SAPIENZA
UNIVERSITÀ DI ROMA

JTrafficControl

Corso di Metodi Formali nell'Ingegneria del Software
A.A. 2006-2007

Docente Toni Mancini

A cura di:
Raffaele Giuliano, Marco Piva, Fabio Terella, Emanuele Tracanna

JTrafficControl - Introduzione

- **Cos'è?** *JTrafficControl* è una applicazione in Java in grado di risolvere il problema dello scheduling per i semafori di un incrocio stradale.

- **Perché?** Lo scopo della tesina è di coniugare quanto appreso durante il corso di Metodi Formali nell'Ingegneria del Software con un caso concreto e reale.

Definizione del modello

Dopo aver individuato il dominio dell'applicazione, siamo passati alla definizione formale di ***incrocio***, per trovare un modello sul quale operare.

- Un *incrocio* è un insieme di strade.
- Una *strada* è un insieme di *corsie* che può avere o meno attraversamento pedonale e può permettere o meno l'inversione di marcia.
- Una *corsia* può essere entrante o uscente, rispettivamente se entra o esce dall'incrocio.
- Ogni *corsia* ha associato uno specifico tipo di traffico, con il vincolo che una corsia entrante ha solamente una tipologia di traffico, mentre una corsia uscente può avere associate una o più tipologie di traffico.
- Le tipologie di traffico permesse sono ***Auto*** e ***Tram***.

Definizione del modello

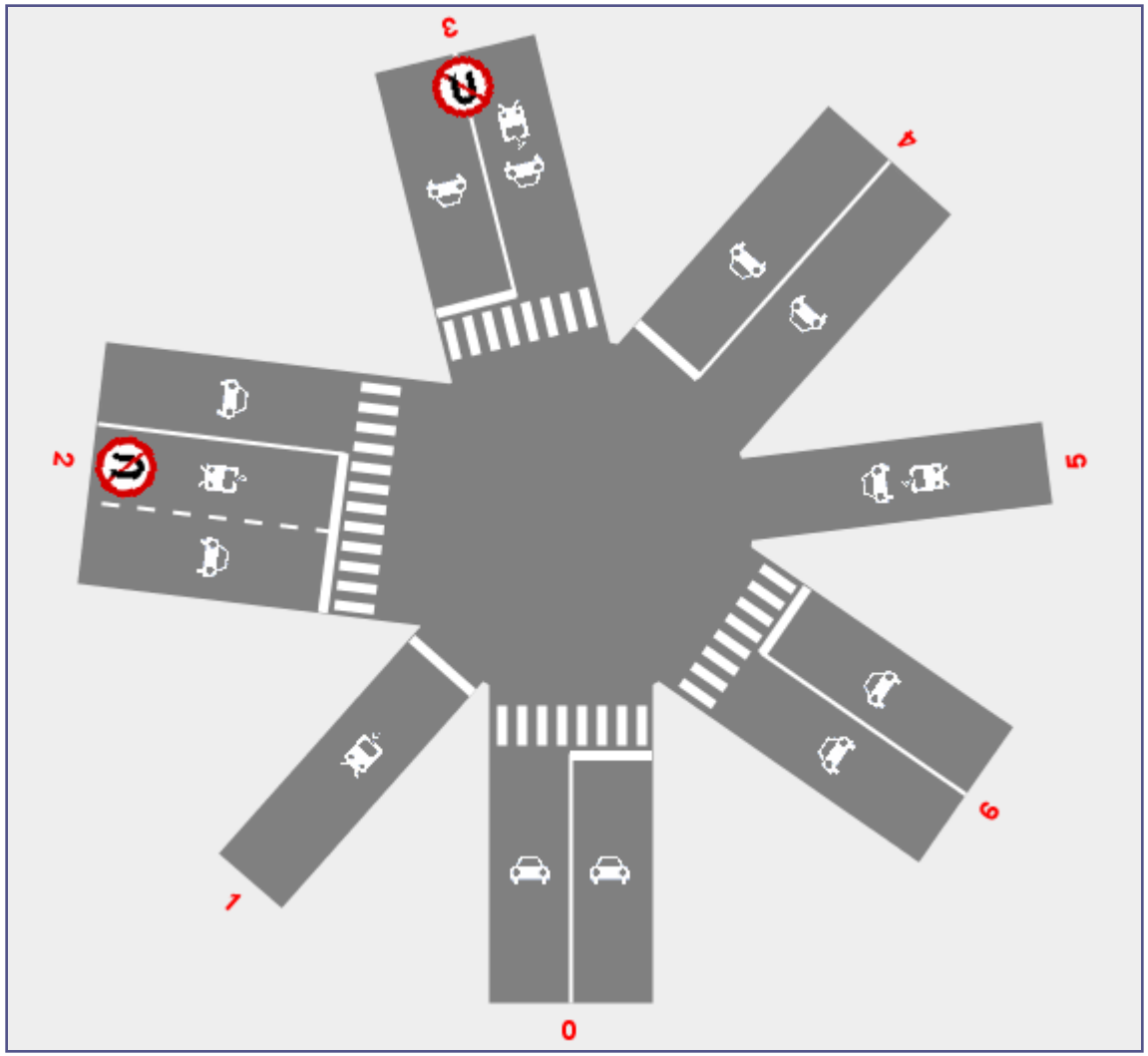
- Il modello lavora per ***corsie logiche***, ovvero non possono esistere all'interno della medesima strada, corsie entranti o uscenti con la stessa tipologia di traffico.

- Una strada può avere **al più** una corsia entrante (uscente) per ogni tipo di traffico. L'utilizzo di corsie logiche non è una limitazione, ma permette una più semplice modellazione.

- Una corsia logica può corrispondere a più corsie fisiche.

- Scelta una strada di riferimento, a cui viene assegnato l'indice **0**, le altre strade vengono numerate con indice crescente, proseguendo in senso orario finché tutte le strade dell'incrocio non vengono numerate.

Esempio di Incrocio



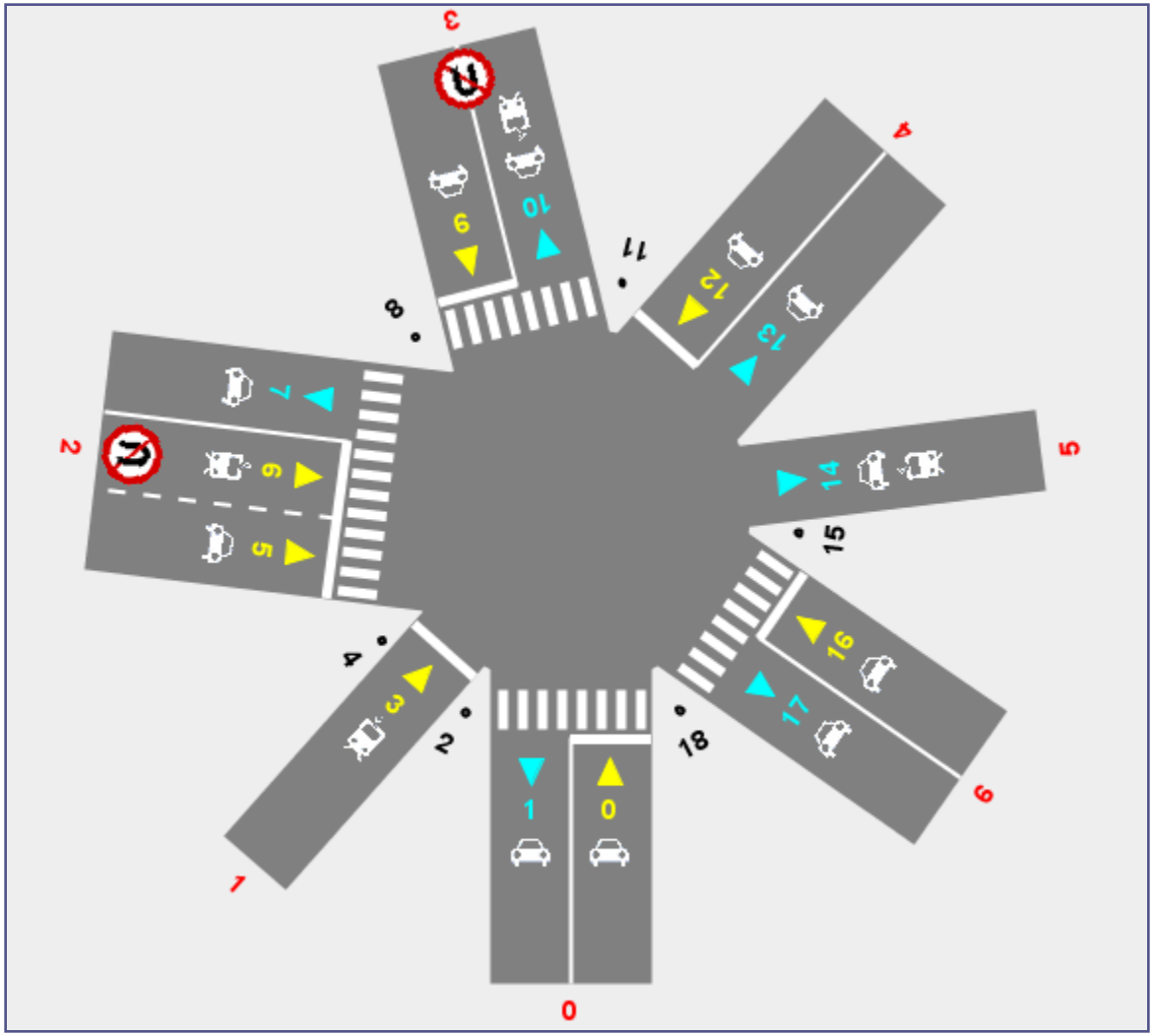
Definizioni varie

La creazione dell'incrocio, con le sue strade e rispettive corsie, ci permette di definire le seguenti cose:

- ***Sorgente***: consideriamo punto sorgente ogni corsia entrante nell'incrocio
- ***Destinazione***: consideriamo punto di destinazione ogni corsia uscente dall'incrocio.
- ***Punti di attraversamento pedonale***: consideriamo un punto aggiuntivo, apposto tra due strade adiacenti ogniqualvolta almeno una delle due strade ha un attraversamento pedonale.

I punti, appartenenti a questi 3 insiemi, sono numerati con indici crescenti procedendo in senso orario, a partire dalla prima corsia entrante della strada **0**.

Esempio di Incrocio numerato



Definizione degli insiemi S, D, A

In base agli insiemi precedentemente individuati, possiamo dare le seguenti definizioni:

$$T = \{insieme\ di\ tutti\ i\ tipi\ di\ traffico\ veicolare\ (esclusi\ quindi\ i\ pedoni)\}$$

$$S_{TipoTraffico} = \{insieme\ delle\ sorgenti\ che\ consentono\ il\ transito\ di\ TipoTraffico\}$$

$$D_{TipoTraffico} = \{insieme\ delle\ destinazioni\ che\ consentono\ il\ transito\ di\ TipoTraffico\}$$

L'unione di questi insiemi, così definiti, formano rispettivamente

$$S = \bigcup_{TipoTraffico \in T} S_{TipoTraffico} = \{insieme\ di\ tutte\ le\ sorgenti\}$$

$$D = \bigcup_{TipoTraffico \in T} D_{TipoTraffico} = \{insieme\ di\ tutte\ le\ destinazioni\}$$

$$A = \{insieme\ dei\ punti\ di\ attraversamento\}$$

Definizione di Flusso

Flusso: un flusso di traffico è rappresentato come una coppia di indici ***(s,d)*** in cui: $s \in S \cup A$ è l'indice del punto di partenza del flusso (sorgente), mentre $d \in D \cup A$ è l'indice del punto di arrivo del flusso (destinazione).

La definizione di flusso appena fornita è molto generale e consente anche il collegamento di sorgenti e destinazioni di diverso tipo di traffico.

Flusso veicolare: un flusso ***(s,d)*** si dice veicolare di tipo $TipoTraffico \in T$ se $s \in S_{TipoTraffico}$ e $d \in D_{TipoTraffico}$. Ad esempio un flusso veicolare è di tipo ***Tram*** (flusso tramviario) se $s \in S_{Tram}$ e $d \in D_{Tram}$. Per ogni tipo di traffico (eccetto quello pedonale), possiamo quindi identificare l'insieme:

$$F_{TipoTraffico} = \{(s,d) \mid s \in S_{TipoTraffico}, d \in D_{TipoTraffico}\}$$

Considerando l'unione dei vari insiemi possiamo definire l'insieme dei ***flussi veicolari***, denotato dal simbolo:

$$F_v = \bigcup_{TipoTraffico \in T} F_{TipoTraffico}$$

Definizione di Flusso

Flusso pedonale: un flusso ***(s,d)*** si dice *pedonale* se $s, d \in A$ e sono adiacenti ad una stessa strada che ammette attraversamento. L'insieme dei flussi pedonali di un incrocio è denotato dal simbolo:

$$F_p = \{(s,d) \mid s, d \in A \text{ e sono adiacenti a una stessa strada che ammette attraversamento}\}$$

Cappio: Un flusso ***(s,d)*** si dice *cappio* se sia la sorgente che la destinazione appartengono alla stessa strada. L'insieme dei cappi è denotato dal simbolo:

$$F_c = \{(s,d) \mid s \text{ e } d \text{ appartengono alla stessa strada}\}$$

Flusso ammissibile: un flusso ***(s,d)*** si dice *ammissibile* se la sorgente e la destinazione sono dello stesso tipo di traffico e appartengono a strade diverse. Un flusso è *ammissibile* se non è un *cappio* ed è un *flusso veicolare* o *pedonale*. L'insieme dei flussi ammissibili è denotato dal simbolo:

$$F_a = (F_v \cup F_p) \setminus F_c$$

Definizione di Flusso

Flussi vietati: l'utente può decidere, in fase di inserimento degli input al problema, di vietare il transito di veicoli in determinati flussi. Tali flussi vengono detti *vietati*. Individuiamo, quindi, il seguente insieme:

$$F_x = \{(s,d) \mid (s,d) \text{ è } vietato\}$$

Flussi ammessi: un flusso ***(s,d)*** si dice *ammesso* se è un flusso ammissibile che non è stato esplicitamente vietato dall'utente. Definiamo quindi l'insieme dei flussi ammessi come:

$$F = F_a \setminus F_x$$

Semaforo: ogni flusso *ammesso* è regolato da un semaforo. Ogni semaforo regola un solo flusso *ammesso*. Il semaforo può avere due stati: rosso e verde. Se il semaforo associato al flusso ***(s,d)*** è rosso nessun veicolo può percorrere il flusso . Se invece è verde, possono percorrerlo.

Insiemi necessari per le collisioni

Nonostante queste definizioni, manca ancora da modellare la parte più problematica della tesina: le ***collisioni*** tra flussi.
Dato un flusso ***(s,d)*** possiamo definire questi 4 insiemi:

- $LS_{(s,d)}$: insieme di tutte le sorgenti che si trovano a sinistra (*Left Sources*) rispetto al flusso ***(s,d)***.
- $RS_{(s,d)}$: insieme di tutte le sorgenti che si trovano a destra (*Right Sources*) rispetto al flusso ***(s,d)***.
- $LD_{(s,d)}$: insieme di tutte le destinazioni che si trovano a sinistra (*Left Destinations*) rispetto al flusso ***(s,d)***.
- $RD_{(s,d)}$: insieme di tutte le destinazioni che si trovano a destra (*Right Destinations*) rispetto al flusso ***(s,d)***.

Definizione di Collisione e Confluenza

Collisione: un flusso ***(x,y)*** è in *collisione* con un altro flusso ***(s,d)*** se e solo se:

$x \in LS_{(s,d)} \wedge y \in RD_{(s,d)}$ oppure $x \in RS_{(s,d)} \wedge y \in LD_{(s,d)}$

Nel nostro modello abbiamo voluto anche distinguere la *collisione* con la *confluenza*, chiamando quest'ultima anche *collisione debole*.

Collisione Debole (confluenza): un flusso ***(x,y)*** è in *collisione debole (confluenza)* con un altro flusso ***(s,d)*** se e solo se:

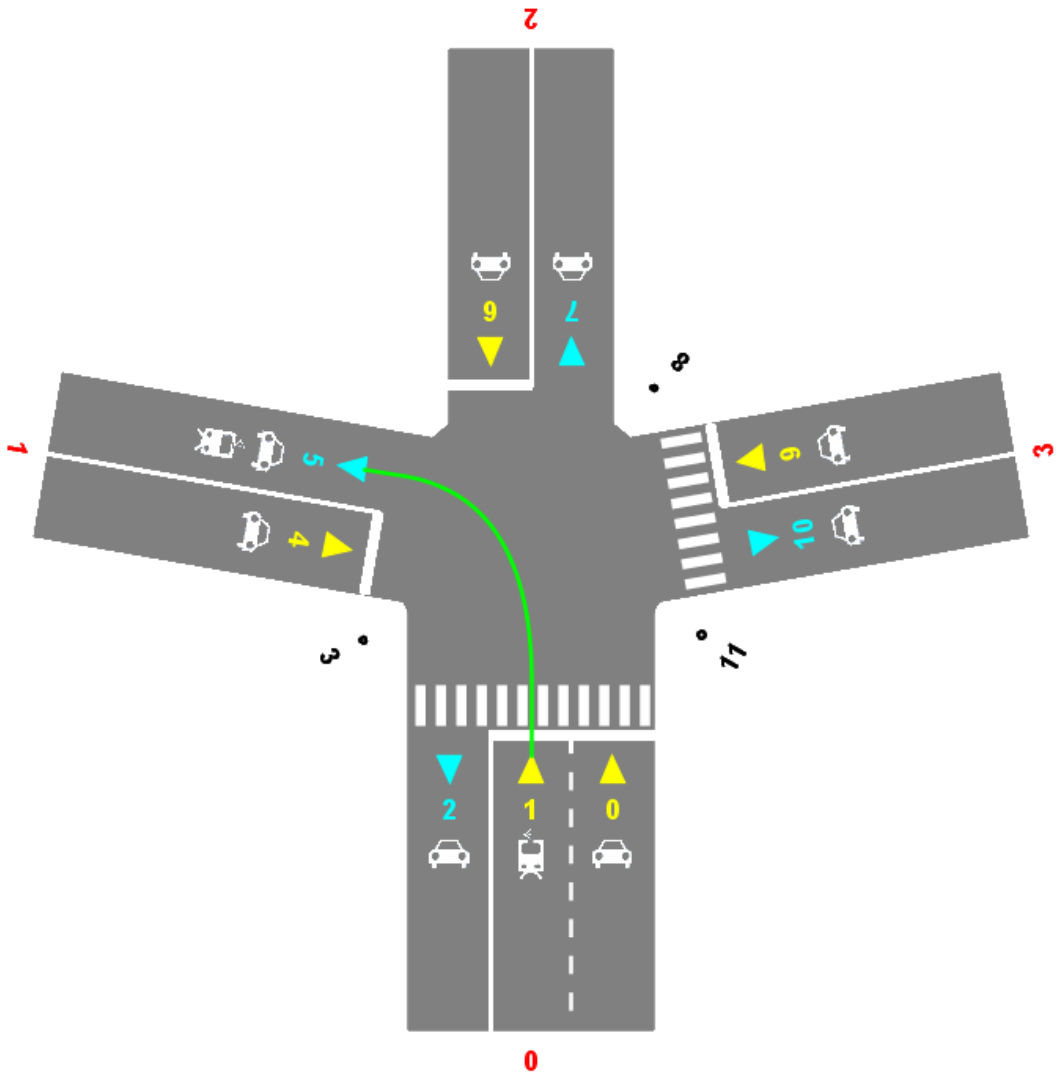
$x \neq s \wedge y = d$

La definizione di collisione appena descritta è piuttosto generale e soffre di due problemi:

- nel calcolo delle collisioni non vengono considerati i flussi pedonali;
- il calcolo delle collisioni genera anche flussi non ammissibili (cappi e flussi in cui sorgente e destinazione ammettono tipi di traffico diversi).

Questi problemi vengono superati intelligentemente dall'algoritmo che calcola le collisioni.

Esempio



S = {0, 1, 4, 6 ,9}
D= {2, 5, 7, 10}
A= {3, 8, 11}

$LS_{(1,5)}$={4}
$RD_{(1,5)}$={7, 10}
$RS_{(1,5)}$={0, 6, 9}
$LD_{(1,5)}$={2}

Collisioni con (1,5):
(4,7), (4,10)
(0,2), (6,2), (9,2)

Confluenze con (1,5):
(0,5), (4,5), (6,5), (9,5)

Algoritmo per il calcolo delle Collisioni

L'algoritmo in grado di superare i problemi evidenziati è il seguente.
Finché non sono stati considerati tutti i *flussi ammessi*:

1. si prende come riferimento un flusso ammesso che non sia già stato preso in precedenza (considerando anche i flussi pedonali, in quanto ammessi);
2. si utilizza la definizione di collisione per calcolare i flussi che collidono con quello di riferimento (qui invece, in base alla definizione, non vengono generati flussi pedonali);
3. ciascun flusso calcolato verrà considerato nel risultato solo se è ammesso.

Il fatto che i flussi pedonali possano essere presi come riferimento nel punto 1, ma non vengano considerati nel punto 2, **non è** una limitazione perché:

- due flussi pedonali non possono mai collidere tra loro, o meglio non vengono mai considerati collidenti;
- la collisione è una *relazione simmetrica* tra due flussi e pertanto se esprimo il fatto che f_1 *collide con* f_2, non c'è bisogno di esplicitare anche che f_2 *collide con* f_1 (sarebbe inutilmente ridondante).

Approfondimento algoritmo

Data la *simmetria* della relazione di collisione, quando l'algoritmo opera su *flussi veicolari*, si possono generare relazioni di collisioni ridondanti, perché già specificate. In questo caso il sistema tiene traccia di tutte le collisioni trovate e quindi evita di considerare collisioni già accertate, escludendo automaticamente i flussi già presi come riferimento in precedenza.

Un discorso a parte va fatto per le *confluenze*. Il sistema di default considera solamente le *collisioni "forti"*, non considerando quindi le confluenze. E' comunque possibile forzare il sistema, tramite un'apposita *checkbox* sull'interfaccia grafica, affinché consideri come collisioni anche le confluenze.

Attenzione: considerare le confluenze come collisioni complica il problema, dovendo imporre un numero maggiore di vincoli!

Traduzione in NuSMV

Per risolvere il problema dello *scheduling* di un incrocio, bisogna utilizzare un tipo di modellazione che permetta di rappresentare gli aspetti dinamici del sistema, legati al cambiamento dei semafori.

Per tale motivo, è stato quindi necessario utilizzare la logica temporale => utilizzo del Model Checker ***NuSMV***.

La traduzione del modello formale (così come è stato precedentemente presentato) in un file SMV viene effettuata in maniera automatica dal sistema.
Il file SMV risultante è costituito da un solo modulo (**MODULE main**), che contiene due sezioni principali:

- la sezione **VAR**;
- la sezione **LTLSPEC**.

Traduzione in NuSMV

Sezione VAR: contiene le variabili del sistema. Abbiamo una variabile binaria f_i per ogni *flusso ammesso*. Il valore della variabile rappresenta lo stato del semaforo associato al flusso: **0** significa che il semaforo è rosso, mentre **1** significa che il semaforo è verde.

Sezione LTLSPEC: contiene le formule in logica temporale che rappresentano i vincoli che devono essere rispettati dallo *schedule* risultante. Al fine di trovare uno *schedule*, è necessario negare l'AND delle formule che rappresentano i seguenti vincoli:

- Ciascun semaforo che regola un flusso di traffico, deve diventare verde un numero infinito di volte. Questo viene tradotto con una formula del tipo:

  $$G\ F\ f_i$$

- Se due flussi semaforici collidono tra loro, allora non deve mai essere possibile che i loro semafori siano contemporaneamente verdi. Supponiamo che il flusso f_1 collida con i flussi f_4, f_5 ed f_7. Avremo, dunque:

  $$G\ (f_1 \rightarrow !f_4\ \&\ !f_5\ \&\ !f_7)$$

Traduzione in NuSMV

Sezione LTLSPEC

- Ciascun semaforo deve rispettare gli eventuali vincoli di ***durata minima del verde***, imposti dall'utente. Un vincolo di durata minima del verde pari a ***n*** istanti ci impone che in ogni istante, se il semaforo è rosso e diventa verde nell'istante successivo (primo istante verde), allora sarà verde anche negli ***n-1*** istanti successivi al prossimo (per un totale appunto di almeno ***n*** istanti verdi). Se supponiamo che il flusso f_i abbia un vincolo di almeno 3 istanti verdi, allora avremmo una formula del tipo

$$G\,(!\,f_i\,\&X\,f_i \rightarrow X\,X\,f_i\,\&X\,X\,X\,f_i)$$

- Ciascun semaforo deve rispettare gli eventuali vincoli di ***durata massima del rosso***, imposti dall'utente. Un vincolo di durata massima del rosso pari a ***n*** istanti, ci impone che in ogni istante, se il semaforo è rosso ora e negli ***n-1*** istanti successivi, allora nell'***n-esimo*** istante successivo sarà verde. Se supponiamo che per il rosso abbiamo un vincolo di durata di almeno 5 istanti, allora si avrà una formula del tipo

$$G\,(!\,f_i\,\&X\,!\,f_i\,\&X\,X\,!\,f_i\,\&X\,X\,X\,!\,f_i\,\&X\,X\,X\,X\,!\,f_i \rightarrow X\,X\,X\,X\,X\,f_i)$$

Esecuzione di NuSMV

Il file così generato viene dato in pasto a ***NuSMV***, al fine di trovare uno *schedule* che soddisfi tutti i vincoli.

Problema: con il *Symbolic Model Checking*, già con incroci modesti per i quali sono stati impostati vincoli temporali, ***NuSMV*** potrebbe non terminare in tempo ragionevole, occupando anche più di 2GB di RAM.

Soluzione: si può utilizzare il *Bounded Model Checking* con lunghezza di cammino sufficientemente grande.

Abbiamo verificato empiricamente che se la soluzione c'è, è composta da un numero di stati non troppo elevato. Impostando, quindi, un limite massimo (bound) pari a ***30~40***, si può ragionevolmente assumere che se il bounded model checking non trova una soluzione, allora molto probabilmente non esiste nessuna soluzione interessante al problema.

Il sistema permette di:

- scegliere se utilizzare la modalità symbolic o bounded;
- fissare una *durata massima* per la computazione.

Esecuzione di NuSMV

Problema: anche utilizzando il *Bounded Model Checking* spesso non si riesce ad ottenere uno *schedule*, per via dei vincoli troppo stringenti.

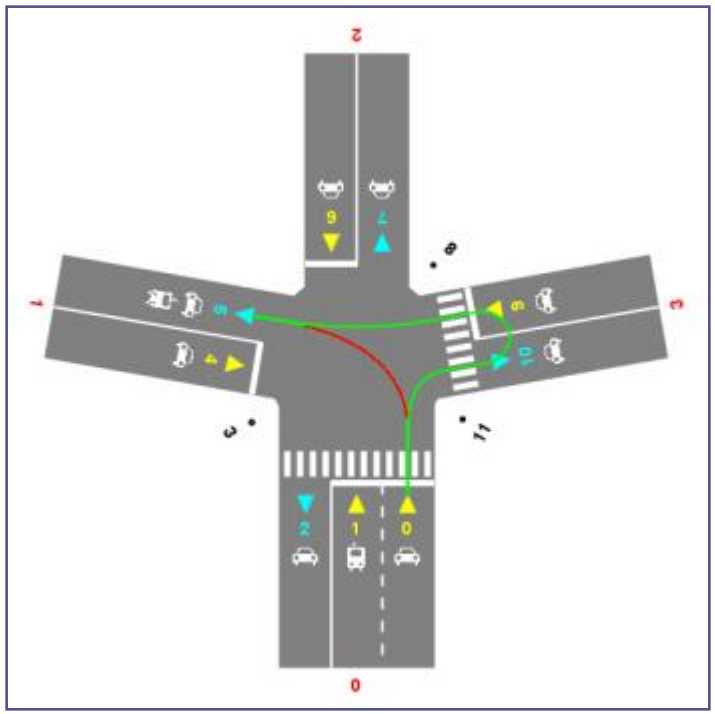
Soluzione: le possibili strategie per risolvere questo problema sono due:

- rilassare i vincoli temporali sui flussi;
- ridurre il numero di collisioni.

La ***prima strategia*** è lasciata esclusivamente a carico dell'**utente**, che dovrà rilassare i vincoli di durata e cioè: *ridurre la durata minima del verde* e/o *aumentare la durata massima del rosso*.

Anche la ***seconda strategia*** può essere intrapresa dall'utente, vietando alcuni flussi "critici" o deselezionando l'opzione che considera le confluenze come collisioni. Oltre a questo, però, la seconda strategia viene anche messa in pratica automaticamente dal **sistema**, attraverso lo ***spezzamento dei flussi***.

Spezzamento di un Flusso



Flusso spezzato: Dato un flusso ***(s,d)***, esso può essere spezzato in due flussi ***(s,a)*** e ***(b,d)***, dove ***a*** e ***b*** sono due corsie che appartengono ad una stessa strada di riferimento tale che:

- permetta l'inversione di marcia
- sia situata a destra del flusso ***(s,d)***.

Indichiamo lo spezzamento con la notazione:

$$(s,d) = (s,a) + (b,d)$$

o, utilizzando le variabili relative ai flussi:

$$f_i = f_j + f_k$$

L'operazione di spezzamento elimina uno o più flussi e le collisioni relative ad essi semplificando il problema ed aumentando, quindi, la probabilità che esista una soluzione per esso.

Spezzamento di un Flusso

Nel **sistema**, vengono provati ***tutti*** *i possibili spezzamenti di un flusso o di due flussi* (vengono provate tutte le combinazioni possibili). Il sistema si ferma al massimo a due flussi perché andando oltre si avrebbe un'esplosione delle combinazioni, tale che il calcolo risulterebbe troppo oneroso e comunque poco interessante.
Come ulteriore ottimizzazione, il sistema prova a spezzare i flussi in modo *ordinato (decrescente) rispetto al numero di collisioni*.

A seguito di uno spezzamento, invece di creare un file NuSMV ex-novo, adeguando le varie strutture dati e ricalcolando il tutto, abbiamo preferito ***modificare il file NuSMV*** precedentemente generato.
Applicare le modifiche direttamente sul codice NuSMV offre due vantaggi:

- è più semplice e meno oneroso rispetto alla creazione ex-novo;
- rende molto semplice lo spezzamento di ulteriori flussi.

Modifica del file NuSMV

Supponendo di effettuare lo spezzamento $f_i = f_j + f_k$, avremo le seguenti modifiche:

Sezione VAR

- viene ***rimossa*** la variabile f_i

Sezione LTLSPEC

- il vincolo $G\ F\ f_i$ viene ***sostituito*** da $G\ F\ (f_j\ \&\ f_k)$
- vengono ***rimossi*** i vincoli di collisione in cui f_i appare nel termine a sinistra dell'implicazione oppure appare come unico letterale presente nel termine a destra
- in ogni altro vincolo di collisione che non ricade nel caso precedente viene ***rimosso*** dal termine a destra dell'implicazione qualsiasi riferimento a f_i
- nel vincolo di durata minima del verde relativo al flusso f_i, la variabile f_i viene ***sostituita*** da $(f_j\ \&\ f_k)$
- nel vincolo di durata massima del rosso relativo al flusso f_i, la variabile f_i viene ***sostituita*** da $(f_j\ \&\ f_k)$

Implementazione in Java

JTrafficControl è totalmente implementato in Java. Le caratteristiche principali dell'applicazione sono:

- Scritta per poter funzionare con JDK 1.5, come progetto di NetBeans.
- E' composta da 4 package: struttura, motore, grafica, icone.
- Permette di disegnare un incrocio aggiungendo (eliminando) strade e corsie.
- Permette il salvataggio dell'incrocio, per poterlo caricare in futuro.
- Permette di esprimere tutti i vincoli offerti dal modello e calcola in automatico lo spezzamento di uno o più flussi.
- L'esecuzione di NuSMV è del tutto mascherata all'utente ed è implementata in un motore multi-threading.