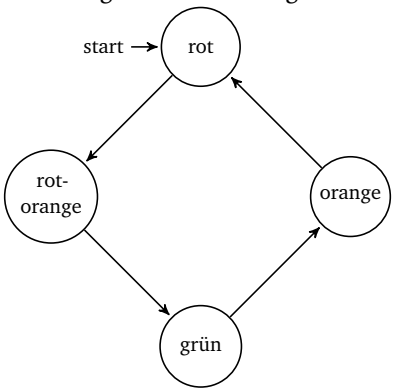
AUFGABE 5: WP-KALKÜL – FRAMA-C

In dieser Aufgabe werden Sie die Korrektheit zweier Programme mit Hilfe von Frama-C und des WP-Kalküls nachweisen.

Aufgabenstellung

1. *Ampel-Implementierung:* Implementieren Sie die vorgegebene Schnittstelle der Ampelschaltung und beachten Sie die Dokumentation der Schnittstelle bei der Implementierung. Die Details des grundlegenden Datentyps `TLight` sind bewusst offengelassen. Die Implementierung der Funktion `tlight_set_next_phase()` soll den Zustandsautomaten in folgender Abbildung umsetzen:

☞ `make doxy`



Wie Sie dem Zustandsgraphen entnehmen können, besitzt die Zustandsmaschine vier Zustände und jeder dieser Zustände hat genau einen Folgezustand. *Wie können Sie erzwingen, dass Ihr Datentyp nur diese vier Zustände annehmen kann? Wie können Sie sicherstellen, dass nur legale Zustandsübergänge auftreten? Achten Sie bei Ihrer Implementierung auf Typsicherheit!*

2. *Ampel-Verifikation:* Annotieren Sie nun die Schnittstelle Ihres Ampel-Programms in ACSL mit je mindestens einer nicht-trivialen `assigns`-, `ensures`- und `requires`-Aussage. *Fassen Sie diese Aussagen so streng wie möglich.* Sofern sinnvoll, benutzen Sie wiederverwendbare Prädikate. Weisen Sie die Korrektheit der Implementierung mit Hilfe des WP-Plugins von Frama-C nach.

3. *Vollständige Kreuzung:* Erweitern Sie nun Ihr Programm so, dass eine vollständige Kreuzung mit vier Ampeln simuliert wird. Die Kreuzung soll initialisierbar sein und in der Lage sein in den nächsten Zustand überzugehen. Hierbei darf nie eine Situation eintreten, die Verkehrsteilnehmer unnötig gefährdet. Sehen Sie hierbei auch eine Möglichkeit für die Kreuzung vor, in einen *sicheren Grundzustand* zurückzukehren und von diesem aus den normalen Betrieb wiederaufzunehmen. Weisen Sie die Korrektheit Ihres Programms mit Hilfe von Frama-C nach.

4. `lower_bound()`-*Verifikation:* Bestimmen Sie den Vertrag, die Schleifen*invarianten* und die Schleifen*variante* der Funktion `lower_bound()` und weisen Sie deren Korrektheit in Frama-C nach. Die Implementierung und die Annotationsrümpfe finden Sie in der aktuellen Vorgabe.

```
1  unsigned int lower_bound(const int *a,
2                           unsigned int n,
3                           int val) {
4    unsigned int left = 0;
5    unsigned int right = n;
6    unsigned int middle = 0;
7
8    while (left < right) {
9      middle = left + (right - left) / 2;
10     if (a[middle] < val) {
11       /*@ assert \forall integer i; 0 <= i < middle + 1
12                                     ==> a[i] < val;
13        */
14       left = middle + 1;
15     } else {
16       right = middle;
17     }
18   }
19
20   return left;
21 }
```

Hinweise

- Bearbeitung: Gruppe mit je zwei bis drei Teilnehmern.
- Abgabezeit: 03.07.2014
- Fragen bitte an `i4ezs@lists.informatik.uni-erlangen.de`