SUMO - More on... Traffic Lights

Daniel Krajzewicz

Danilot Tete Boyom

SUMO - More on... Traffic Lights

Daniel Krajzewicz
Danilot Tete Boyom

Table of Contents

1. Introduction 1
Document Information 1
Document Scope 1
Document Audience 1
Document Structure 1
2. Right-Of-Way Rules in SUMO 2
Links: A Connection between Lanes 2
Usage of Links within the Simulation 2
Setting Information by MSLogicJunctions 2
Switching the Phases 3
The whole procedure 3
3. Existing TLS-Algorithms in SUMO 5
Static TLS (MSSimpleTrafficLightLogic) 5
4. Implementing Own TLS-Algorithms 6
Existing APIs 6
5. Loading of TLS- and ROW-logics 7
XML-Representation 7
Description of TLS-Logics 7
Connections between Streams and TLS-Logics 9
Parsing the network description 10
6. Further Information 11
Questions & Answers 11
Changing the duration of Cycles and Phases 11
Coupling Traffic Lights 11

List of Figures

2.1. Junction classes within SUMO 2
2.2. TLS classes within SUMO 3

List of Tables

5.1. Meanings of combinations of phase/brake/yellow 8

Chapter 1. Introduction

Document Information

This document is a part of *SUMO* - *S*imulation of *U*rban *MO*bility. SUMO is an open source microscopic road traffic simulation. It can be found under http://sumo.sourceforge.net/.

This document fits to version 0.9.5. We will try to keep it up to date with the next versions.

Document Scope

Herein, we will describe how traffic lights are implemented and used within SUMO. This document has the following structure:

1. As basically traffic lights affect the right-of-way rules at a junction, we will first describe how right-of-way rules are implemented within SUMO and how traffic lights are implemented as an extension to this concept.
2. Algorithms already implemented in SUMO are described and it is shown how they work.
3. An introduction about how to implement own traffic light algorithms is given.
4. It is described, how traffic light logics and row-logics are loaded and instantiiated.

Document Audience

This document contains information for *users* who are interested how traffic light systems are implemented in SUMO and who need further information on how to define own tls programs. It is also meant for *developers* who want to extend SUMO.

Document Structure

At first, a description about how right-of-way rules are applied and how traffic lights work in SUMO is given.

Then, tls-algorithms already implemented in SUMO are described *(TBD: incomplete)*.

Then, it is shown how one can implement own tls algorithms *(TBD: incomplete)*.

Then it is described how right-of-way rules and tls-algorithms are loaded.

We close with some further information about how to deal with tls within SUMO. These information are mainly answers to questions posted by users.

Chapter 2. Right-Of-Way Rules in SUMO

Links: A Connection between Lanes

Each lane (`MSLane`) beside dead ends has a list of at least one following lane. A link (MSLink) connects this following and the lane we regard. Basically, we consider two types of connections between lanes. In the first case, vehicles are able to pass the connection every time, without regarding other, possibly crossing traffic. Such connections may be found at highways or if the connection is a primary road and it is assured that the vehicle is able to pass this link without a collision. Such connections are called "*priorised*" connections. The other connections are called "*unpriorised*". They are found on minor roads and a vehicle passing them must decelerate in front of them, because it may have to stop to let other vehicles pass the junction first. Please regard, that a junction may consist of several connections (links) of different type.

Usage of Links within the Simulation

If a vehicle approaches a link (`MSLink`), it lets the link know about it by calling `void MSLink::setApproaching(MSVehicle *approaching)`. This is done in `void MSVehicle::vsafeCriticalCont( SUMOReal boundVSafe )` as soon as a vehicle will have to decelerate to manage to stop in front of the link. Even if the vehicle is too near or too fast to decelerate in front of the link, the link will still be informed until the vehicle will finally pass it. By doing this, each link knows whether a vehicle is approaching or not and stores this vehicle in `MSLink::myApproaching`. In addition, the information about approaching a link is stored by the link into the `MSLogicJunction::Request` structure of the junction (`MSRightOfWayJunction`) it belongs to. After all vehicles have stored this information, `MSRightOfWayJunction` is able to compute which of the approaching vehicles will have to wait and which not using its `MSJunctionLogic`.

If you are confused about all these classes, the next diagram which shows how the hierarchy of SUMO junction classes, may help you.

Note

Remark that some of the classes are interfaces with only one implementation. It is possible, that this tree is a matter to change in the next time.

Figure 2.1. Junction classes within SUMO

In the next pass (but within the same simulation step), each of the vehicles is touched again, but by now it is known whether the vehicle is allowed to pass the junction using his link or not. If not, the vehicle will decelerate and stop in front of the link (or a vehicle in front of it, that is also waiting). If the link is "unpriorised", the vehicle will decelerate in front of it, even if it may pass it. If the link is "priorised" and the vehicle may pass, the vehicle will continue his drive without being hindered.

Setting Information by MSLogicJunctions

As we have seen, a MSLogicJunction gives way to the vehicles after the knowledge about all incoming vehicles has been set. We have seen that this information was stored in a structure named "`MSLogicJunction::Request`". This is simply a list of bits, a `std::bitset<64>`, where each bit represents the information whether a certain link is approached by a vehicle currently (it's then set to true otherwise it's false).

What traffic lights do is simply to mask off those requests which are approaching a currently red sign. By doing this, the response for these streams is also set to false which forces the approaching vehicles to stop in front of the junction. Furthermore, this method keeps the row-logic of the junction untouched, what allows to use it for letting vehicles moving left - in the part of the world that drives on the right sied, that means those streams which have to cross an enemy stream - wait until there is a gap in the enemy stream they can use to move over the junction.

Figure 2.2. TLS classes within SUMO

Switching the Phases

What we furthermore need to know is how a traffic light is forces to switch its phases and how the links get informed about this. When being instantiated, the tls adds an event to SUMO's event handler (`MSEventHandler`). This is done within the `MSTrafficLightLogic`-constructor, a class from which all tls algorithms are derived (see above). This event, being a `MSTrafficLightLogic::SwitchCommand` executes the following steps:

```
SUMOTime execute() {
   // get the current phase index
   size_t step1 = myTLLogic->getStepNo();
   // try to switch, get next duration
   SUMOTime next = myTLLogic->trySwitch();
   // get the phase index after possible switch
   size_t step2 = myTLLogic->getStepNo();
   // has the phase changed?
   if(step1!=step2) {
      // yes -> do some further actions (see below)
      myTLLogic->onSwitch();
      // inform the links about current state
      myTLLogic->setLinkPriorities();
   }
   // return period after which the tls shall be tried to switch again
   return next;
}
```

Let's review it. At first, we remember the state of the tls before trying to switch it in the variable `step1`. This "state" is simply the index of the current tls phase. Then, we try to switch the tls using `trySwitch()` and gain the period to the next try which we save in `next`. After this, we check whether the tls has really changed by asking for its current step. If the step before and the step after trying to switch the tls are not the same, the tls has switched. We then inform the links about it using the `MSTrafficLightLogic::setLinkPriorities` function. The call to `onSwitch` should not bother you, herein. It is not a part of the tls-steering, but used to inform connected structures about the phase change. By now, only assigned detectors may be informed and their assignment is done separately from the tls-implementations. The comparison between the step before and after trying to switch is needed, as maybe the tls decides not to switch, yet, although the phases duration is over *(TBD: add a description of agentbased tls)*. After all has been done, we return the period to wait until we again try to switch the tls. If you want, your implementation of "`trySwitch`" may always return 1 in order to check the tls each simulation step.

The whole procedure

So the whole procedure of bringing a vehicle save over a junctions is as following:

1. `MSNet::myJunctions->resetRequests();`

   This resets all junctions' requests to false, that means, that no vehicles from previous steps may inference with the current network state

2. ... (several other things may happen, here)
3. `MSNet::myEdges->moveCritical();`

   Herein, vehicles which approach a junction set their request
4. `MSNet::myLogics->maskRedLinks();`

   Requests for vehicles which have red are reset to false
5. `MSNet::myJunctions->setAllowed();`

   The junctions compute which vehicles may pass them and which have to wait
6. `MSNet::myLogics->maskYellowLinks();`

   Reallow vehicles trying to pass a yellow tls to pass the junction
7. `MSNet::myEdges->moveFirst();`

   Move all those vehicles (not only the first one, but all first ones) that approach a junction

What is the meaning of the sixth step? Imagine what shall happen with vehicles driving over yellow. They should stop, but if they are too near to the junction, one has to let them through. Also, they have to be taken into account if an enemy stream exists. In this case, they maybe have to make vehicles on this stream wait. To achieve this, at first only red links are set to false. This is given as input to the junction's row-logics and that lets all vehicles having red wait, vehicles which have yellow may be able to pass the junction. Then, vehicles having yellow are set to false which lets them try to stop in front of the tls.

Chapter 3. Existing TLS-Algorithms in SUMO

Static TLS (MSSimpleTrafficLightLogic)

This tls-type simply obtains the list of phases from the description and switches between the cycles after each cycle's period is over. We have seen that the tls algorithm is stored in `trySwitch()`. What a simple traffic light's implementation does is the following:

```
SUMOTime
MSSimpleTrafficLightLogic::trySwitch()
{
    // increment the index
    myStep++;
    // if the last phase was reached ...
    if(myStep==myPhases.size()) {
        // ... set the index to the first phase
        myStep = 0;
    }
    // return offset to the next switch
    assert(myPhases.size()>myStep);
    return myPhases[myStep]->duration;
}
```

That means that we simply rotate our index over the phases and return each phase's duration as the time that has to pass before the traffic light is switched one step further.

Chapter 4. Implementing Own TLS-Algorithms

Existing APIs

This tls-type simply obtains the list of phases from the description and switches between the cycles after each cycle's period is over. We have seen that the tls algorithm is stored in `trySwitch()`. What a simple traffic light's implementation does is the following:

```
SUMOTime
MSSimpleTrafficLightLogic::trySwitch()
{
    // increment the index
    myStep++;
    // if the last phase was reached ...
    if(myStep==myPhases.size()) {
        // ... set the index to the first phase
        myStep = 0;
    }
    // return offset to the next switch
    assert(myPhases.size()>myStep);
    return myPhases[myStep]->duration;
}
```

That means that we simply rotate our index over the phases and return each phase's duration as the time that has to pass before the traffic light is switched one step further.

Chapter 5. Loading of TLS- and ROW-logics

Within the next subchapters, we will describe what all these information means and how it is parsed.

XML-Representation

Description of TLS-Logics

Normally, the descriptions of traffic lights are stored within the network descriptions and look like this (example from `cross3ltl.net.xml`):

```
<tl-logic type="static">
   <key>0</key>
   <subkey>0</subkey>
   <phaseno>8</phaseno>
   <offset>0</offset>
   <phase duration="20" phase="0000111100001111" brake="1111110011111100" \
      yellow="0000000000000000"/>
   <phase duration="4" phase="0000110000001100" brake="1111111111111111" \
      yellow="0000001100000011"/>
   <phase duration="3" phase="0000110000001100" brake="1111001111110011" \
      yellow="0000000000000000"/>
   <phase duration="4" phase="0000000000000000" brake="1111111111111111" \
      yellow="0000110000001100"/>
   <phase duration="20" phase="1111000011110000" brake="1100111111001111" \
      yellow="0000000000000000"/>
   <phase duration="4" phase="1100000011000000" brake="1111111111111111" \
      yellow="0011000000110000"/>
   <phase duration="3" phase="1100000011000000" brake="0011111100111111" \
      yellow="0000000000000000"/>
   <phase duration="4" phase="0000000000000000" brake="1111111111111111" \
      yellow="1100000011000000"/>
</tl-logic>
```

This description is build during network conversion/generation within the netbuilding classes.

The meaning of the tags and the information stored within them is the following:

- `tl-logic`: defines the begin of the description of a tls-logic

  Attributes: `type="static|actuated|agentbased"`; describes which tls-algorithm shall be used by this tls (string-enum)

  Information in CDATA: none

- `key`: defines the begin of the description of a tls-logic

  Attributes: none

  Information in CDATA: the key (name) of the tls (string)

- `subkey`: the subkey of the logic; different programs for the same junction must be distinct; used since version 0.9.4

  Attributes: none

  Information in CDATA: a name (subid) of the logic

- `phaseno`: the number of phases this tls-logic has

  Attributes: none

  Information in CDATA: the number of phases of the logic (int)

- `offset`: the offset by which the tls shall change to next phase

  This allows you to "rotate" the red/green phases. The offset is a delay to the simulation begin given in seconds with which the tls shall change to the next phase for the first time.

  Attributes: none

  Information in CDATA: the offset in seconds (int)

- `phase`: definition of a phase (see below)

Please remark, that since version 0.9.4 there is no `inclanes`-tag within a program's description and that "`logicno`" has been renamed to "`subkey`".

The definitions of a phase should be investigated more deeply. Each phase has an own definition and the number of such definitions should be equal to the value stored in `phaseno`. The mandatory attributes store the following information:

- `duration`: the duration of the phase in seconds
- `phase`: a bitset which holds the information which streams have a "green" light (1). Vehicles within these streams are allowed to pass the junctions, vehicles within streams that have a zero (0) here, have either "red" light in front or a "yellow" one
- `brake`: a bitset which holds the information which streams have to decelerate in front of the junction. Vehicles within a stream may have to decelerate either because they have a red or yellow light or because a foe stream has green, to (this is mostly the case for left-turning streams). Streams that have to decelerate are marked with a 1, here, zero (0) means the vehicles may pass without decelerating in front of the tls
- `yellow`: a bitset which holds the information which vehicles are in front of yellow lights (1).

Caution

The indices of "`phase`", "`brake`" and "`yellow`" are from right to left, that means the rightmost (last) entry will be later found at index 0.

The next table shows again the meaning of phase/brake/yellow combinations:

Table 5.1. Meanings of combinations of phase/brake/yellow

phase	brake	yellow	meaning
0 (may not pass)	0 (does not have to brake)	0	may not pass, but does not have to brake--> invalid!!
0 (may not pass)	0 (does not have to brake)	1	may not pass, but does not have to brake --> invalid!!
0 (may not pass)	1 (must brake)	0	the vehicle must brake in front of the tls to regard other traffic and must not pass it --> equivalent to a red light
0 (may not pass)	1 (must brake)	1	the vehicle must brake in

phase	brake	yellow	meaning
			front of the tls to regard other traffic and must not pass it, but the "yellow" value is set, so if the vehicle is too fast to stop in front, it will be passed --> equivalent to a yellow light
1 (may pass)	0 (does not have to brake)	0	the vehicle may pass the junction without braking (regarding other traffic) --> equivalent to green light for streams that may pass the junction without regarding other traffic (higher priorised roads)
1 (may pass)	0 (does not have to brake)	1	invalid combination
1 (may pass)	1 (must brake)	0	the vehicle may pass the junction but has also to slw done in front of the tls to let vehicles on roads with a higher priority pass if any --> equivalent to green light for streams that have to regard other traffic (lower priorised roads)
1 (may pass)	1 (must brake)	1	invalid combination

Also some further non-mandatory attributes may be stored within the `phase`-tag. until now, these tags are used by the actuated and the agentbased tls only. They are:

- `min_duration`: a minimum duration for the phase. The tls-logic may not shorten the duration of the phase below this value (int, in seconds)
- `max_duration`: a maximum duration for the phase. The tls-logic may not extend the duration of the phase beyond this value (int, in seconds)

There is no possibility to insert values for `min_duration` and `max_duration` into the network descriptions by now but by hand.

Connections between Streams and TLS-Logics

You probably have seen, that the `inclanes-tag` is buggy. In fact, the links (connections between two consecutive lanes, see above) are added almost at the end of the loading process. When a link is built, the tls-logic which controls it is informed about it getting the information which stream information is used by the link. This information is given as an index to the fields `phase`, `brake` and `yellow` from the `phase`-element of the tls-description described above.

The information about a link is represented as following (example from `cross3ltl.net.xml`):

```
<succ edge="1si" lane="1si_0" junction="0">
   <succlane lane="3o_0" via=":0_12_0" \
      tl="0" linkno="12" yield="1" dir="r" state="t"/>
</succ>
```

Let's consider the element "succlane" only, the element "succ" gives us only the names of the edge/

lane that is currently processed and the junction this edge/lane yields in.

- `lane`: the connected lane's id (mandatory, string)
- `via`: the connected internal lane's id (mandatory, string)

Caution

internal lanes are not yet supported, this is just a developer's information by now

- `tl`: the id of the tls-logic that is responsible for this link; this field is missing if the link is not controlled by a tls
- `linkno`: the index of the information within the controlling tls logic that shall be used for this link
- `yield`: information whether vehicles have to wait in front of this link (valid only for uncontrolled links)
- `dir`: the information about which arrow shall be drawn for this link
- `state`: the information what kind of state shall be drawn

Parsing the network description

An instance of NLHandler is used to parse the XML-descriptions of the network and of the tls-logics contained herein. All elements related to traffic lights and junctions at all are delegated to an instance of NLJunctionControlBuilder (from version 0.9.0 on). Herein, the whole description is stored temporarily until the complete description has been read. Then, as the closing `tl-logic` tag occures, the tls logic is build in accordace to the read type.

Chapter 6. Further Information

You can find some further questions posted by users herein.

Questions & Answers

Changing the duration of Cycles and Phases

Question: I want to study the effects of changing the periods of the traffic light. (time green left/straight/right in both directions and the "dead time" = all directions have red); how do I change the period of the traffic light?

Answer: If you have a network that contains traffic lights, you can open it with a text editor. Then search for "tl-logic". You will find the definition(s) of your traffic light(s). Now, you can see a set of "phase"-definitions within each of the traffic lights definitions. You can change the time for each phase directly in here.

Coupling Traffic Lights

Question: Is it possible to "connect" two traffic lights? In other words, is it possible to say that one light has to become green x seconds after the other?

Answer: Is it possible to "connect" two traffic lights? In other words, is it possible to say that one light has to become green x seconds after the other?