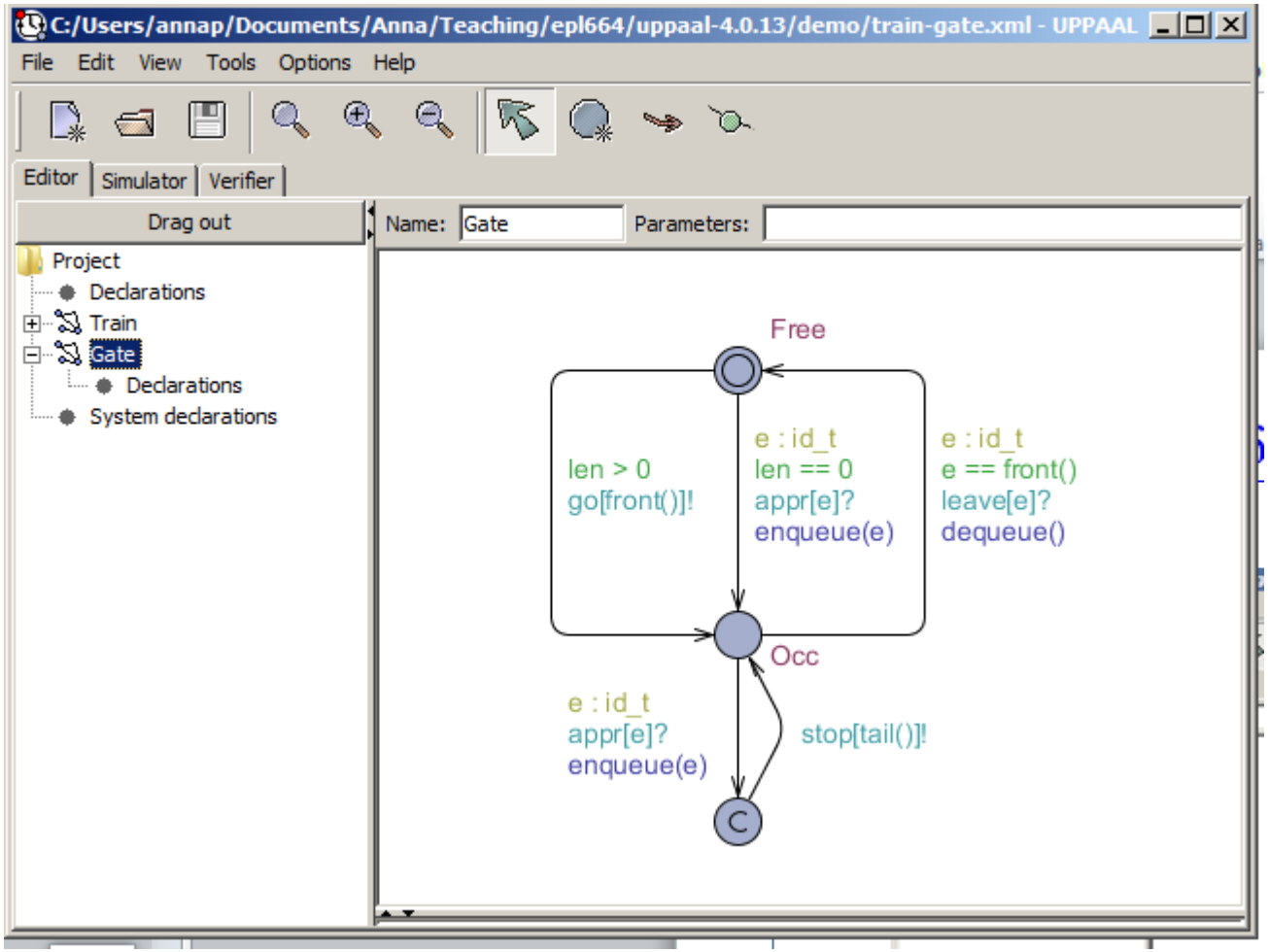
Παράδειγμα (συν.)

- Τοπικές Δηλώσεις αυτομάτου Train



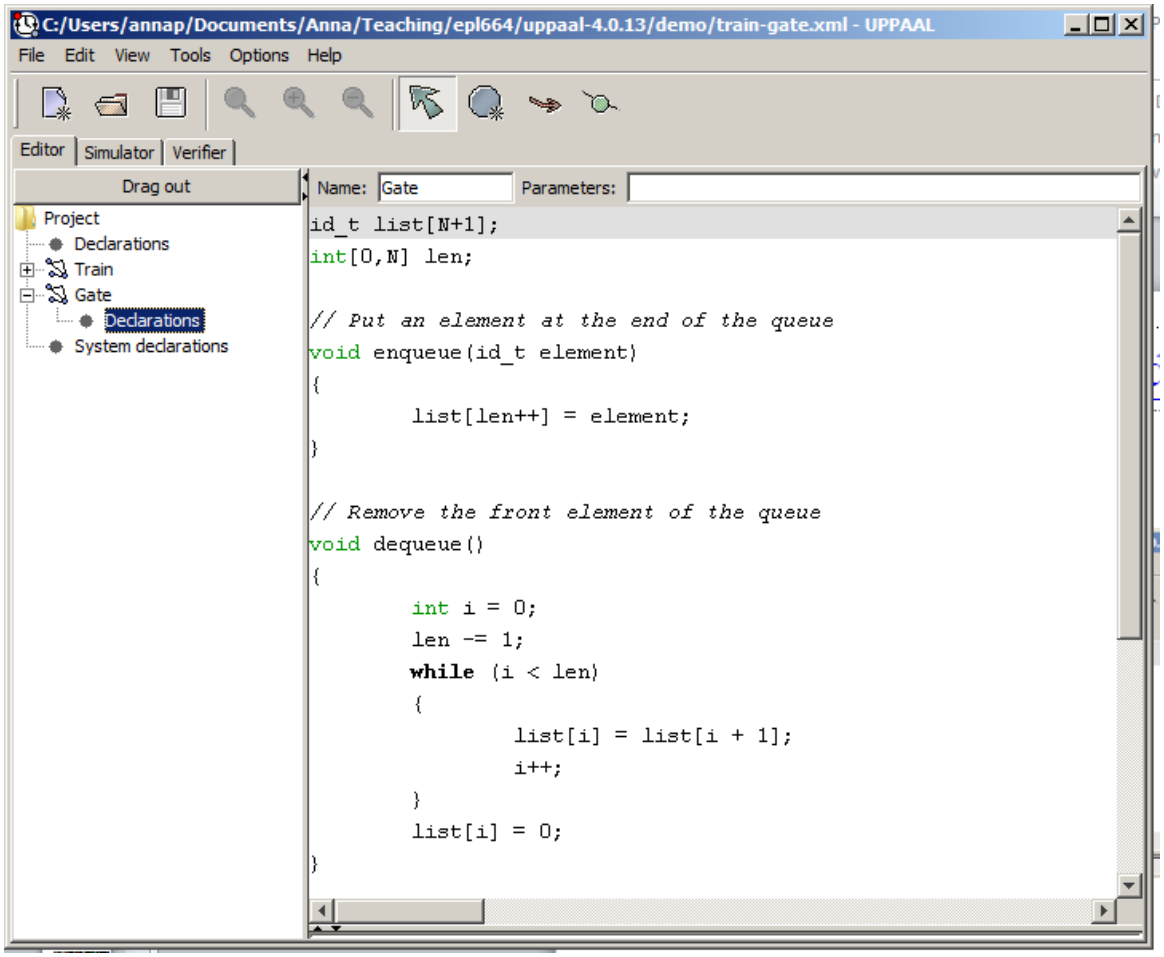
Παράδειγμα (συν.)

- Κλάση αυτόματου Gate



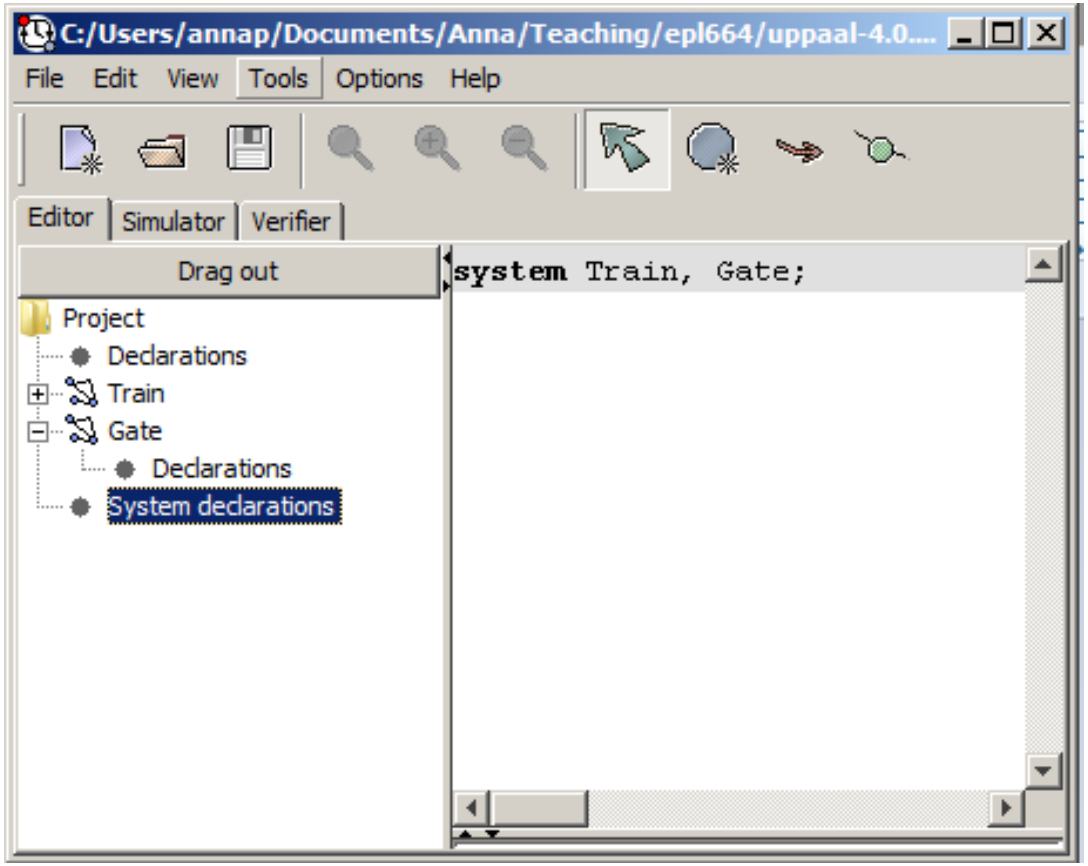
Παράδειγμα (συν.)

- Τοπικές Δηλώσεις αυτομάτου Gate



Παράδειγμα (συν.)

- Το σύστημα



Γλώσσα Μοντελοποίησης

- Τα χρονικά αυτόματα στην Uppaal, επεκτείνουν τον ορισμό που έχουμε δώσει, αφού διαθέτουν
  - παραμέτρους και
  - μεταβλητές τις οποίες μπορούν επεξεργάζονται μέσω συναρτήσεων.

- 4 τύποι δεδομένων
  - int: [-32768, 32767]
  - bool: {true,false}
  - clock: ρολόγια
  - chan: κανάλια

Δηλώσεις

- Οι δηλώσεις μπορούν είτε να αναφέρονται τοπικά σε μία κλάση αυτομάτου, είτε καθολικά σε ένα σύστημα από αυτόματα

- Περιλαμβάνουν
  - Δηλώσεις Ρολογιών
  - Δηλώσεις Ακεραίων
  - Δηλώσεις Καναλιών
  - Δηλώσεις Πινάκων
  - Δηλώσεις Τύπων
  - Δηλώσεις Εγγραφών

Δηλώσεις - Παραδείγματα

- `const int a = 1;`
  Δηλώνεται η σταθερά a τύπου int με τιμή 1
- `bool b[8], c[4];`
  Δηλώνονται δύο πίνακες τύπου boolean, με ονόματα b και c, και μήκος 8 και 4 αντίστοιχα.
- `int[0,100] a=5;`
  Δηλώνεται η ακέραια μεταβλητή a η οποία παίρνει τιμές από το πεδίο [0, 100] και η οποία αρχικοποιείται με 5.
- `int a[2][3] = {{1,2,3},{4,5,6}};`
  Δηλώνεται ένας δυσδιάστατος πίνακας ο οποίος και αρχικοποιείται
- `clock x, y;`
  Δηλώνονται δύο ρολόγια

Δηλώσεις - Παραδείγματα

- `chan d;`

  Δηλώνεται ένα κανάλι

- `struct {int a; bool b;} s1 = {2,true}`

  Δηλώνεται η s1 ως εγγραφή με δύο πεδία, τα a και b, η οποία και αρχικοποιείται.

- `typedef struct {int a; bool b; clock c;} S;`

  Δήλωση εγγραφής

Εκφράσεις

- Εκφράσεις στην Uppaal έχουν ως βασικά δομικά στοιχεία
  - Μεταβλητές
  - Ακέραιοι
- Βάσει αυτών μπορούμε να κτίσουμε:

```
E ::= E [E] | ( E ) | E ++ | E -- | E := E | E = E
    | Unary E | E Binary E | E ? E : E | E.ID
    | …
Unary ::= + | - | ! | not
Binary ::= < | <= | == | != | >= | > | + | - | * | /
         | && | || | …
```

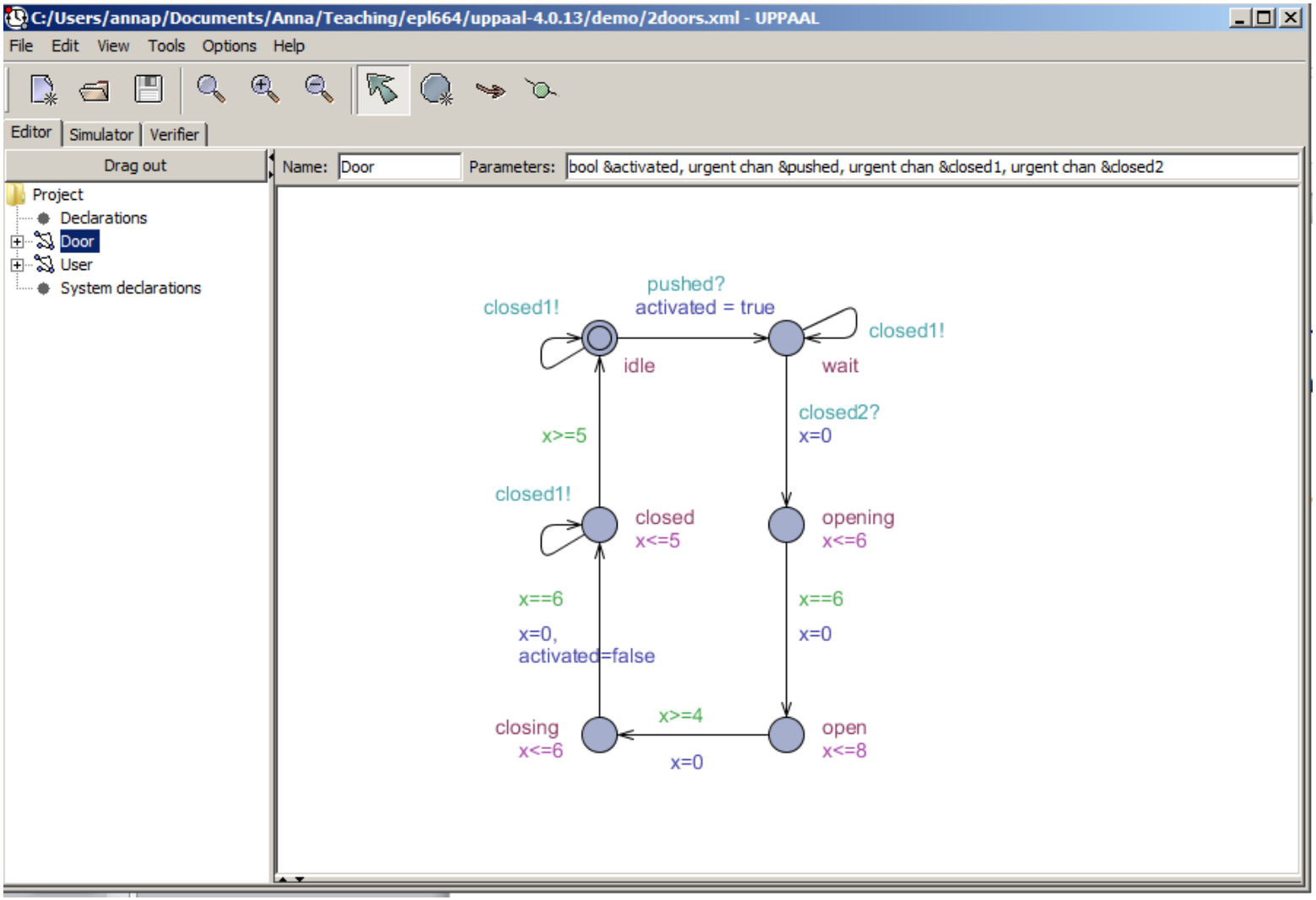
- `E1 ? E2 : E3  =  if E1 then E2 else E3`
- Εκφράσεις που χρησιμοποιούν ρολόγια είναι επιτρεπτές σε συνθήκες κατάστασης, φρουρούς μεταβάσεων και περιορισμούς. Η σύνταξή τους είναι περιορισμένη ανάλογα με την περίπτωση.

Κλάσεις Αυτομάτων

- Στην UPPAAL μπορούμε να ορίσουμε *κλάσεις* (template) χρονικών αυτομάτων.
- Κάθε κλάση αυτομάτου μπορεί
  - Να χρησιμοποιεί εκφράσεις για να ελέγχει και να ενημερώνει τις τιμές των ρολογιών, των μεταβλητών και των εγγραφών.
  - Να καλεί συναρτήσεις.
  - Να έχει τοπικές μεταβλητές και παραμέτρους.
- Παράμετροι στις κλάσεις μπορούν να δηλωθούν ως
  - Call by value (περνούμε την τιμή)
  - call-by-reference (περνούμε τη διεύθυνση)
    - Τα κανάλια και τα ρολόγια δηλώνονται πάντα ως παράμετροι call-by-reference

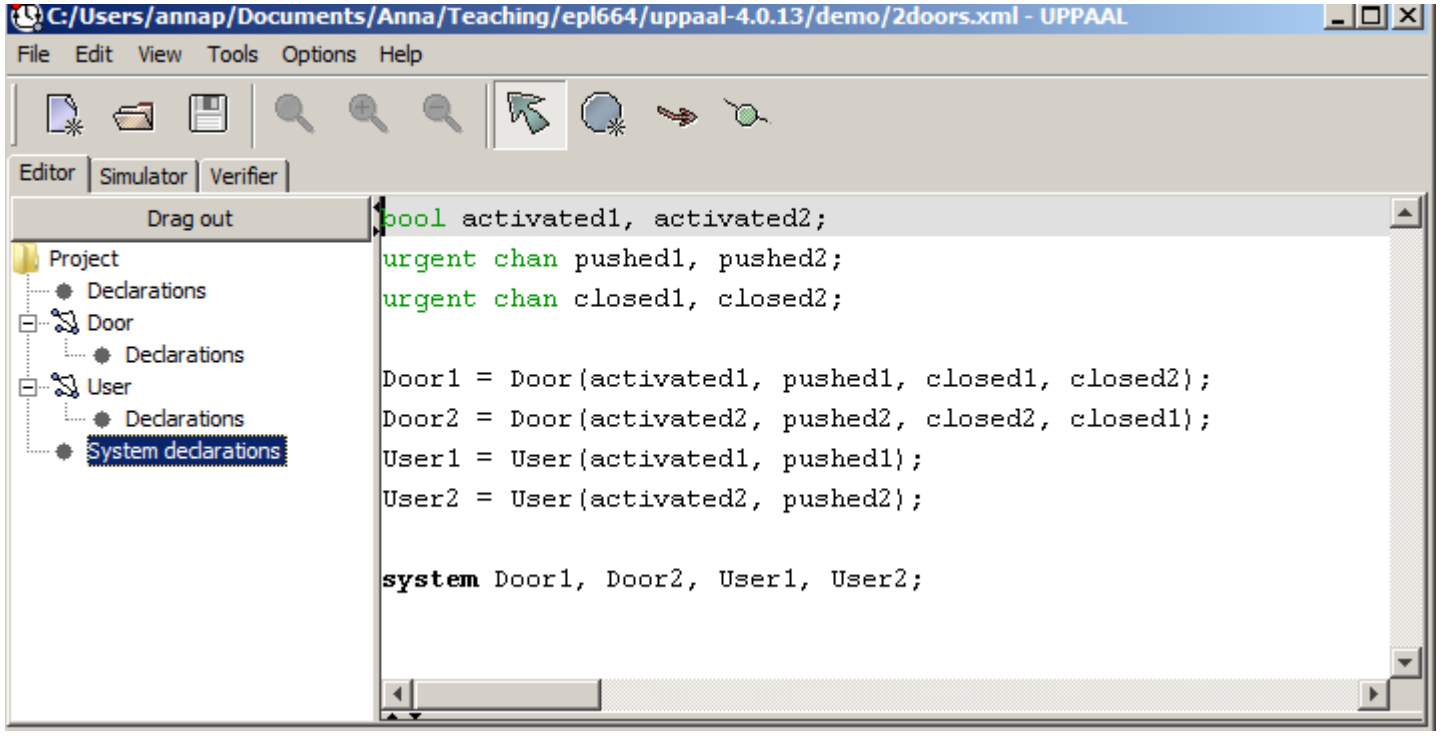
```
P(clock &x, bool bit, chan &ack)
```

Κλάσεις Αυτομάτων και Παράμετροι



Κλάσεις Αυτομάτων και Στιγμιότυπα

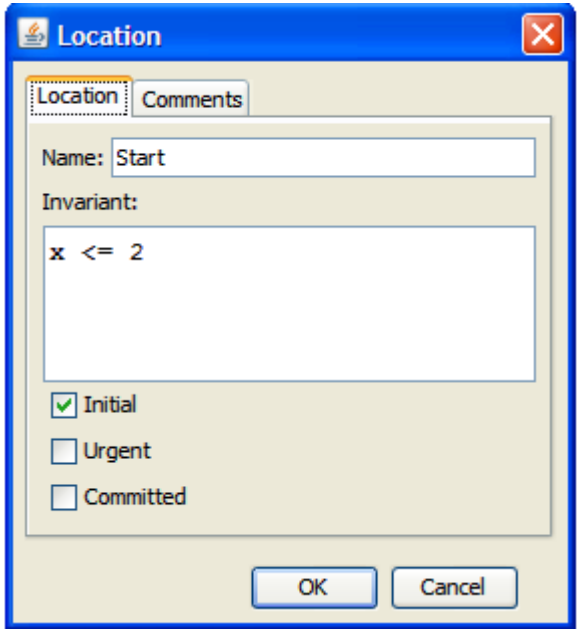
- *Στιγμιότυπα* μιας κλάσης αυτομάτου μπορούν να δημιουργηθούν στον ορισμό του συστήματος.



Αυτόματα στην Uppaal

- Αυτόματα στην Uppaal αποτελούνται από *τοποθεσίες* (locations) και *ακμές* (edges).

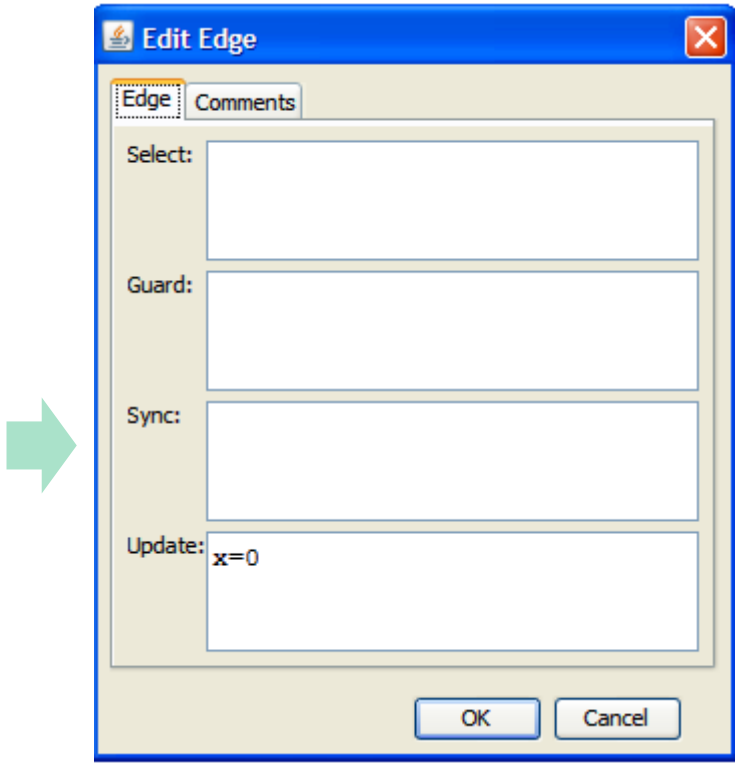
- Οι τοποθεσίες
  - αναπαριστούνται ως κύκλοι
  - μπορούν να έχουν κάποιο όνομα (name)
  - συσχετίζονται με ιδιότητες κατάστασης (invariant)



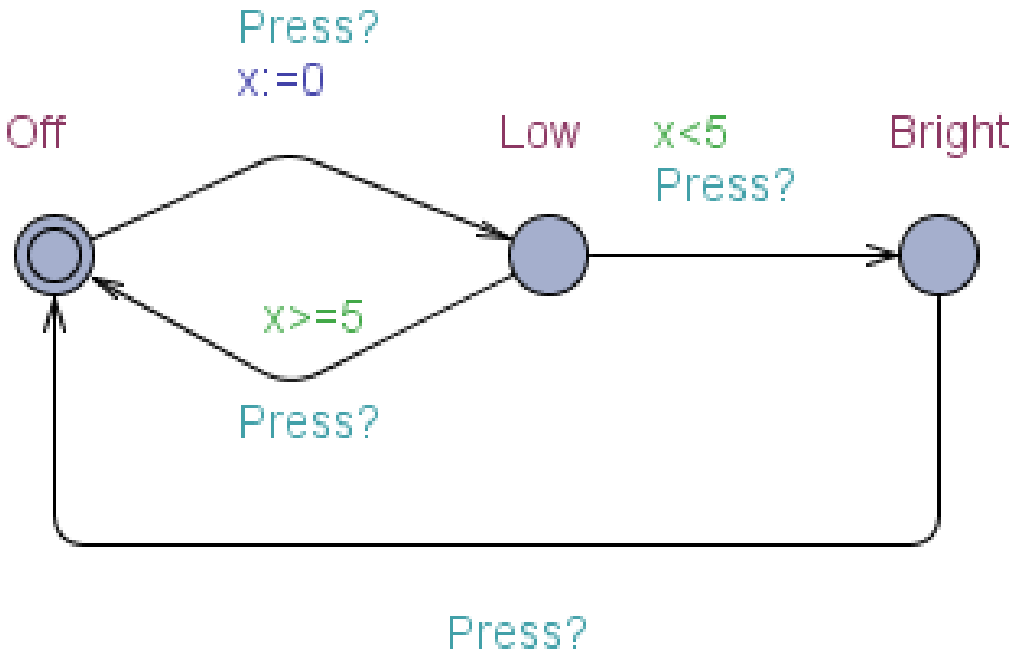
Αυτόματα στην Uppaal

Οι ακμές

- συνδέουν ζεύγη τοποθεσιών
- συσχετίζονται με
  - επιλογές (*selections*)
  - φρουρούς (*guards*),
  - συγχρονισμούς (*synchronisations*) και
  - ενημερώσεις (*updates*).



Παράδειγμα



Τοποθεσίες

- **Ιδιότητες κατάστασης** – ακολουθούν τη σύνταξη των εκφράσεων με τους πιο κάτω περιορισμούς:
  - Αποτελούν συζεύξεις απλών συνθηκών σε ρολόγια ή τη διαφορά της τιμής δύο ρολογιών
  - Λογικές εκφράσεις που δεν εμπεριέχουν ρολόγια
  - Τα όρια στις συνθήκες πρέπει να είναι εκφράσεις πάνω σε ακέραιες τιμές
  - Απαγορεύεται η χρήση κατώτατων συνθηκών σε ρολόγια.

- Οι πιο κάτω είναι έγκυρες ιδιότητες κατάστασης όπου τα x και y είναι ρολόγια και το A είναι ένας πίνακας τύπου int.
  - `x <= 2`
  - `x < y`
  - `(A[0]+1) != (A[1]*10)`

- **Αρχικές Τοποθεσίες**
  - Κάθε κλάση αυτομάτου πρέπει να έχει ακριβώς μια αρχική τοποθεσία
  - Η αρχική τοποθεσία σημειώνεται με διπλό κύκλο

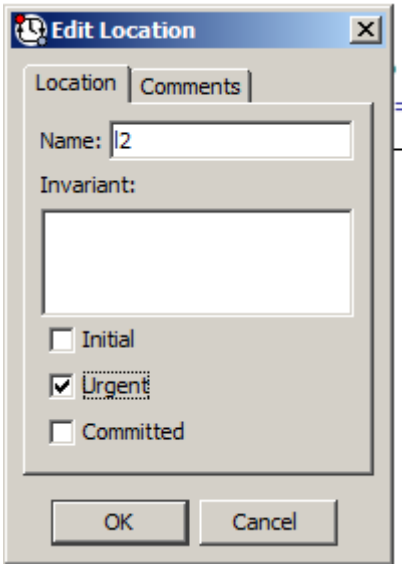
Τοποθεσίες

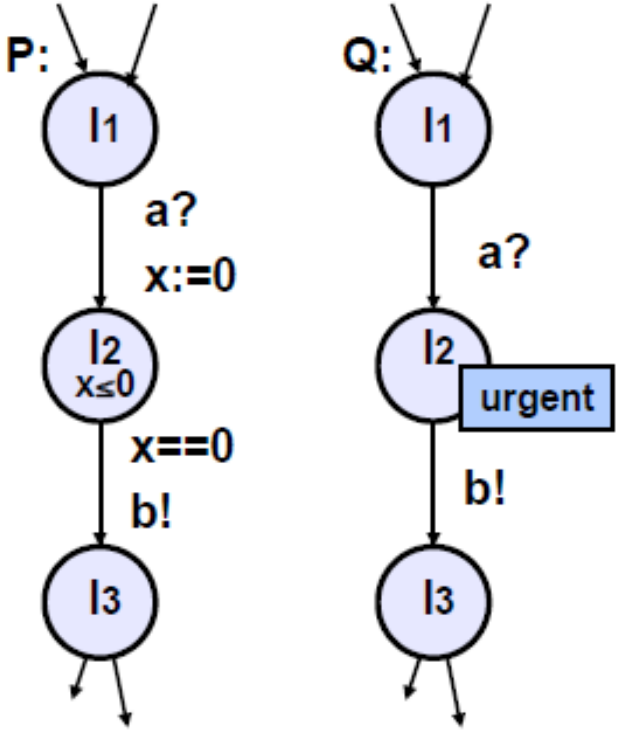
- **Κρίσιμες Τοποθεσίες (urgent locations)**
  - Παγώνουν τον χρόνο
  - Σημασιολογικά, αυτό είναι ισοδύναμο με:
    - Εισάγουμε ένα καινούριο ρολόι x το οποίο να μηδενίζεται σε κάθε ακμή εισόδου της κατάστασης καθώς και την ιδιότητα κατάστασης $x <= 0$.
- **Δεσμευμένες Τοποθεσίες (Committed locations)**
  - Παγώνουν τον χρόνο
  - Αν κάποιο αυτόματο βρίσκεται σε δεσμευμένη τοποθεσία, η επόμενη μετάβαση πρέπει να χρησιμοποιεί κάποια ακμή από κάποια δεσμευμένη τοποθεσία του συστήματος.
- Οι δεσμευμένες τοποθεσίες χρησιμοποιούνται για τη δημιουργία ατομικών ακολουθιών μεταβάσεων και για να διασφαλίζουν συγχρονισμό ανάμεσα σε περισσότερες από 2 οντότητες.

Κρίσιμες Τοποθεσίες - Παράδειγμα

- Για να εξαναγκάσουμε τη μετάβαση στο l2 να γίνει άμεσα:
  - Χρησιμοποιούμε ρολόγια, ή
  - Δηλώνουμε την τοποθεσία ως κρίσιμη

- Τα P και Q έχουν την ίδια συμπεριφορά.

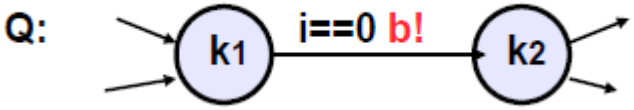


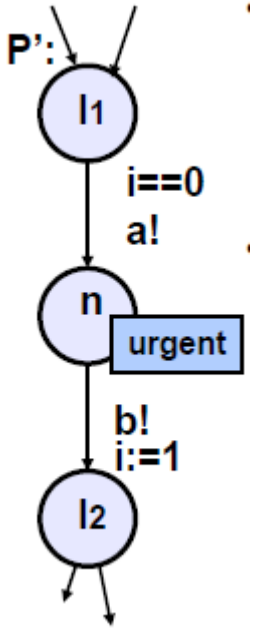


Δεσμευμένες Τοποθεσίες – Παράδειγμα

- Έστω ότι το αυτόματο P' θέλει να στείλει ταυτόχρονα τα μηνύματα a και b σε δύο διεργασίες.

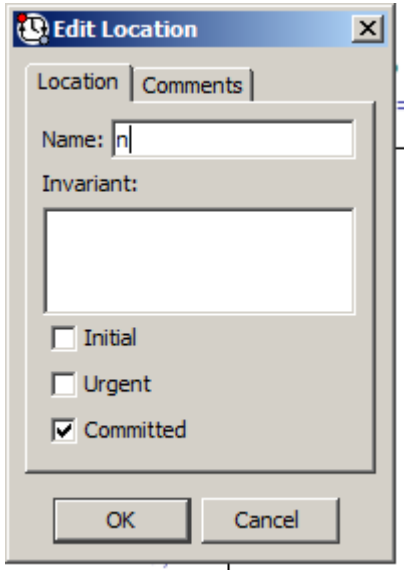
- Η δήλωση της τοποθεσίας n ως κρίσιμη έχει σαν αποτέλεσμα το πάγωμα του χρόνου στην τοποθεσία χωρίς όμως να αποτρέπει την παρεμβολή άλλων διεργασιών, π.χ. το Q :

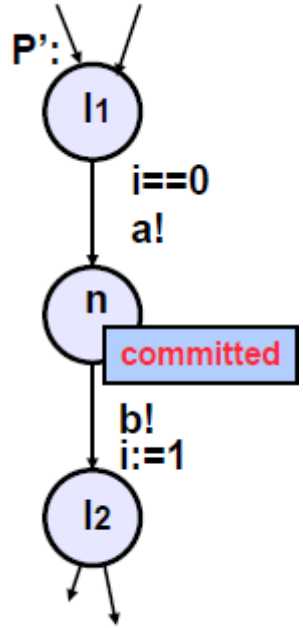




Δεσμευμένες Τοποθεσίες – Παράδειγμα

- Αν όμως δηλώσουμε την n ως δεσμευμένη αποτρέπει την παρεμβολή οποιασδήποτε άλλης διεργασίας σε αυτό το σημείο.





Ακμές και ενημερώσεις

- **Ενημερώσεις/Updates**: Εκφράσεις που εκτελούνται κατά την εκτέλεση μιας μετάβασης

- Παραδείγματα

```
x = 0
j = ( i[1]>i[2] ? i[1] : i[2] )
x = 1, y=2*x
```

- Οι εκφράσεις εκτελούνται σειριακά, π.χ. στην ενημέρωση

```
x = 1, y=2*x
```

θα εκτελεστεί πρώτα το `x = 1` και μετά το `y=2*x`

Ακμές και επιλογές, φρουροί

- **Επιλογές/Selections**: Δεσμεύουν μη ντετερμινιστικά μια μεταβλητή σε κάποια τιμή από ένα πεδίο.
  - `select: i : int[0,3]`

- **Φρουροί/Guards**: Διατυπώνουν τις συνθήκες κάτω από τις οποίες η μετάβαση είναι εκτελέσιμη.
  - Αποτελούν συζεύξεις απλών συνθηκών σε ρολόγια ή τη διαφορά της τιμής δύο ρολογιών
  - Λογικές εκφράσεις που δεν εμπεριέχουν ρολόγια
  - Τα όρια στις συνθήκες πρέπει να είναι εκφράσεις πάνω σε ακέραιες τιμές
  - Παράδειγμα:
    - `x >= 1 && x <= 2`
    - `(i[0]+1) != (i[1]*10)`

Ακμές και συγχρονισμοί

- **Συγχρονισμοί/Synchronisation**: Περιέχουν το όνομα μιας ενέργειας πάνω στην οποία η μετάβαση οφείλει να συγχρονιστεί με κάποιο άλλο αυτόματο.
  - Δύο είδη ενεργειών: e? και e!
  - Δυο αυτόματα συγχρονίζονται όταν έχουν συμπληρωματικές ενέργειες συγχρονισμού.
  - Οι ενέργειες e1? και e2! είναι συμπληρωματικές αν οι εκφράσεις e1 και e2 αναφέρονται στο ίδιο κανάλι.

- Όταν δύο αυτόματα συγχρονίζονται
  - Οι μεταβάσεις τους εκτελούνται ταυτόχρονα
  - Οι ενημερώσεις της ακμής που εκτελεί την ενέργεια e1! υλοποιούνται πριν από τις ενημερώσεις της ακμής που εκτελεί την ενέργεια e2?.

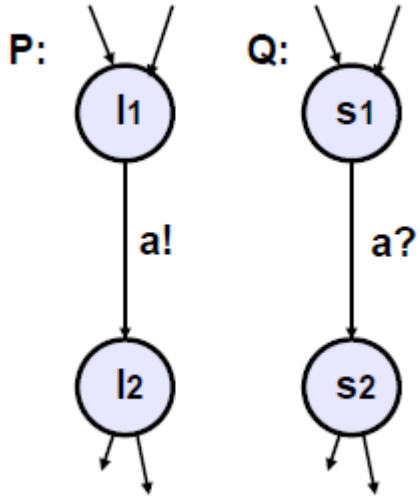
Ακμές και συγχρονισμοί

- **Κρίσιμα κανάλια (urgent channels)**: Ο συγχρονισμός πάνω σε κρίσιμα κανάλια δεν μπορεί να καθυστερήσει.
  - Δεν επιτρέπεται να ορίζουμε φρουρούς με ρολόγια σε ακμές που χρησιμοποιούν κρίσιμα κανάλια

- **Κανάλια εκπομπής (broadcast channels)**: Επιτρέπουν συγχρονισμούς του τύπου 1-to-many.
  - Όταν εκτελείται η ενέργεια e! και το e είναι κανάλι εκπομπής, τότε όλες οι διαθέσιμες μεταβάσεις με ενέργεια e? θα συγχρονιστούν μαζί του.
  - Δεν είναι απαραίτητο να υπάρχει διαθέσιμη ενέργεια e?.
  - Δεν επιτρέπεται να ορίζουμε φρουρούς με ρολόγια σε ακμές που περιμένουν κάποιο μήνυμα σε κανάλι εκπομπής.

Κρίσιμα Κανάλια – Παράδειγμα 1

- Έστω ότι δύο διεργασίες μπορούν να συγχρονιστούν μέσω ενός καναλιού και επιθυμούμε ο συγχρονισμός να πραγματοποιηθεί μόλις και οι δυο διεργασίες είναι έτοιμες.



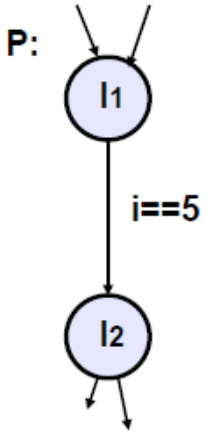
- Είναι δυνατόν να εκφράσουμε αυτή τη συμπεριφορά χρησιμοποιώντας ιδιότητες κατάστασης;
  - Όχι. Για παράδειγμα στην τοποθεσία l1 δεν είναι δυνατό να ξέρουμε πότε η Q θα φτάσει στην τοποθεσία s1.

- Λύση: δηλώνουμε το κανάλι a ως κρίσιμο κανάλι.

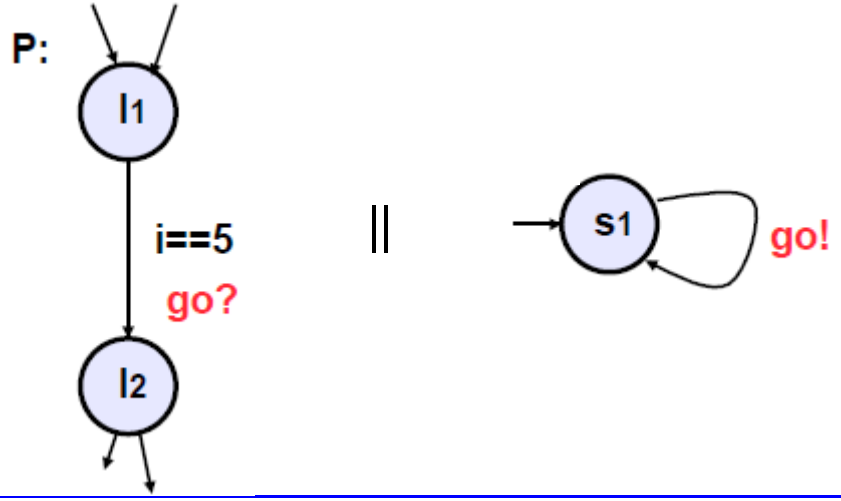
urgent chan a;

Κρίσιμα Κανάλια – Παράδειγμα 2

- Τα κρίσιμα κανάλια επίσης μας διευκολύνουν να ορίζουμε μεταβάσεις που πρέπει να εκτελεστούν αμέσως μόλις ο φρουρός τους γίνει true.



- Έστω ότι θέλουμε η μετάβαση από το l1 στο l2 να συμβεί μόλις i==5 (i μεταβλητή δεδομένων).
- Δημιουργούμε ένα καινούριο κρίσιμο κανάλι και μια παράλληλη διεργασία έτοιμη να συγχρονιστεί σε αυτό.



Συστήματα

- **Σύστημα:** ορίζει το μοντέλο ενός συστήματος
- Αποτελείται από
  - Ένα ή περισσότερα αυτόματα
  - Τοπικές και καθολικές μεταβλητές
  - Κανάλια
- Οι δηλώσεις μεταβλητών και καναλιών δεν είναι άμεσα προσβάσιμες από τις κλάσεις αυτομάτων αφού ορίζονται μετά από αυτές.
- Χρησιμεύουν για να δώσουν τιμές στις τυπικές παραμέτρους των κλάσεων.
- Λέξη κλειδί system: χρησιμοποιείται για να δημιουργήσει τα αυτόματα που απαρτίζουν το σύστημα

System Declaration – Παράδειγμα

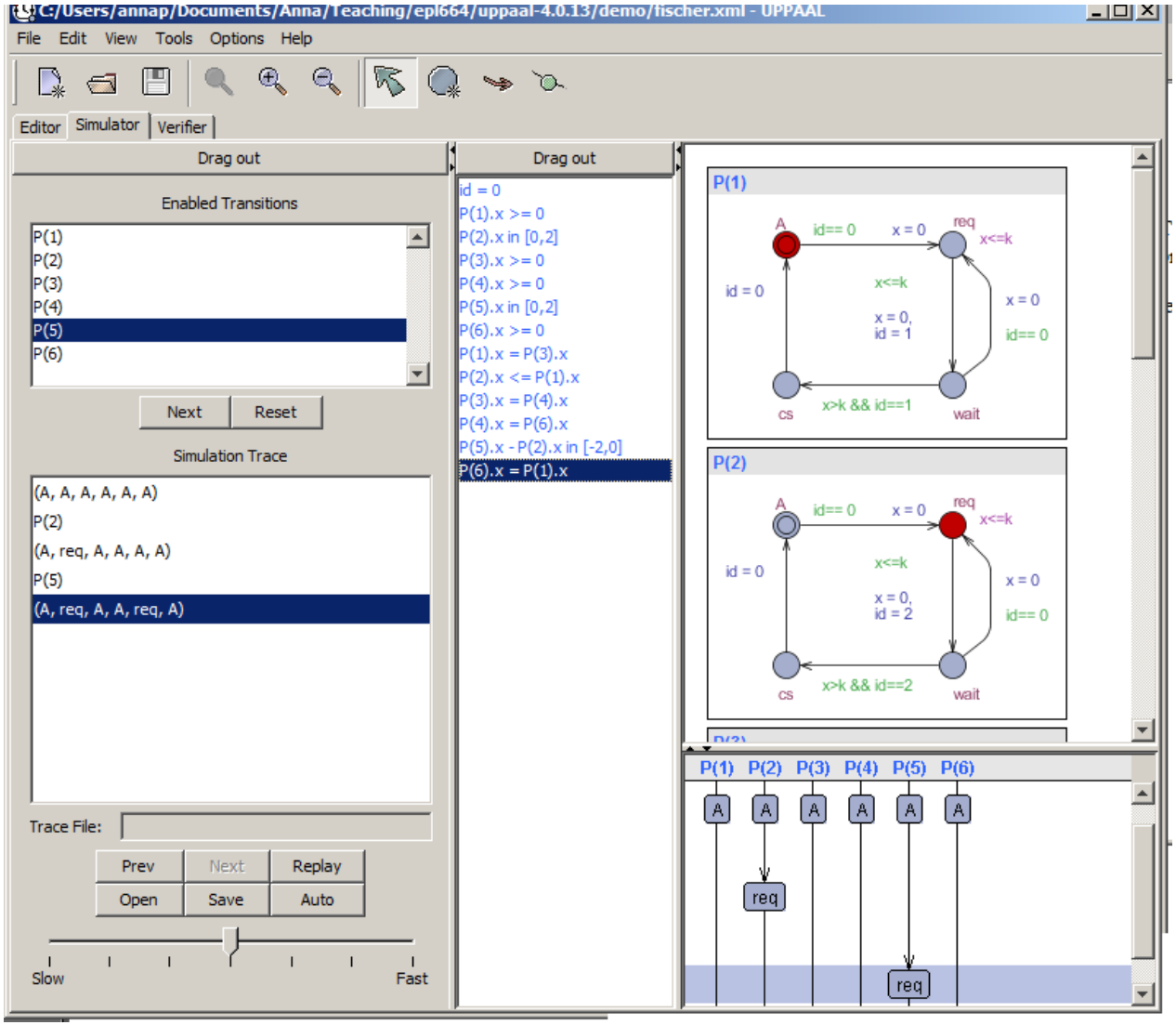
```
const int fastest = 5;
const int fast    = 10;
const int slow    = 20;
const int slowest = 25;

Viking1 = Soldier(fastest);
Viking2 = Soldier(fast);
Viking3 = Soldier(slow);
Viking4 = Soldier(slowest);

system Viking1, Viking2, Viking3, Viking4, Torch;
```

Ο Προσομοιωτής

- Ο προσομοιωτής (simulator) του εργαλείου UPPAAL επιτρέπει τη διενέργεια προσομοιώσεων ενός μοντέλου.

- Επιτρέπει
  - Παρακολούθηση συμπεριφορών καθώς και
  - Μετακίνηση μπρος και πίσω πάνω σε μία εκτέλεση
  - Παρακολούθηση των τιμών των μεταβλητών
  - Παρακολούθηση της επικοινωνίας ανάμεσα στις διάφορες συνιστώσεις μέσω Message Sequence Chart.

Ο Προσομοιωτής



Ο Επαληθευτής

- Ο Επαληθευτής (verifier) επιτρέπει το μοντελοέλεγχο ιδιοτήτων ασφάλειας και ζωτικότητας διατυπωμένων σε μια χρονική εκδοχή της χρονικής λογικής CTL.

- Γλώσσα προδιαγραφής ιδιοτήτων:

```
φ ::= A[] Expression | E<> Expression | E[] Expression
    | A<> Expression | Expression --> Expression
```

- Expression είναι οποιαδήποτε νόμιμη έκφραση της γλώσσας.

- Επιπλέον ως έκφραση μπορούμε να γράψουμε την
  process.location
  της οποίας η τιμή γίνεται true όταν η διεργασία process βρίσκεται στην τοποθεσία location.

- Σημείωση: δεν επιτρέπεται το φώλιασμα των χρονικών τελεστών.

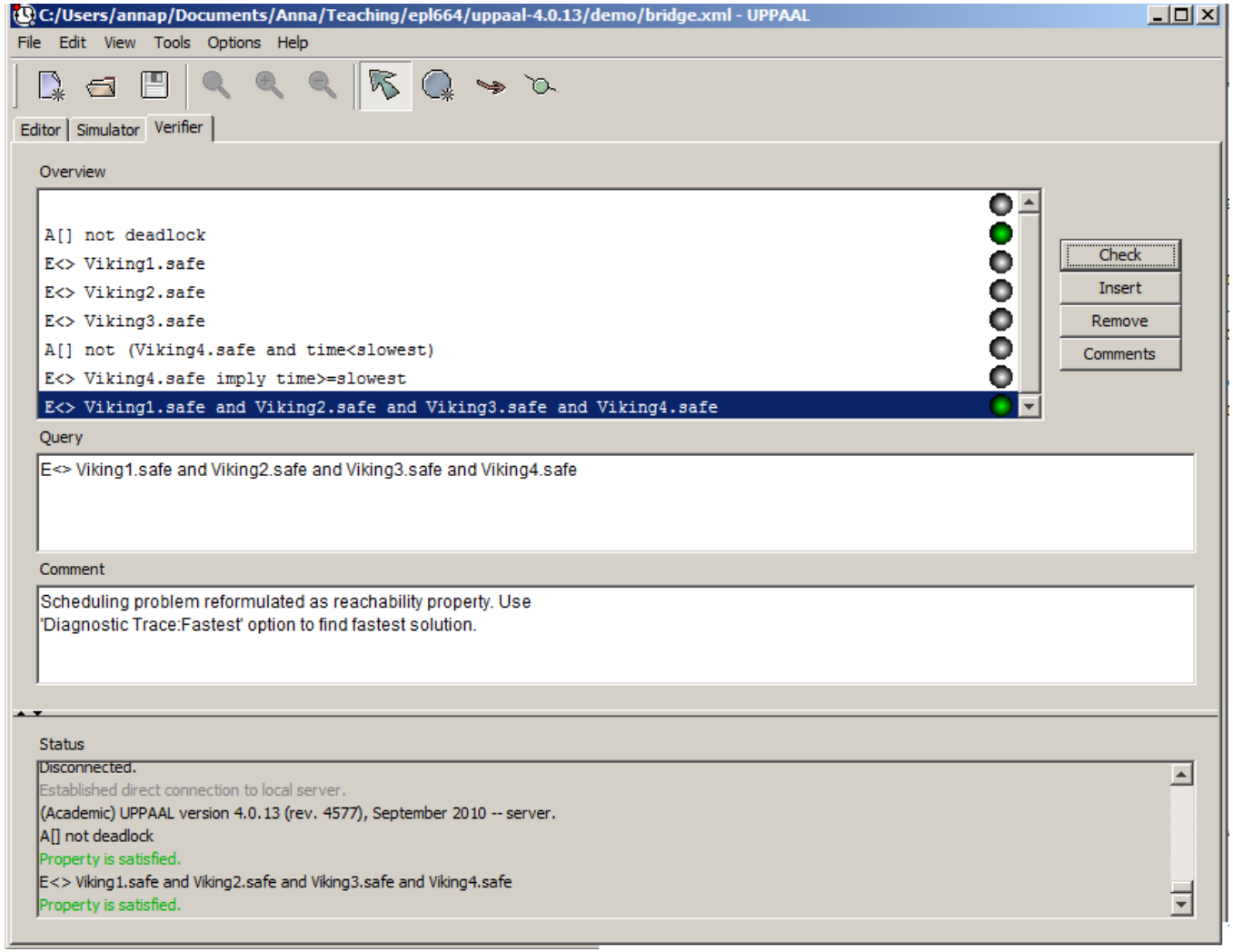
Επεξήγηση Τελεστών

- `A[] φ ≡ AG φ` Σε κάθε εκτέλεση και κάθε κατάσταση αυτής ισχύει το φ
- `E<> φ ≡ EF φ` Υπάρχει εκτέλεση όπου στο μέλλον φ
- `E[] φ ≡ EG φ` Υπάρχει εκτέλεση όπου σε κάθε κατάσταση αυτής ισχύει το φ
- `A<> φ ≡ AF φ` Σε κάθε εκτέλεση στο μέλλον φ
- `φ1 --> φ2 ≡ AG(φ1 → AF φ2)` Σε κάθε εκτέλεση και πάντοτε αν φ1 τότε σε κάθε εκτέλεση στο μέλλον φ2

Παραδείγματα

- `E<> (p1.cs and p2.cs)`
- `A[] (p1.cs imply not p2.cs)`
- `A[] not deadlock`
- `A<> (P1.Landed and P2.Landed)`
- `A[] [(P1.Landed and P2.Landed) imply (y>=3 or z>=3) ]`

Ο Επαληθευτής



Case Study – Ο Αλγόριθμος του Fischer

- Αμοιβαίος Αποκλεισμός
- Τυπική Δομή Αλγόριθμου

```
while true do
begin
    remainder region
    trying region
    critical region
    exit region
end
```

- Στόχος
  - Ανά πάσα στιγμή το πολύ μια διεργασία μπορεί να βρίσκεται στην κρίσιμή της περιοχή
  - Απουσία Αδιεξόδων

Impossibility Results

- Γνωστοί αλγόριθμοι για n διεργασίες
  - Απαιτούν $O(n)$ καταχωρητές και $O(n)$ πράξεις πάνω σε αυτές
  - Μη πρακτικό για εφαρμογές μεγάλου μεγέθους
- Υπάρχει καλύτερος αλγόριθμος;
- Θεώρημα [Burns/Lynch 1996] Δεν υπάρχει ασύγχρονος αλγόριθμος για διασφάλιση αμοιβαίου αποκλεισμού ανάμεσα σε $n \geq 2$ διεργασίες που να χρησιμοποιεί λιγότερους από n καταχωρητές.
- Impossibility results
  - Θέτουν ακριβή όρια στο τι είναι ή όχι εφικτό στα πλαίσια ενός υπολογιστικού μοντέλου
  - Διασαφηνίζουν τις συνθήκες κάτω από τις οποίες υπάρχουν τα όρια
    - Οι συνθήκες ικανοποιούνται σε διαφορετικά σενάρια;
    - Τι συμβαίνει σε υπολογιστικά μοντέλα που ικανοποιούν διαφορετικές ιδιότητες;

Χρονικός Αλγόριθμος Αμοιβαίου Αποκλεισμού

- Ο Αλγόριθμος του Fischer: ο πρώτος αλγόριθμος που «έσπασε» το κάτω φράγμα των O(n) καταχωρητών.

- Χρησιμοποιεί *ένα* κοινό καταχωρητή, id, με αρχική τιμή 0.

- Επιτρέπει στις διεργασίες, να καθυστερήσουν για κάποιο χρόνο delay

Ο Αλγόριθμος του Fischer

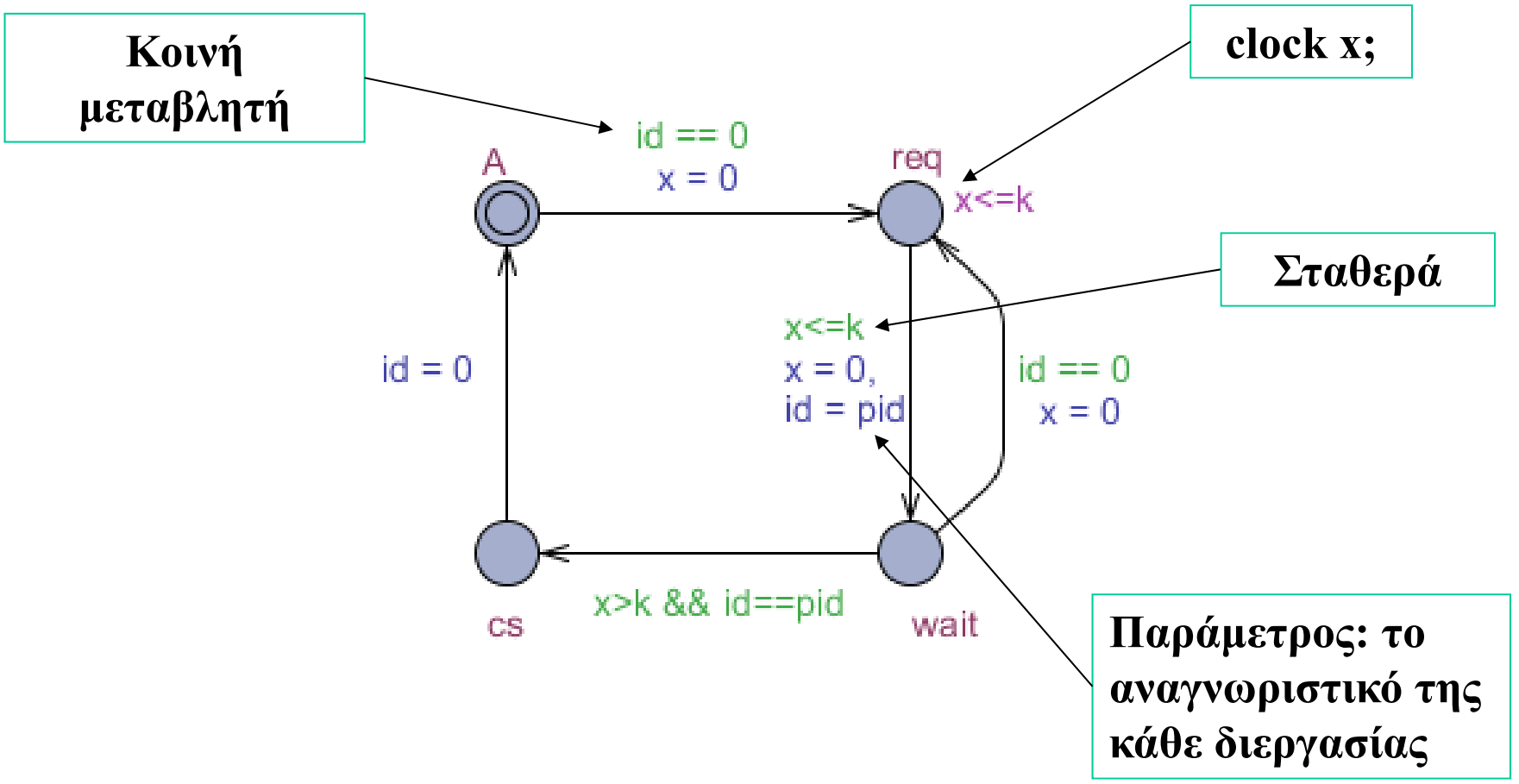
- Κώδικας για τη διεργασία με αναγνωριστικό i:

```
while true do
begin
    'nocritical section';
    L: if id != 0 then goto L;
    1: id = i;
    2: pause (delay);
    3: if id != i then goto L;
    'critical section';
    id := 0;
end
```

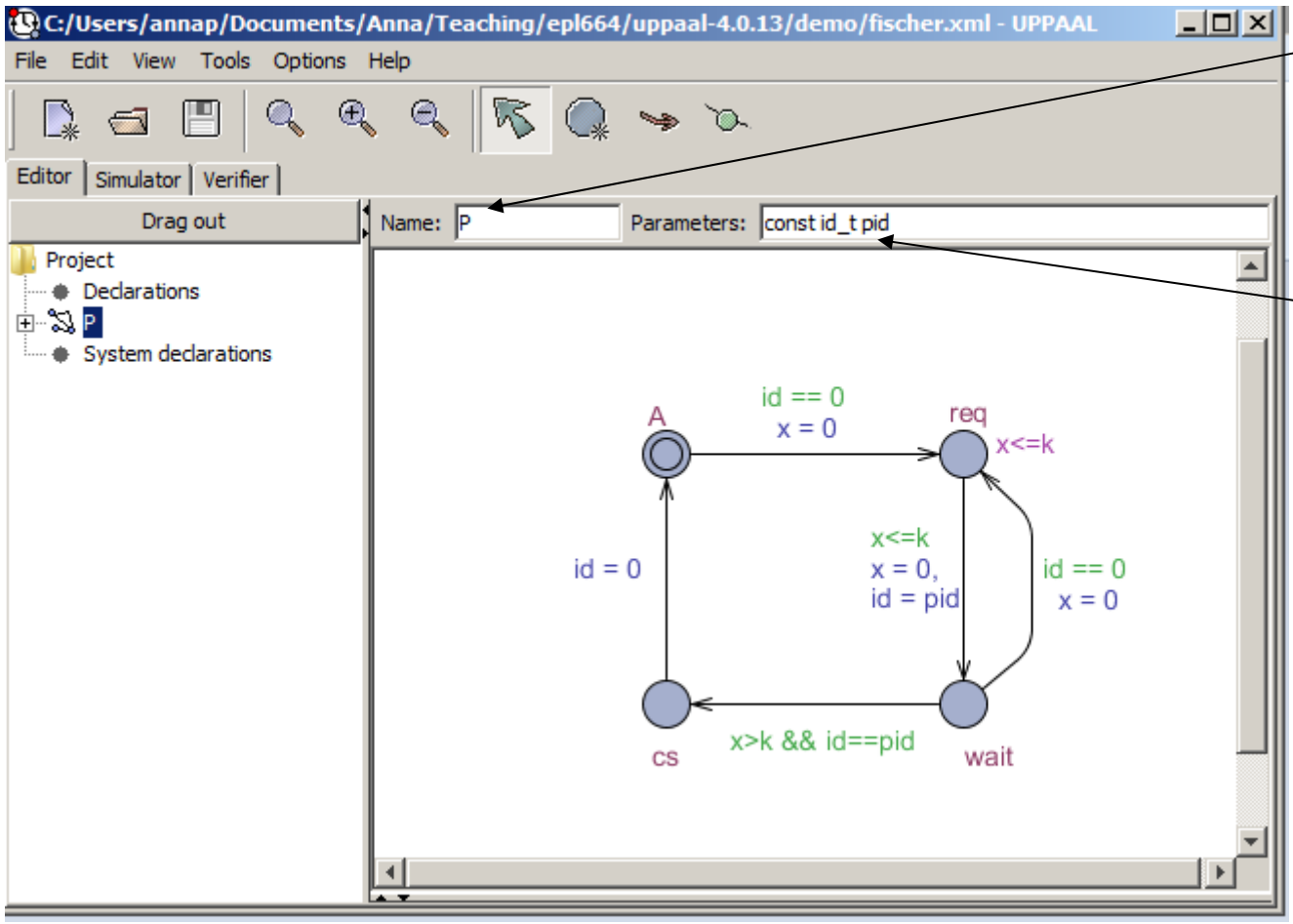
Η καθυστέρηση delay

- Ποια είναι η κατάλληλη επιλογή για την καθυστέρηση delay;
- Έστω k ο μέγιστος χρόνος οποιουδήποτε βήματος και οποιασδήποτε διεργασίας που συμμετέχει στον αλγόριθμο.
- Θέτουμε delay > k.
- Επεξήγηση Επιλογής: Μέχρι τη στιγμή που μια διεργασία φτάσει στο βήμα 3, όλες οι διεργασίες που πέρασαν τον έλεγχο στο βήμα L έχουν εκτελέσει και το βήμα 2 αφού η καθυστέρηση είναι μεγαλύτερη από τον μέγιστο χρόνο που θα μπορούσε να καθυστερήσει αυτό το βήμα.
- Επομένως, αν στο βήμα 3 μια διεργασία βρει το id να έχει ως τιμή την τιμή του αναγνωριστικού της, μπορεί να προχωρήσει στο κρίσιμό της τμήμα ως η διεργασία που υπερισχύει έχοντας γράψει τελευταία την τιμής της στον καταχωρητή.

Μοντελοποίηση του Αλγόριθμου



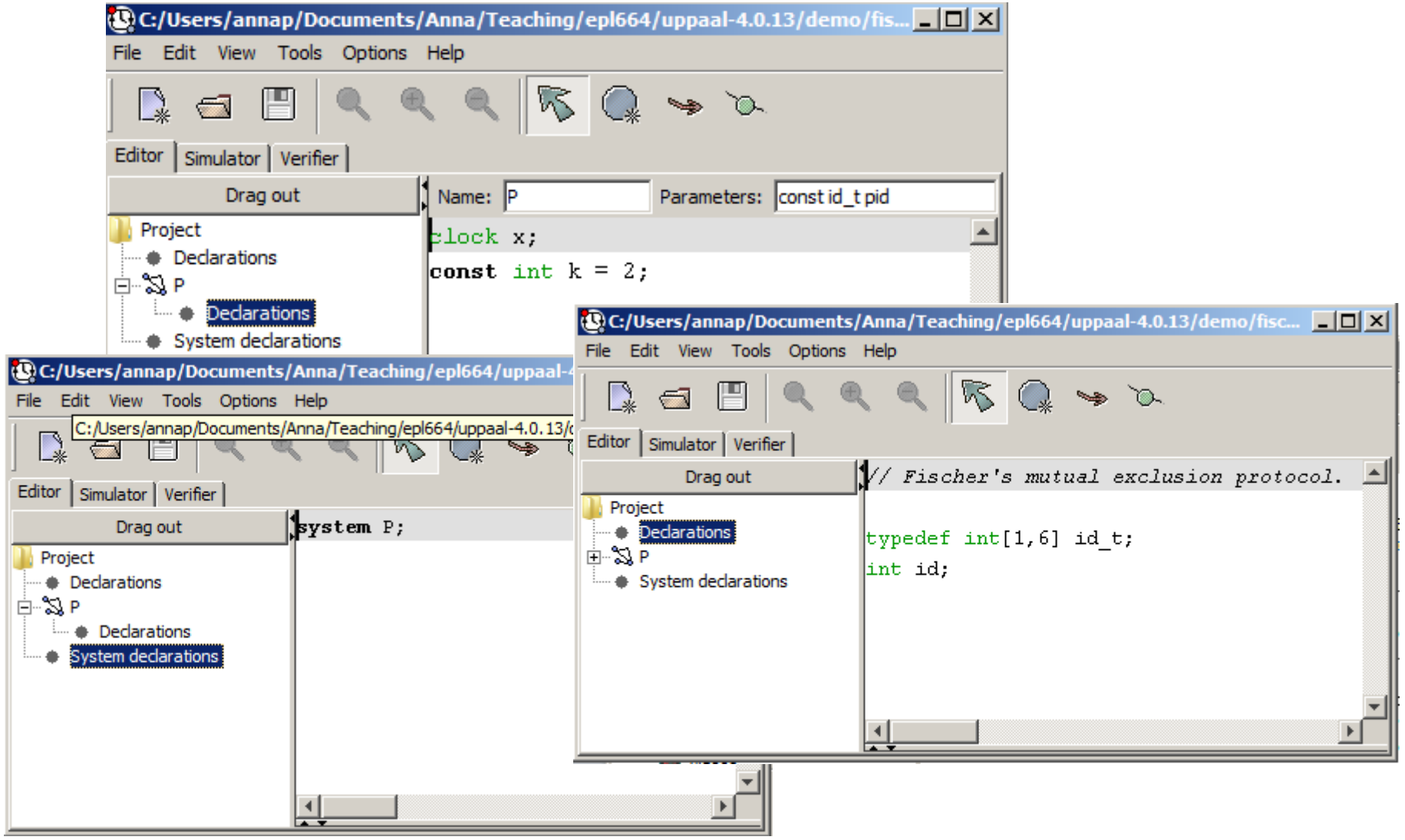
Δήλωση Κλάσης



Όνομα κλάσης

Παράμετροι κλάσης

Καθολικές και Τοπικές Δηλώσεις



Ανάλυση του Αλγόριθμου

- Αμοιβαίος Αποκλεισμός:

```
A[] forall (i:id_t)
     forall (j:id_t)
P(i).cs && P(j).cs imply i == j
```

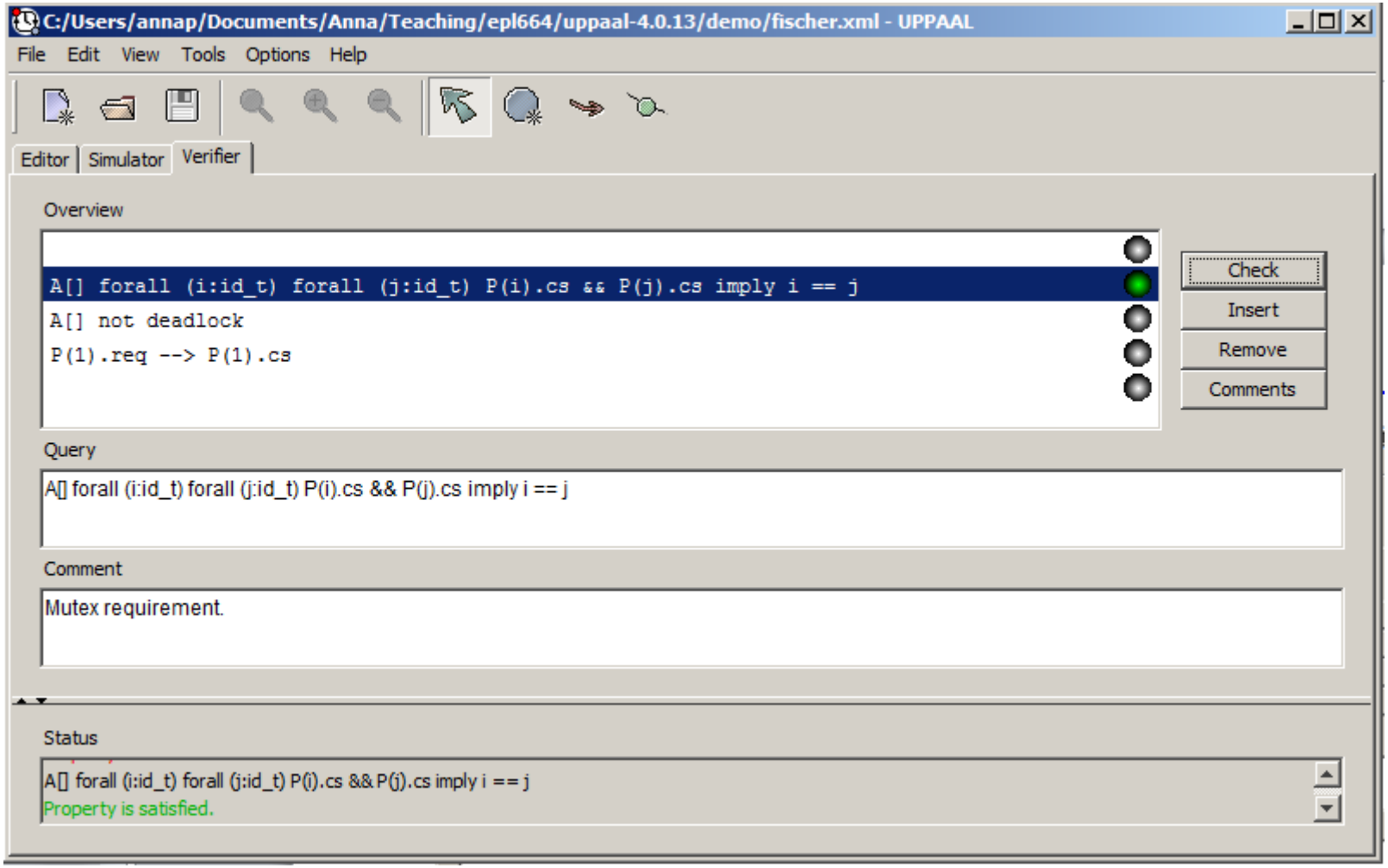
- Απουσία Αδιεξόδων:

```
A[] not deadlock
```

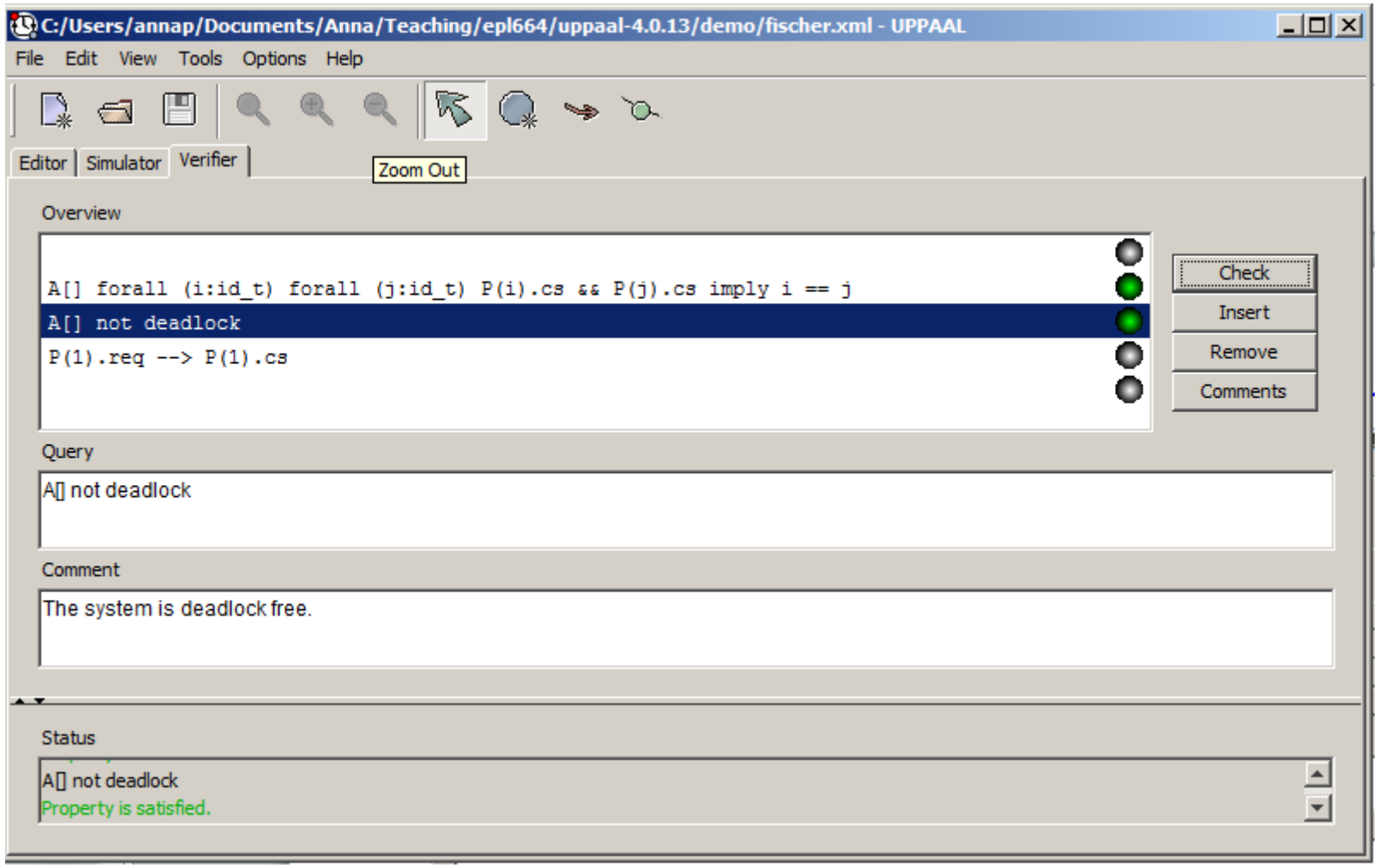
- Παρατεταμένη στέρηση:

```
P(1).req --> P(1).cs
```

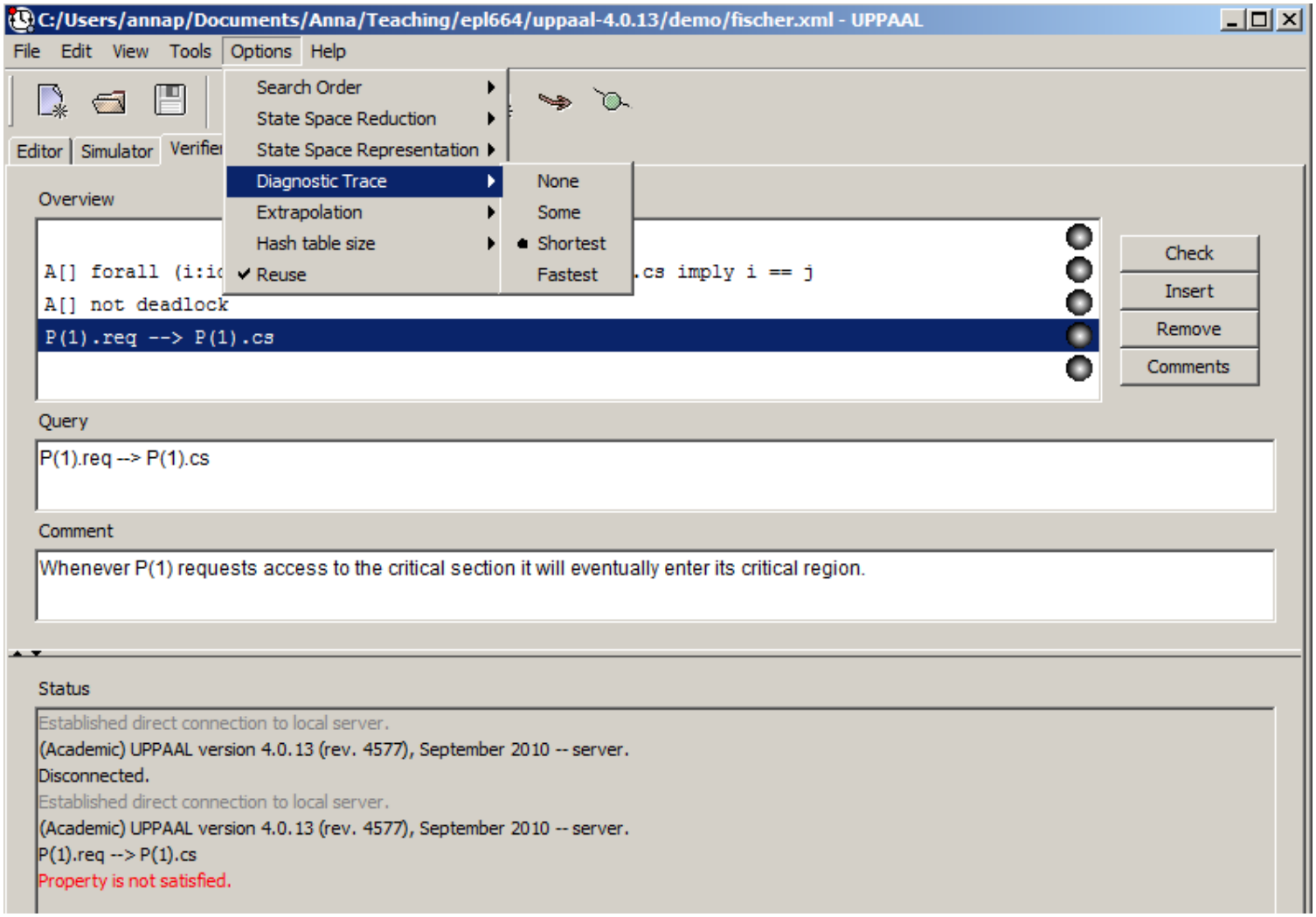
Μοντελοέλεγχος Ιδιοτήτων (1)



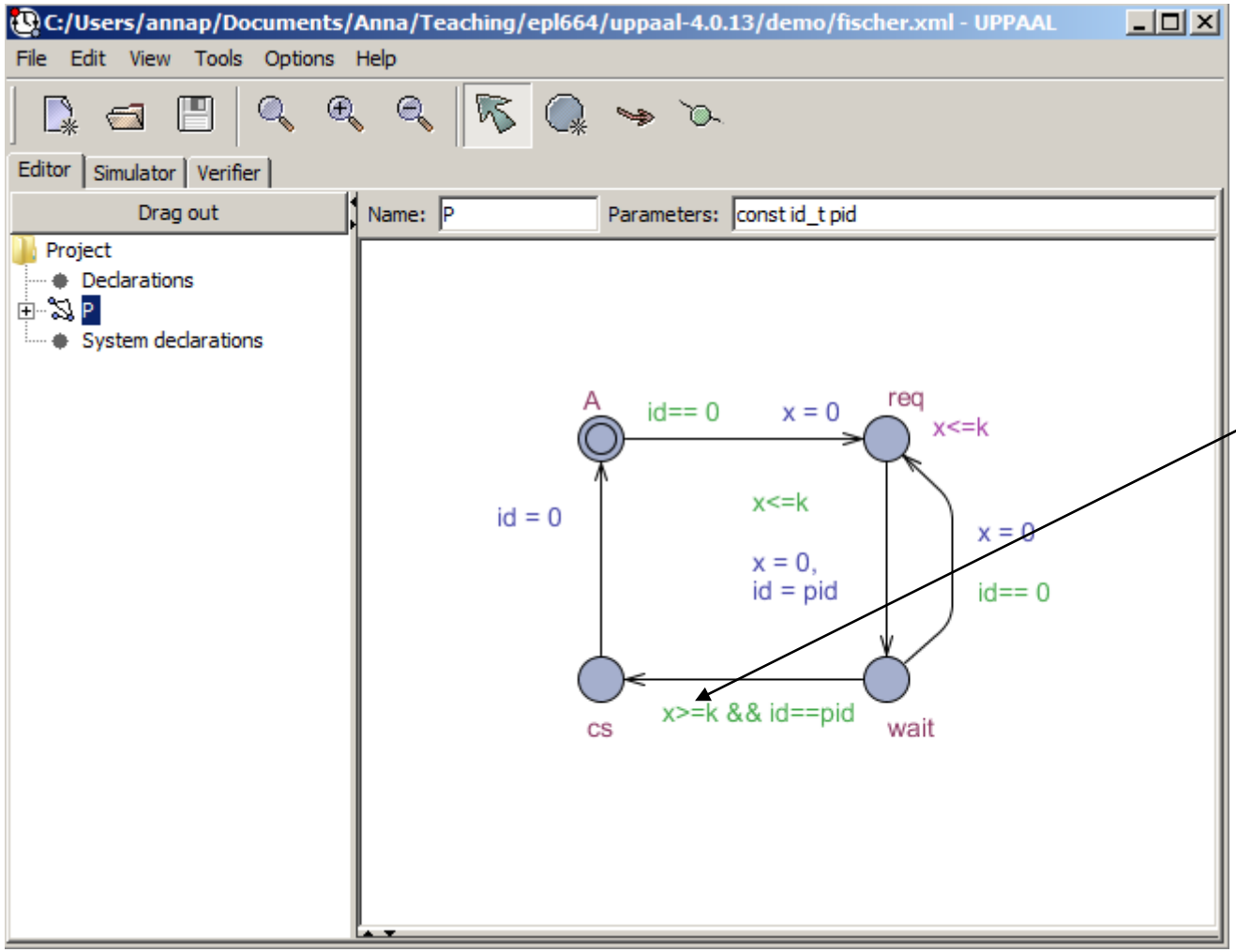
Μοντελοέλεγχος Ιδιοτήτων (2)



Μοντελοέλεγχος Ιδιοτήτων (3)

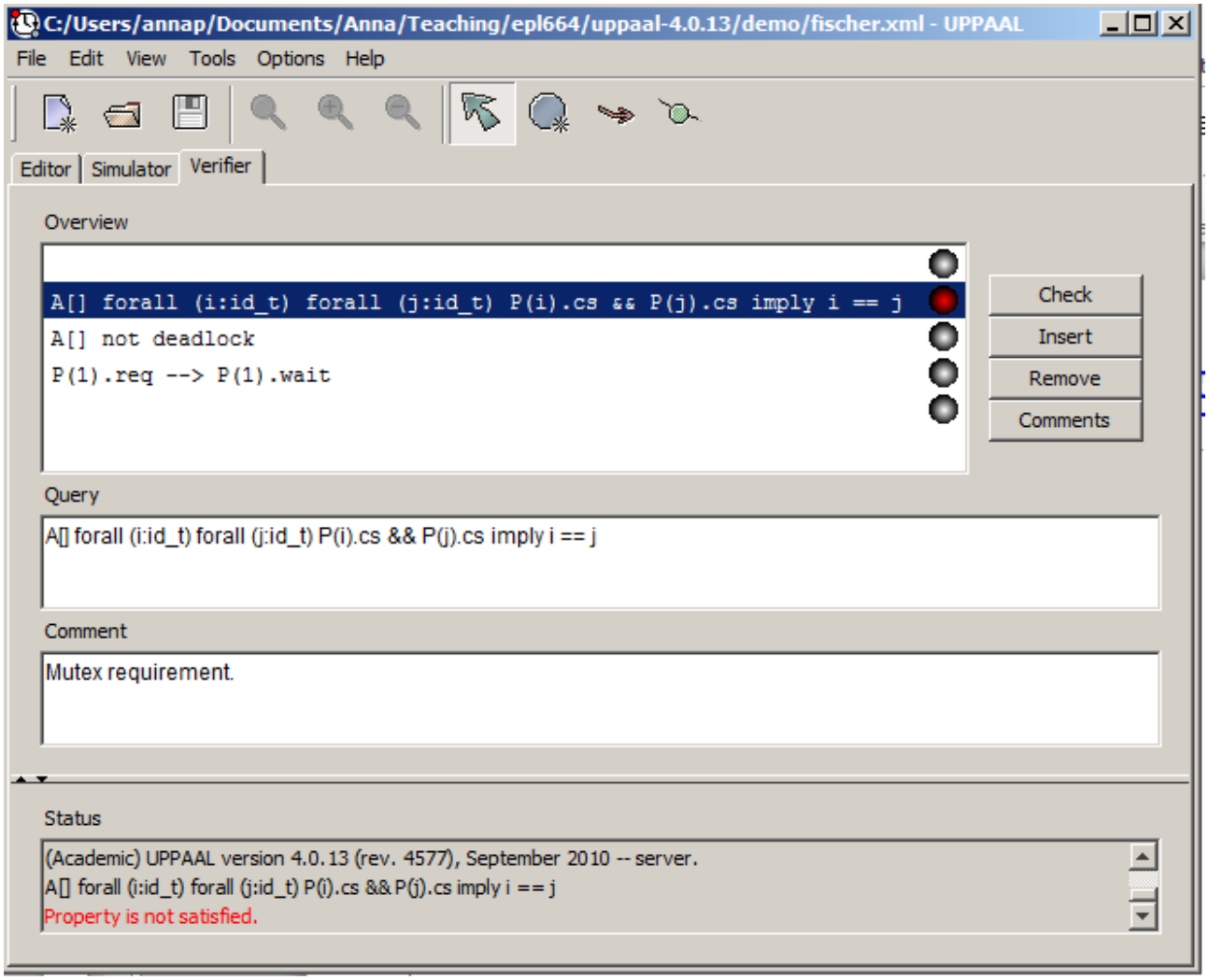


Παραλλαγή του μοντέλου



Τι θα γίνει αν μειωθεί η καθυστέρηση;

Παραβιάζεται ο αμοιβαίος αποκλεισμός



Βοηθητικό Υλικό

- http://www.uppaal.org
- http://www.it.uu.se/research/group/darts/papers/texts/new-tutorial.pdf
- http://www.it.uu.se/research/group/darts/papers/texts/lpw-sttt97.ps.gz
- Στο Help του εργαλείου!