**ΤΕΙ ΚΡΗΤΗΣ ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ**

**ΤΜΗΜΑ ΜΗΧΑΝΟΛΟΓΙΑΣ**


# ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

# «Παραγωγή κώδικα c από προδιαγραφές σε κατηγορηματικό λογισμικό»

Όνομα σπουδαστή: Ξυλούρης Ανδρέας

# Περιεχόμενα

≡

# 1 Εισαγωγή.

Σε αυτή την εργασία παρουσιάζουμε πως μετατρέπουμε λογικές προτάσεις σε γλώσσα υψηλού προγραμματισμού. Για να το πετύχουμε πρέπει να ορίσουμε μια θεωρητική περιγραφή αλγορίθμου. Δηλαδή θα βρίσκουμε ένα τρόπο να παίρνουμε το κάθε τμήμα της λογικής πρότασης και με βάση τα βήματα του αλγόριθμου να μετατρέπουμε την πρόταση σε γλώσσα υψηλού προγραμματισμού.Η γλώσσα που χρησιμοποιούμε είναι η C.Επίσης προσπαθούμε να δείξουμε ότι η μετατροπή μέσω θεωρητικής περιγραφής αλγόριθμου λειτουργεί. Για αυτό αποδεικνύουμε την εγκυρότητα του αλγορίθμου μας. Μετά θα δείξουμε πως κατασκευάζονται διαγράμματα που μας δείχνουν συνοπτικά πως λειτουργούνε τα προγράμματα που φτιάξαμε. Τέλος θα παρουσιάσουμε κάποια παραδείγματα για να δούμε πως δουλεύουν στην πράξη.

# 2. Θεωρητική περιγραφή αλγορίθμου μετατροπής λογικών προτάσεων σε C.

Εξετάζουμε τον αλγόριθμο βήμα βήμα :

**Πρώτο βήμα.**

Τα μέρη της λογικής πρότασης ή τις ενέργειες μιας λογικής πρότασης τις μετατρέπουμε σε ακεραίους που παίρνουν τις τιμές 0 ή 1.

Παράδειγμά.

Εάν έχουμε ένα κύκλωμα με δύο διακόπτες στην σειρά όταν είναι και οι δύο πατημένοι τότε ο λαμπτήρας είναι αναμμένος. Σε άλλη περίπτωση είναι κλειστός .Έστω ότι ο πρώτος διακόπτης s1 και ο δεύτερος s2 και lamp ο λαμπτήρας του κυκλώματος. Έχω την λογική πρόταση ups1 ^ ups2 > light. Τα μέρη της λογικής πρότασης είναι αυτά που πήραμε προηγουμένως δηλαδή ο διακόπτης 1 ο διακόπτης2 και ο λαμπτήρας .Άρα με το πρώτο βήμα του αλγορίθμου μας θα έχουμε

Int s1;

Int s2;

Int lamp;

Όπου οι ακέραιοι μας μπορούνε να πάρουνε τις τιμές 0 ή 1.Δηλαδή το up(s1) αντιστοιχεί στο s1=1 και το¬ up(s1) στο s1=0;Το ίδιο και με τα υπόλοιπα μέρη μιας λογικής πρότασης.

**Δεύτερο βήμα**

Όλα τα Λ τα μετατρέπουμε σε && και όλα τα V σε ||.

**Τρίτο βήμα.**

Όταν θέλουμε να μετατρέψουμε τα →, θα πρέπει να προσέξουμε να μην παραβιάζεται αυτό που δηλώνουν.

Ας δούμε κάποια παραδείγματα για το δεύτερο και το τρίτο βήμα. Έστω ότι έχω την λογική πρόταση :up(S1) ^Λup(s2) → light.Τότε θα έχουμε σύμφωνα με το δεύτερο και το τρίτο βήμα του αλγορίθμου:

```
If(s1==1 && s2==1){

   Light=1;

}
```

Εάν είχα την λογική πρόταση : light  up(s1) Λup(s2).τότε θα μπορούσαμε να γράψουμε το εξής:

```
If( light ==1 ) {

    S1=1;

    S2=1;

}
```

Σε αυτή την περίπτωση βλέπουμε ότι η λογική μας πρόταση ισχύει και προς τις δύο κατευθύνσεις αλλά έτσι όπως την εκφράζουμε παραπάνω δεν έχει και μεγάλη ωφελιμότητα. Άρα θα μπορούσαμε να την εκφράσουμε όπως και στο προηγούμενο παράδειγμα.

**Τέταρτο βήμα:**

Τα συμβάντα τις καταστάσεις δηλαδή της λογικής πρότασης τις μετατρέπουμε σε συναρτήσεις που μπορούνε να πάρουνε σαν ορίσματα άλλες ενέργειες ή μέρη της λογικής πρότασης αλλά και ένα επιπλέον όρισμα τον χρόνο .Οι συναρτήσεις αυτές επιστρέφουνε σαν 0 ή 1.

Για να καταλάβουμε ας δούμε ένα παράδειγμα:

Έχουμε το κύκλωμα του προηγούμενου παραδείγματος, όπου πλέον υπάρχει και ένας διακόπτης αναστροφής σε κάποια ορισμένα χρονικά σημεία. Έχω την λογική πρόταση. Toggle_switch(s) → up(s) if ¬ up(s) όταν time=5.Εδώ ο χρόνος δεν έχει άμεση επιρροή στον διακόπτη αναστροφής αλλά τον ενεργοποιεί άμα του ζητηθεί.

```
Int toggle_switch(int s) { }
```

Βλέπουμε ότι η συνάρτηση μας παίρνει σαν όρισμα ένα από τα μέρη της λογικής πρότασης και μπορεί να κλιθεί οποιαδήποτε στιγμή.

Ας δούμε ένα παράδειγμα που έχει σαν όρισμα κάποια ενέργεια αλλά και τον χρόνο. Σε μια εταιρία υπάρχουν διάφοροι υπάλληλοι οι οποίοι ανάλογα με τις ενέργειες τους στο πέρασμα του χρόνου μεταφέρονται σε άλλες καταστάσεις .Θεωρούμε ότι κάποιος υπάλληλος σε κάποια συγκεκριμένη στιγμή κάνει κάποιο παράπτωμα, αυτό σημαίνει ότι η εταιρία πρέπει να λάβει κάποια μέτρα για να το μεταφέρει σε κάποια άλλη κατάσταση ώστε να γνωρίζει ότι με τον συγκεκριμένο υπάλληλο υπάρχει πρόβλημα. Έχουμε την λογική πρόταση: occur(misdemeanor(p),t) *** illegal(p,5m).Πρώτα πρέπει να σκεφτούμε και να κάνουμε κάποια ερωτήματα, ποιος είναι αυτός που έκανε το παράπτωμα; Ποιος είναι ο παράνομος ;Εδώ λόγω του ότι δεν έχουμε πολλές λογικές προτάσεις δεν μπορούμε να σκεφτούμε εύκολα ότι αναφερόμαστε σε έναν υπάλληλο αλλά αν είχαμε πιο πολλές θα καταλήγαμε εύκολα στο συμπέρασμα ότι αναφερόμαστε στον υπάλληλο μιας εταιρίας. Θεωρούμε τελικά ότι το βρίσκουμε. Άρε λέμε ότι ο υπάλληλος θα αποτελεί μια κλάση. Μέσα στην οποία θα υλοποιούμε όλες τις συναρτήσεις μας. Στην συνέχεια βλέπουμε ότι ο υπάλληλοςp μας μια συγκεκριμένη στιγμήt έκανε μια ενέργεια. Η ενέργεια αυτή αποτελεί ένα παράπτωμα. Η ενέργεια αυτή οδηγεί στο να πούμε πως αυτό αποτελεί ένα συμβάν, που έχει σαν αποτέλεσμα ο υπάλληλος μας να μεταφερθεί σε μια κατάσταση , του παρανόμου για κάποιο χρονικό διάστημα. Τώρα με όλα αυτά τα δεδομένα πρέπει να μετατρέψουμε την λογική πρόταση σε C.

Θα πρέπει πρώρα να φτιάξουμε μια κλάση employee.

```
Public class employee { }
```

Σύμφωνα με το τέταρτο βήμα βάζουμε μέσα τις απαιτούμενες συναρτήσεις.

```
Int occur(int misdemeanor, int time) { }

Int illegal(int time) { }
```

Σε αυτό το σημείο θα πρέπει να προσθέσουμε κάποια επιπλέον στοιχεία που δεν φαίνονται στην λογική μας πρόταση έτσι ώστε να βρούμε την συσχέτιση μεταξύ των συναρτήσεων μας. Η πρόταση μας εδώ είναι η δήλωση μιας επιπλέων μεταβλητής που θα συγκρατεί τον χρόνο που έγινε το παράπτωμα του υπάλληλου. Άρα θα έχω:

```
Public class employee{

    Int misdemeanor_time=0;
```

```
    Int occur(int misdemeanor , int time) {
          If( misdemeanor ==1) {
         Misdemeanor_time = time
          return 1;
  }
   Return 0;
}
Int illegal(int time) {
   If(misdemeanor_time == 0 ){
            Return 0;
}
   else if(time< misdemeanor_time  +(5*30)){
        Return 1;
}
Return 0;
}
```

Τέλος πρέπει να αναφέρουμε ότι όταν στις λογικές μας προτάσεις ο χρόνος είναι άπειρος τον διαχωρίζουμε ανάλογα, δηλαδή θα αφήνουμε το αντικείμενο μας σε αυτή την κατάσταση που βρίσκεται μέχρι να υπάρξει άλλη αίτηση αλλαγής της κατάστασης. Επίσης να προσέξουμε τις περιπτώσεις επιλογής του χρόνου. Δηλαδή min(t1, t2) και max(t1,t2) όπου απλά καλούμε τις συναρτήσεις μας με τον μικρότερο και τον μεγαλύτερο χρόνο αντίστοιχα.

Σημείωση: Παραπάνω χρησιμοποιήσαμε μια θεωρητική ενός αλγορίθμου με τέσσερα βασικά βήματα και κάποια παραδείγματα πάνω σε αυτά έτσι ώστε να κατανοήσουμε πως θα γίνει η μετατροπή από μια λογική πρόταση σε γλώσσα υψηλού προγραμματισμού. Ουσιαστικά παραπάνω δεν κατασκευάσαμε κάποιον αλγόριθμο αλλά μια γενική θεωρητική ανάλυση που θα μας βοηθήσει όμως να φτιάξουμε έναν θεωρητικό αλγόριθμο.

***Αλγόριθμος:***

1 Ανάλογα σε τι αναφέρονται οι λογικές μας προτάσεις φτιάχνουμε την ανάλογη κλάση.

1.1 Άνθρωπο ή ζώο.

1.1.1 Μετατρέπουμε τις ενέργειες του σε ακεραίους που μπορούνε να πάρουνε τις τιμές 0 ή 1.

1.1.2. Μετατρέπουμε όλα τα Λ σε && και τα V τα μετατρέπουμε σε ||.

1.2.3. Όταν θέλουμε να μετατρέψουμε τα →, θα πρέπει απλά να προσέξουμε να μην παραβιάζεται αυτό που δηλώνουν και να χρησιμοποιούμε τις κατάλληλες εντολές ελέγχου.

1.2.4. Μετατρέπουμε τα συμβάντα και τις καταστάσεις του σε συναρτήσεις που μπορούνε να πάρουνε ορίσματα ενέργειες χρόνο και ότι άλλο θεωρούμε εμείς αναγκαίο.

1.1.4.1. Χρησιμοποιούμε ότι επιπλέον στοιχεία χρειαζόμαστε που δεν μας τα δίνουνε οι λογικές προτάσεις κυρίως για να ελέγχουμε τον χρόνο.

1.1.4.2. Προσέχουμε να εκπληρώνονται οι χρονικοί περιορισμοί.

1.2 Πράγμα

1.2.1. Μετατρέπουμε τα μερη του σε ακεραίους που μπορούνε να πάρουνε τιμές 0 ή 1

1.2.2 Μετατρέπουμε όλα τα Λ σε && και όλα τα V τα μετατρέπουμε σε||.

1.2.3 Όταν θέλουμε να μετατρέψουμε → , θα πρέπει απλά να προσέξουμε να μην παραβιάζεται αυτό που δηλώνουν και χρησιμοποιούμε τις κατάλληλες εντολές ελέγχου.

1.2.4 Μετατρέπουμε τα εργαλεία σε συναρτήσεις που μπορούνε να πάρουνε ορίσματα ενέργειες , χρόνο και ότι άλλο εμείς θεωρούμε αναγκαίο.

1.2.4.1 Χρησιμοποιούμε ότι επιπλέον στοιχεία χρειαζόμαστε που δεν μας τα δίνουνε οι λογικές προτάσεις κυρίως για να ελέγχουμε τον χρόνο.

1.2.4.2 Προσέχουμε να εκπληρώνονται οι χρονικοί περιορισμοί

# 3 Απόδειξη της μετατροπής λογικών προτάσεων σε C.

Αφού λοιπόν έχουμε τον θεωρητικό αλγόριθμο μας για να μπορούμε να μετατρέψουμε μια λογική πρόταση σε κώδικα γλώσσας υψηλού προγραμματισμού, θα δείξουμε μια απόδειξη η οποία επαληθεύει την μετατροπή??? που δημιουργήσαμε.

**ΑΠΟΔΕΙΞΗ**

Μια λογική πρόταση αποτελείται από δύο σκέλη. Για την απόδειξη μας θα πρέπει να λάβουμε υπόψην και τα δυο σκέλη έτσι ώστε να μπορούμε να πούμε ότι όντως η μετατροπή που κάνα με είναι σωστή.

**Πρώτη περίπτωση** ( →) Σε αυτή την πρώτη περίπτωση θα πρέπει να δούμε ότι αν οι όροι του πρώτου σκέλους της λογικής μας πρότασης αποδίδονται το ίδιο και μετά την μετατροπή. Δηλαδή θα πρέπει ένα ένας όρος της λογικής πρότασης είναι αληθείς τότε σύμφωνα με τον παραπάνω αλγόριθμο να έχει την τιμή 1 αλλιώς αποδίδονται σωστά και οι χρονικοί περιορισμοί εάν υπάρχουνε. Εάν όλοι οι όροι του πρώτου σκέλους ανταποκρίνονται τότε συνεχίζουμε. Ετσι μέχρι στιγμής έχουμε καταφέρει να δείξουμε ότι το ένα σκέλος της λογικής πρότασης έχει μετατραπεί σωστά. Στην συνέχεια ένα στο δεύτερο σκέλος όλοι οι όροι ανταποκρίνονται στις τιμές της λογικής πρότασης αλλά και στους χρονικούς περιορισμούς (εάν υπάρχουν) , τότε και το δεύτερο μας σκέλος έχει μετατραπεί σωστά και μπορούμε να πούμε με σιγουριά ότι η μετατροπή έχει γίνει σωστά.

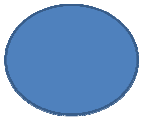
Δεύτερη περίπτωση( )

Σε αυτή την δεύτερη περίπτωση θα πρέπει πέρα από αυτά που αποδείξαμε στην πρώτη περίπτωση , επιπρόσθετα να αποδείξουμε ότι μπορούμε να πάμε τόσο από το πρώτο μέλος στο δεύτερο αλλά και το αντίστροφο. Άρα θα πρέπει να δούμε ότι ένα ανταποκρίνονται οι όροι του πρώτου σκέλους και του δεύτερου τότε θα πρέπει εάν είχαμε ένα από τα δύο σκέλη σαν δεδομένα να συμπεράνουμε το δεύτερο. Δηλαδή ένα στον κώδικα μας είχαμε για το πρώτο σκέλος ότι είναι 0 θα έπρεπε να γνωρίζουμε ότι το δεύτερο σκέλος έχει τιμή 1 , και αν γνωρίζαμε ότι το δεύτερο σκέλος έχει τιμή 1 ότι τότε το πρώτο σκέλος έχει τιμή 0.Αν βγαίνει το συμπέρασμα αυτό από τον κώδικα μας τότε μπορούμε να πούμε ότι είναι σωστός.

## 4 Κατασκευή διαγραμμάτων

Στο σημείο αυτό , έχουμε δείξει πώς να μετατρέπουμε λογικές προτάσεις σε γλώσσα υψηλού προγραμματισμού , θα δείξουμε και πώς να κατασκευάζουμε διαγράμματα καταστάσεων, τα οποία μπορούν να μας διευκολύνουν πάρα πολύ στο να αναγνωρίζουμε πολύ εύκολα τι δηλώνουν τα σύνολα των λογικών προτάσεων αλλά και μεγάλα τμήματα κώδικα.

Αρχικά όλα τα διαγράμματα μας ξεκινάνε από την αρχική κατάσταση που συμβολίζεται έτσι:



Επειδή όμως μπορεί να έχουμε επιστροφές στην αρχική κατάσταση καλό θα ήταν να χρησιμοποιήσουμε ένα επιπλέον σχήμα 2 , το οποίο θα επεξεργάζεται αυτό ότι διασυνδέσεις πρόκειται να γίνουν με την αρχική κατάσταση. Έτσι έχουμε το εξής σχήμα:


Τα σχήματα 2δηλώνουνε τις διάφορες καταστάσεις που βρισκόμαστε σε βάθος χρόνου:

Σχήμα 2


Τα βέλη που υπάρχουνε στις συνδέσεις μεταξύ των καταστάσεων δείχνουνε τις μεταβάσεις που μπορούμε να κάνουμε από μια κατάσταση σε μία άλλη


Οι σημειώσεις που υπάρχουνε επάνω στις συνδέσεις αποτελούνε τις ενέργειες και τα συμβάντα , όταν έχουμε ανθρώπους ή ζώα , που πρέπει να γίνουν για να πραγματοποιηθούνε οι μεταβάσεις ή τα μέρη και τα εργαλεία που πρέπει να ενεργοποιηθούνε , όταν έχουμε πράγματα. Επίσης αποτελούνε και τους περιορισμούς χρόνου που πρέπει να ισχύουν έτσι ώστε να πραγματοποιηθούνε οι μεταβάσεις μεταξύ των καταστάσεων.

Occur or action/part or tool/time


Όπως βλέπουμε στα διαγράμματα μας δεν έχουμε βάλει τελικές καταστάσεις και αυτό γιατί έχουμε να κάνουμε και με χρόνο. Με συνέπεια να μην υπάρχει μια τελική κατάσταση αλλά να έχουμε μια συνεχόμενη εναλλαγή καταστάσεων. Στην περίπτωση που θα θέλαμε να απεικονίσουμε με διαγράμματα κάποια συγκεκριμένη χρονική στιγμή θα μπορούσαμε να χρησιμοποιήσουμε και την τελική της κατάσταση.

## 5.Διάφορα παραδείγματα.

Στο σημείο αυτό θα δείξουμε διάφορα παραδείγματα μετατροπής λογικών προτάσεων σε C, έτσι ώστε να μπορέσουμε να κατανοήσουμε τον αλγόριθμό μας . Επίσης θα παρουσιάσουμε και τα διαγράμματα που προκύπτουνε για να μπορούμε εύκολα και γρήγορα να κατατοπιζόμαστε για την συνολική ερμηνεία τόσο των λογικών προτάσεων όσο και του κώδικα σε C.

Θα ξεκινήσουμε με ένα απλό παράδειγμα ενός ηλεκτρικού κυκλώματος που έχει δύο διακόπτες και ένα λαμπτήρα και έχουμε το παρακάτω σύνολο λογικών προτάσεων που θέλουμε να μετατρέψουμε σε C.

Part _1

Up(s1) Λ up(s2) → light

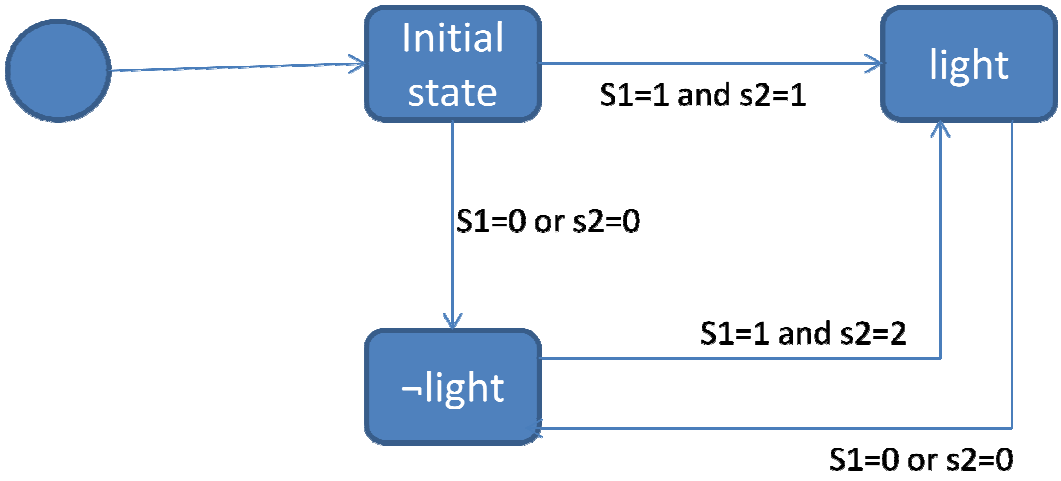
¬up(s1)→¬light

¬up(s2)→¬light

Σε C

```
If(s1==1 && s2==1){
    Light;
}
Else if(s1==0) light=0;
Else if(s2==0) light=0;
```

Ο κώδικας C σύμφωνα με τον αλγόριθμό μας είναι ο παραπάνω, επίσης μπορούμε να τρέξουμε το συγκεκριμένο part του source code και να δούμε τα αποτελέσματα που εκτυπώνονται και να δούμε ότι πράγματι επαληθεύουν την μετατροπή που κάναμε. Το διάγραμμα για το παραπάνω παράδειγμα είναι το παρακάτω.



Στην συνέχεια θα δείξουμε ένα παράδειγμα με ένα εργαλείο το οποίο αλλάζει την κατάσταση των διακοπτών. Το εργαλείο αυτό σύμφωνα με τον αλγόριθμό μας θα πρέπει να το μετατρέψουμε σε συνάρτηση που θα παίρνει σαν όρισμα ένα μέρος του συστήματος δηλαδή ένα διακόπτη.

Part 2

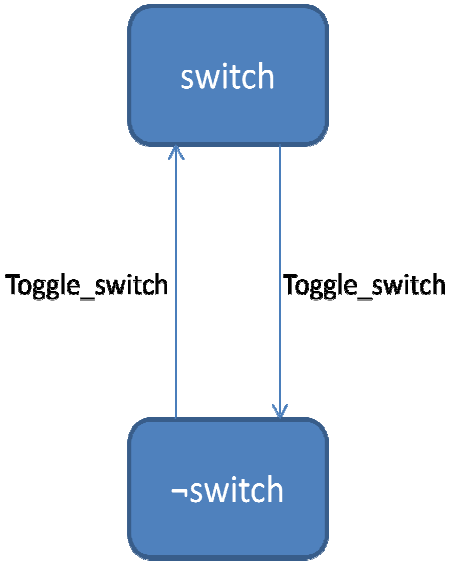
Toggle_switch(s) → up(s) if ¬up(s)

Toggle_switch(s)→ ¬up(s) if up(s)

Σε C

```
Int toggle_switch(int s){
        If(s==0){
            S=1;
}
Else if(s==1){
            S=0;
}
Return s;
}
```

Στο παράδειγμα αυτό είδαμε πως μπορούμε να μετατρέψουμε ένα εργαλείο ενός συστήματος σε συνάρτηση , για να εξακριβώσουμε ότι όντως το εργαλείο δουλεύει μπορούμε να τρέξουμε το Part και να δούμε τα αποτελέσματα που εκτυπώνονται. Ένα διάγραμμα για τον παραπάνω κώδικα είναι αυτό που ακολουθεί.



Προχωρώντας  πειραματιζόμαστε με κάποιες απλές περιπτώσεις αλλά και παράλληλα πολύ σημαντικές για να κατανοήσουμε το πώς γίνονται οι μετατροπές με βάση τον αλγόριθμό μας .

Part 3

Up(s1) causes light if up(s2)

Up(s2) causes light if up(s1)

¬Up(s1) causes ¬light if T

¬Up(s1) causes ¬light if T

Σε C

```
If(s1==1 && s2==1){
    Light=1;
}
If(s2==1 && s1==1){
     Light=1;
}
If(s1==0 && (s2==0 *** s2==1)){
          Light=0;
}
If(s2==0 &&(s1==0 *** s1==1)){
      Light=0;
}
```

Στο συγκεκριμένο παράδειγμα δεν θα παρουσιάσουμε κάποιο σχήμα απλά θα μπορούσαμε να πούμε ότι ένα ενδεικτικό παράδειγμα διαγράμματος είναι αυτό του πρώτου μας παραδείγματος.

Στο επόμενο παράδειγμα έχουμε ένα ηλεκτρικό κύκλωμα με 3 διακόπτες ένα ηλεκτρονόμο και ένα λαμπτήρα. Ενδιαφέρον εδώ προκαλεί το πώς θα εκφράσουμε σε γλώσσα υψηλού προγραμματισμού τις εξαρτήσεις μεταξύ των δύο σκελών των λογικών μας προτάσεων

Part 4

a) light  up(s1) Λ up(s2)

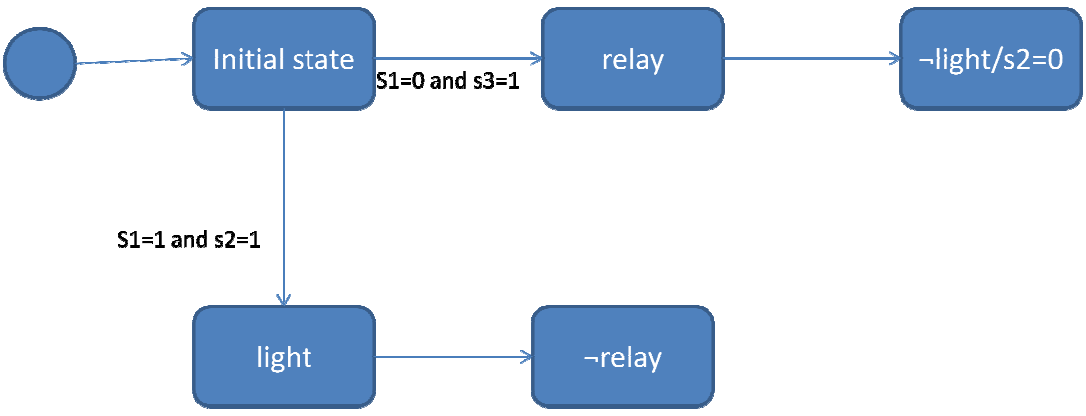
b)relay ¬ up(s1) Λ up(s3)

c) relay → ¬ up(s2)

Σε C

```
If(s1==1 && s2==1){
        Light=1;
}
If(s1==0 && s3==1){
    Relay=1;
}
If(relay==1){
   S2=0;
```

Όπως βλέπουμε από την παραπάνω μετατροπή μας βολεύει πολύ παραπάνω να ξεκινήσουμε από το δεύτερο μέρος και να καταλήξουμε στο πρώτο για τα a , b χωρίς να παραβιάζεται η ερμηνεία των δύο λογικών προτάσεων. Ενώ στην Τρίτη λογική πρόταση βλέπουμε ότι δεν μπορούμε να ξεκινήσουμε από το δεύτερο μέρος και να καταλήξουμε στο πρώτο γιατί εάν το κάνουμε αυτό θα παραβιαστεί η ερμηνεία της λογικής μας πρότασης. Το σχήμα για τον παραπάνω κώδικα είναι αυτό που ακολουθεί




Το επόμενο παράδειγμα είναι παραπλήσιο με το προηγούμενο. Έχουμε ένα ηλεκτρικό κύκλωμα με 5 διακόπτες 1 ηλεκτρονόμο και 1 λαμπτήρα.

Part 5

a)light ≡ up(s1)Λ up(s2)

b)light≡ up(s4) Λ up(s5)

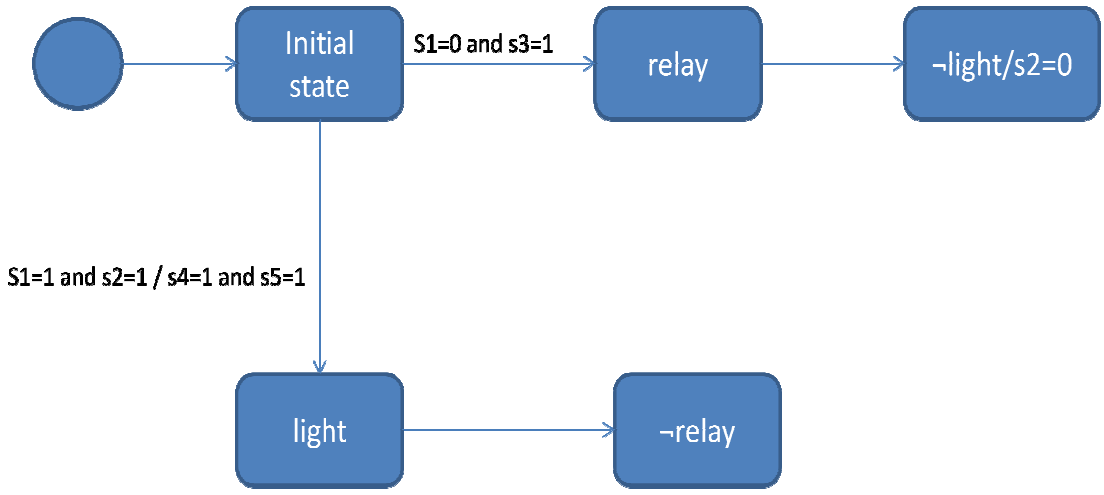
c) relay ≡¬up(s1)Λ up(s3)

δ) relay → $A$¬ up(s2)

Σε C

```
If(s1==1 && s2==1){
      Light=1;
}
If(s4==1 && s5==1){
  Light=1;
}
If(s1==0 && s3==1){
    Relay=1;
}
If(relay==1){
     S2=0;
}
```

Η παραπάνω μετατροπή έγινε σύμφωνα με την λογική του προηγούμενου παραδείγματος και του αλγόριθμού μας. Το διάγραμμα που προκύπτει είναι το παρακάτω:




Μια μικρή μετατροπή του προηγούμενου παραδείγματος είναι αυτή που βλέπουμε παρακάτω.

Part6

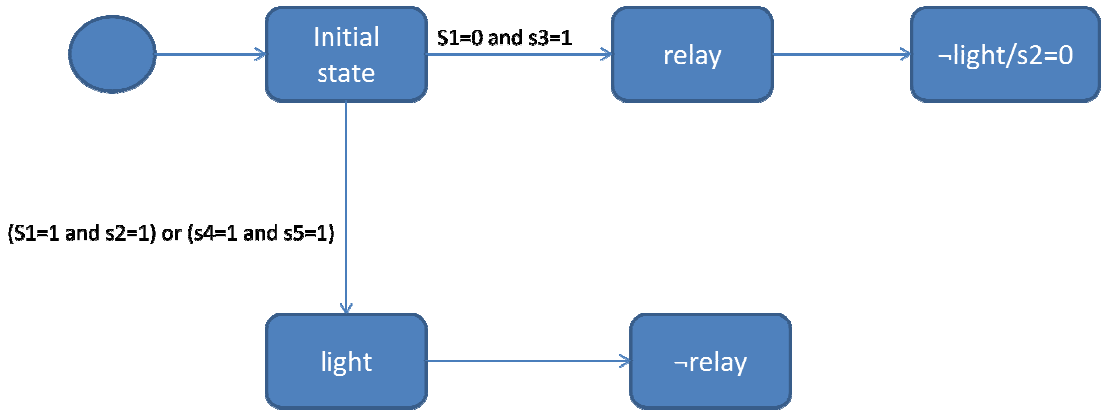
a)light (up(s1) Λ up(s2)) V (up(s4) Λ up(s5))

b)relay ¬up(s1) Λ up(s3)

c) relay → ¬up(s2)

Se c

```
If((s1==1 && s2==1)  *** (s4==1 && s5==1)){
        Light=1;
}
If(s1==0 && s3==1){
         Relay=1;
}
If(relay ==1){
     S2=0;
}
```




Στα τρία προηγούμενα παραδείγματα στα οποία βλέπουμε ότι υπάρχουνε μικρές διαφορές μεταξύ τους μπορούμε να εξακριβώσουμε ότι οι μετατροπές τους έχουν γίνει σωστά , ένα τρέξουμε το κάθε part.Τα αποτελέσματα που βγαίνουν θα δούμε ότι συμβαδίζουνε με αυτά που περιμέναμε από τις λογικές προτάσεις. Έτσι μπορούμε να είμαστε σίγουροι για τις μετατροπές μας.

Στο παράδειγμα που ακολουθεί θα δούμε ότι για να μπορέσουμε να εκφράσουμε το σύνολο αυτών των λογικών προτάσεων θα πρέπει να λάβουμε υπόψην μας και τον χρόνο. Καταρχάς για να υλοποιήσουμε αυτές τις λογικές προτάσεις θα πρέπει να σκεφτούμε γενικά. Ποιος θα μπορούσε να πιεί αλκοόλ; Ποιος θα μπορούσε να είναι μεθυσμένος; Ποιος θα μπορούσε να οδηγήσει ένα αυτοκίνητο; Όλες αυτές οι ερωτήσεις οδηγούν στην απάντηση ένας άνθρωπος. Με δεδομένο αυτό σκεφτόμαστε ότι πρέπει να φτιάξουμε μια κλάση person. Έπειτα σκεφτόμαστε για το τι πεδία θα έχει η κλάση μας αλλά και το μεθόδους . Το να είναι κάποιος μεθυσμένος ή το να οδηγεί κάποιος ένα αμάξι είναι καταστάσεις στις οποίες βρίσκεται ένας άνθρωπος, άρα για αυτό το λόγο θα θεωρήσουμε αυτές τις δύο καταστάσεις ως μεθόδους της κλάσης μας. Θα πρέπει να χρησιμοποιήσουμε και μια επιπλέον μέθοδο την occur όπου θα χρησιμοποιήσουμε τον ακέραιο drink_alcohol. Μπορεί να πάρει τιμές 0 ή 1. Αυτή η μέθοδος αποτελεί το συμβάν κατά το οποίον ένας άνθρωπος πίνει αλκοόλ. Σε όλες μας τις μεθόδους μπορούμε να διακρίνουμε δύο καταστάσεις άρα αυτό έχει ως συνέπεια να επιστρέφουνε 0 ή 1 όπως είπαμε και παραπάνω. Δηλαδή αν ζητήσουμε από την μέθοδο drunk να μας πει εάν ένας άνθρωπος είναι μεθυσμένος μια δεδομένη χρονική στιγμή τότε αυτή θα επιστρέψει 0 εάν **drunk και 1 drunk. Τον χρόνο θα τον χρησιμοποιήσουμε ως πεδίο της κλάσης μας και ειδικότερα τον χρόνο που θα ξεκινάει να πίνει κάποιος που είναι και αυτό που μας ενδιαφέρει. Έτσι θα μπορούμε να κάνουμε τις συγκρίσεις που μας ενδιαφέρουν έτσι ώστε να δούμε εάν κάποιος είναι πιωμένος ή αν μπορεί να οδηγήσει κάποια συγκεκριμένη στιγμή. Με όλα αυτά σαν δεδομένα μπορούμε να κάνουμε την μετατροπή σε κώδικα c και να δούμε ότι όντως δουλεύει.

Part 7

Occur(drink_alcohol(p),t) *** drunk(p,t') Λ (t' < t +5h)

Drunk(p,t) →¬ drive(p,t)

////////////////////////////////////////////////////////////////////////////////////////////////

```
Int take_salary(int time){
  /* if the employee is suspended, he can't take salary this time.*/
If(this.suspended(time) == 1){
        Return 0;
}
/* if the employee isn't suspended, he can take salary this time.*/
/* else if(this.suspended(time) == 0 */
Return 1;
}
```

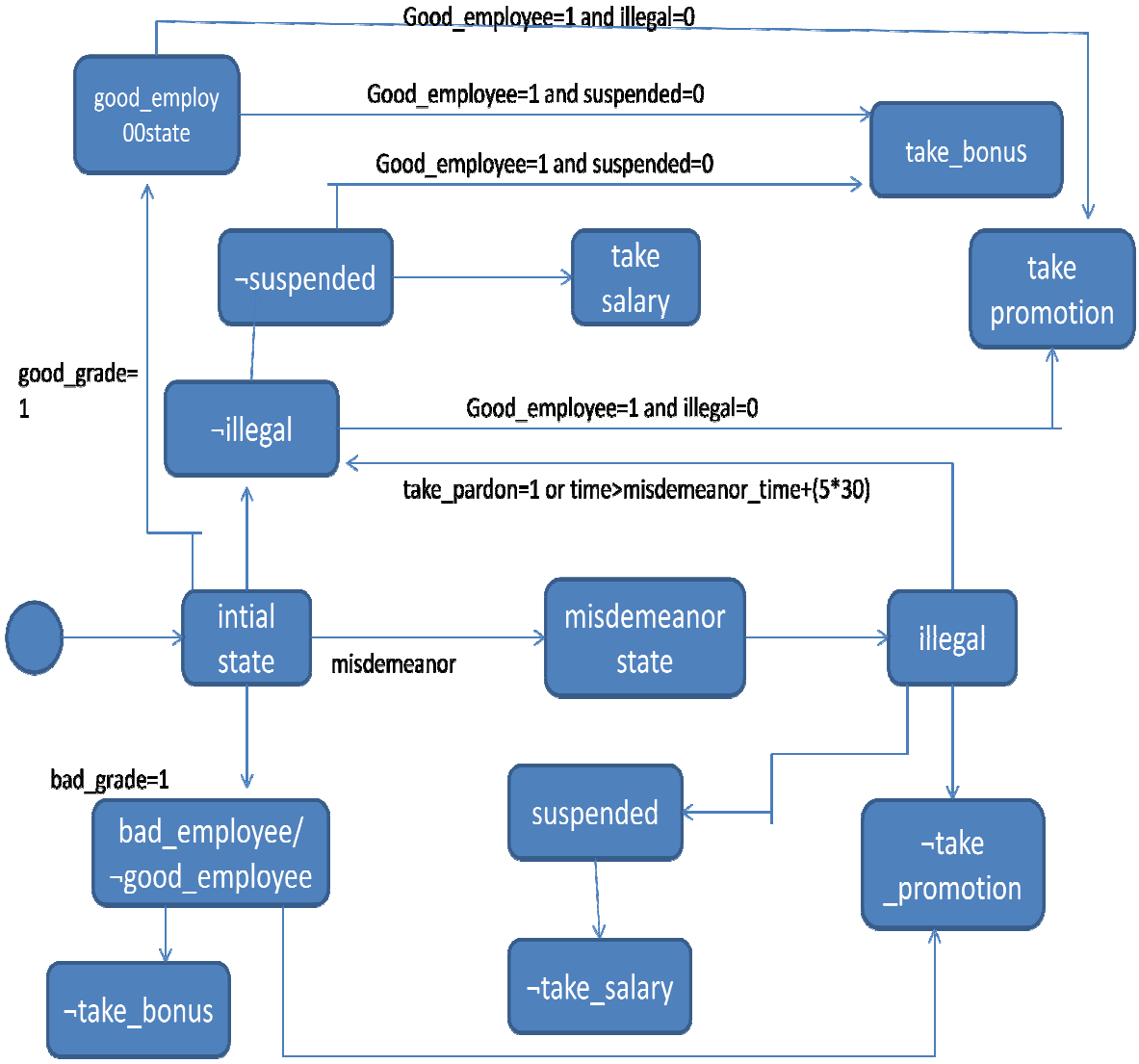
////////////////////////////////////////////////////////////////////////////////////////////////

```
Int take_bonus(int time){
/* we have: ¬suspended(p,t1) Λ good_emploee(p,t2) →
*                  take_bonus(p,min(t1,t2))
*/
/* The employee couldn't take bonus if he isn't good employee. */
If(this.good_employee(time) == 0){
        Return 0;
}
/* The employee can take bonus. */
Else if((this.suspended(time) == )) &&
            (this,good_employee(time)  == 1)){
   Return 1;
}
/* the employee can't take bonus. */
```

Return 0;

}

Για να επιβεβαιώσουμε ότι ο κώδικάς μας λειτουργεί σωστά στο βάθος χρόνου , δηλαδή να έχουνε μετατραπεί σωστά οι λογικές προτάσεις μας και ισχύουν ανά πάσα στιγμή, αρκεί να τρέξουμε το part και να το επιβεβαιώσουμε. Το διάγραμμα για το παραπάνω παράδειγμα είναι το παρακάτω.




}

/* There is no default situation.So we give a negative number. */

Return -1;

}

////////////////////////////////////////////////////////////////////////////////////////////////////

Void bad_employee(int time){

```
    /* No action. */
}
////////////////////////////////////////////////////////////////////////////////////////////
Int suspended(int time){
        /* If the employee is illegal, he is also suspended the same time. */
        If(this.illeal(time) == 1){
                Return 1;
        }
         /* Else he is ¬ suspended.*/
      Return 0;
}
Int take_promotion(int time){
/* If the employee is illegal, he can't take promotion this time. */
If(this.illegal(time) == 1){
           Return 0;
}
/* If the employee isn't good employee(time) == 0)}
          Return 0;
}
/* If something from the other doesn't happen, then the employee can take promotion. */
             Return 1;
}
* Take_salary,take_bonus.
*/
Int illegal(int time){
  /* The initial state of employee is ¬illegal. */
Is(misdemeanor_time == 0){
```

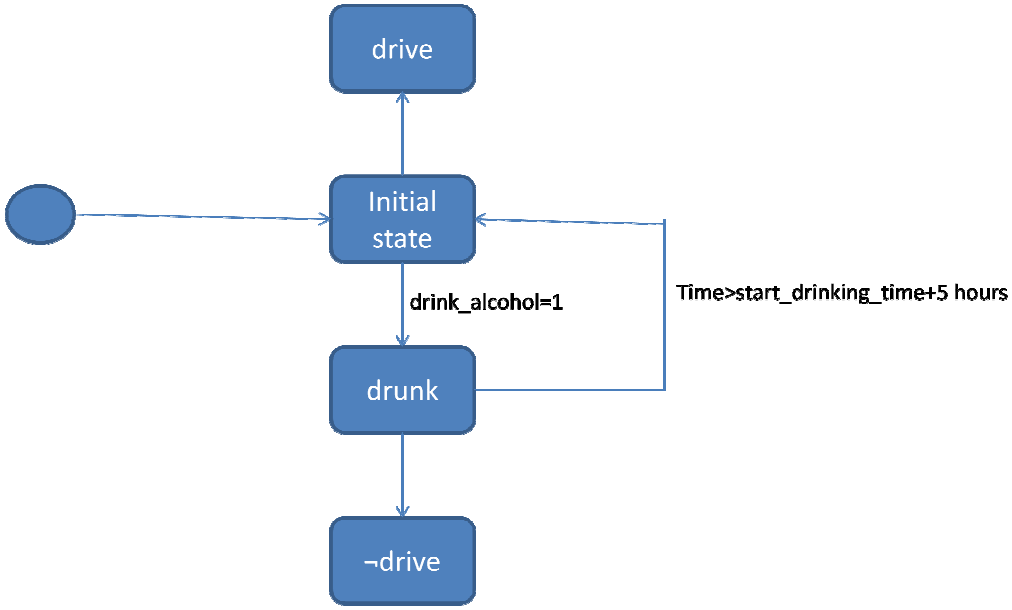
```
      Return 0;
}
/* From the time, employee , take_pardon he is legal again. */
Else if(misdemeanor_time ! =0 &&
       (take_pardon_time < = time) && (take_pardon_time ! = 0)){
        Return 0;
}
/* if the employee make an misdemeanor he will be illegal until the
* time he makes the misdemeanor + 5m.
*/
else if(time < misdemeanor_time +(5*30)){
     return 1;
{
/* The default situation is ¬ illegal.*/
Return 0;
}
////////////////////////////////////////////////////////////////////////////////////////////
Int good_employee(int time){
/* From the time, employee ,take a bad_grade become a
 *¬good_employee.
*/
If((bad_grade_time <= time )&& (bad_grade_time != 0)){
  Return 0;
}
/* From the time, employee ,take a good_grade become a
* good_employee.
*/
```

```
Else if((good_grade_time <= time) && (good_grade_time ! = 0)){
      Return 1;
```

Σε C

```
Publiv class part_7_person {
   Int start_drinking_time=0;
   Int occur(int drink_alcohol,int time){
       If(drink_alcohol == 1){
      Start_drinking_time = time;
      Return 1;
       }
     Return 0;
}
Int drive(int time2){
      If(this.drunk(time2) == 1){
          Return 0;
           }
           Return 1;
       }
}
```

Για να επιβεβαιώσουμε ότι ο κώδικας μας είναι σωστός θα πρέπει να τρέξουμε το συγκεκριμένο part. Το διάγραμμα μας σε αυτή την περίπτωση είναι το παρακάτω.




Είναι ακόμη απλό παράδειγμα μετατροπής λογικών προτάσεων σε κώδικα C που είναι ίδιο με το παράδειγμα 2 αλλά οι λογικές προτάσεις εκφράζονται διαφορετικά και πάλι βλέπουμε πως ο αλγόριθμος καταλήγει στο ίδιο αποτέλεσμα.

Part_8

Toggle_switch(s1) initiates up(s1) when ¬up(s1)

Toggle_switch(s1) initiates ¬up(s1) when up(s1)

Toggle_switch(s1) terminates up(s1) when up(s1)

Toggle_switch(s1) terminates ¬up(s1) when ¬up(s1)

Toggle_switch(s2) initiates up(s2) when ¬ up(s2)

Toggle_switch(s2) initiates ¬up(s2) when up(s2)

Toggle_switch(s2) terminates up(s2) when up(s2)

Toggle_switch(s2) terminates ¬up(s2) when ¬up(s2)

Σε C

```
Int toggle_switch(int s){
    If(s==0 ){
        S=1;
    }
```

```
Else if(s==1){
        S=0;
}
Return s;
}
```

Σε αυτή την περίπτωση η δυσκολία δεν είναι στο να εκφραστούνε οι κανόνες, που τους πιο πολλούς τους έχουμε υλοποιήσει ήδη σε προηγούμενα παραδείγματα αλλά στο ότι πλέον εμπλέκεται και ο χρόνος. Τώρα που εμπλέκεται και ο χρόνος βλέπουμε ότι δεν έχουμε στατικές καταστάσεις πλέον αλλά δυναμικές καταστάσεις των οποίων οι τιμές τους εξαρτώνται από αυτόν(χρόνο).Για να εκφραστεί καλύτερα το παρακάτω μέρος θα πρέπει να δούμε ολόκληρο το πρόγραμμα και όχι ένα μέρος του που αντιστοιχεί μόνο στους περιορισμούς. Αν τρέξουμε το αντίστοιχο part κώδικα θα δούμε ότι αυτό επιβεβαιώνεται.

Part_9

light whenever up(s1) Λ up(s2)

¬light whenever ¬up(s1) Λ ¬up(s2)

Time: 0

up(s1) holds at 0

¬up(s2) holds at 0

¬light holds at 0

Actions:

toggles_switch(s2) happens at 4

toggle_switch(s2) initiates up(s2) when ¬up(s2)

toggle_switch(s2) terminates ¬up(s2) when ¬up(s2)

toggles_switch(s1) happens at 8

toggle_switch(s2) initiates ¬up(s2) when up(s2)

toggle_switch(s2) terminates up(s2) when up(s2)

time: 4

up(s1) holds at 4

up(s2) holds at4

light holds at 4

time:8

¬up(s1) holds at 8

up(s2) holds at 8

¬light holds at 8

Σε C

```
int toggle_switch(int s){
        if(s==0){
            s=1;
        }
        else if( s==1){
            s=0;
       }
       return s;
}
int rules(int s1, int s2){
        if(s1==1 && s2==1){
            return 1;
       }
       else if( s1 ==0  ** s2==0){
            return 0;
      }
      return 0;
}
```

Το διάγραμμα σε αυτή την περίπτωση δεν διαφέρει από το διάγραμμα του πρώτου παραδείγματος παρόλο που εδώ παίζει ρόλο και ο χρόνος. Αυτό γιατί ο χρόνος αποτελεί κριτήριο για το πότε θα αλλάξουμε κατάσταση και όχι αν θα αλλάξουμε όντως κατάσταση. Στο σημείο αυτό για να υλοποιήσουμε το παρακάτω σύνολο λογικών προτάσεων σε κώδικα

C θα πρέπει να σκεφτούμε σύμφωνα με τον αλγόριθμο μας όπως και στο παράδειγμα 7 που είδαμε πριν.

Part_10

Occur(misdemeanor(p),t) ** illegal(p,5m) (1)

Occur(take_pardon(p),t) ** ¬illegal(p,**) (2)

Occur(bad_grade(p),t) ** ¬good_employee(p,**) (3)

Occur(good_grade(p),t) ** good_employee(p,**) (4)

_____

Illegal(p,t1) ** suspenden(p,t1) (5)

Illegal(p,t1) ** ¬take_promotion(p,t1) (6)

Suspended(p,t1) ** ¬take salary(p,t1) (7)

¬good_employee(p,t1) ** ¬take_promotion(p,t1) (8)

¬suspended(p,t1) Λ good_employee(p,t2) **

Take_bonus(p,min(t1,t2)) (9)

¬good_employee(p,t1) ** ¬take_bonus(p,t1) (10)

¬suspended(p,t1) ** take_salary(p,t1) (11)

Σε C

```
Public class part_10_employee{
    Int misdemeanor_time=0;
    Int take_pardon_time=0;
    Int bad_grade_time=0;
    Int good_grade_time=0;

////////////////////////////////////////////////////////////////////////////////////////////
///                                    ACTIONS                                           ///
////////////////////////////////////////////////////////////////////////////////////////////

/* We have four functions for the actions:
* misdemeanor,take_pardon,bad_grade,good_grade.
```

```
*/
int occur1(misdemeanor,int time){
    If(misdemeanor == 1){
          misdemeanor_time = time;
        take_pardon_time = 0;
        Return 1;
    }
return 0;
}
int occur2(int take_pardon,int time){
      If((take_pardon == 1) && (misdemeanor_time == 0)){
            System.out.println("\nEmployee doesn't do a misdemeanor!\n");
             return 1;
}
if (take_pardon == 1){
      take_pardon_time = time;
       misdemeanor_time = 0;/* Intialize the misdemeanor_time.*/
        return 1;
   }
    return 0;
}
Int occur3(int bad_grade,int time){
       If(bad_grade == 1){
             bad_grade_time = time;
              good_grade_time = 0;/* Intialize the good_grade_time.*/
             return 1;
   }
```

```
    return 0;
}

Int occur4(int good_grade,int time){
     if(good_grade == 1){
            good_grade_time = time;
            bad_grade_time = =;/* Intialize the bad_grade_time. */
            return 1;
    }
    return 0;
}

////////////////////////////////////////////////////////////////////////////////////////////////
///                                          FLUENTS                                         ///
////////////////////////////////////////////////////////////////////////////////////////////////

/* WE have seven functions for the fluent:
* illegal,good_employee,bad_employee,suspended,take_promotion,
```