Grammar for the English to LTL Parser

May 2011

There are four types of sentences that can be written in this grammar: initial conditions, environment assumptions, motion/action requirements and conditional sentences. None of these sentences is required for the simulation - the parser has a default for every part of the LTL formula φ (see Section **??**).

All of these sentences must be constructed using the following templates and the list of valid propositions.

For this grammar we define:

> **Valid propositions** := sensor propositions, region names, robot actions and auxiliary propositions defined by the user.
>
> ϕ := Boolean Formula composed from the valid propositions and boolean connectives (parentheses may be added)
>
> $$\phi ::= \text{a valid proposition} \mid \text{not } \phi \mid \phi \text{ or } \phi \mid \phi \text{ and } \phi \mid \phi \text{ implies } \phi \mid \phi \text{ iff } \phi$$
>
> ϕ_{env} := Boolean formula composed **only from valid sensor propositions** and boolean connectives (parentheses may also be added)
>
> ϕ_{region} := Boolean formula composed **only from valid region names** and boolean connectives (parentheses may also be added)
>
> ϕ_{action} := Boolean formula composed **only from robot actions and auxiliary propositions** and boolean connectives (parentheses may also be added)
>
> ϕ_{robot} := Boolean formula composed from all valid propositions **except the sensor propositions** and boolean connectives (parentheses may also be added)

Note that the following templates, except for the propositions, are *not* case sensitive. Furthermore, any line starting with **#** is considered a comment and is ignored.

1 Initial Conditions

Valid initial condition sentences for the environment are:

- "Environment starts with ϕ_{env}" - ϕ_{env} is the initial condition for the sensor propositions
- "Environment starts with true" - all sensor propositions are initially set to true
- "Environment starts with false" - all sensor propositions are initially set to false

Valid initial condition sentences for the robot are:

- "Robot starts in ϕ_{region}" - ϕ_{region} sets the possible initial regions for the robot. If this sentence is omitted, the robot can start in any defined region.

- "Robot starts with ϕ_{action}" - ϕ_{action} sets the initial values of the robot actions and the auxiliary propositions. Propositions that are not part of ϕ_{action} can be initially either true or false.
- "Robot starts with true" - all robot actions and auxiliary propositions are initially set to true
- "Robot starts with false" - all robot actions and auxiliary propositions are initially set to false
- "Robot starts in ϕ_{region} with ϕ_{action}"
- "Robot starts in ϕ_{region} with true"
- "Robot starts in ϕ_{region} with false"

Note: Environment may be replaced with "Env" and "Robot starts" may be replaced with "You start".

2 Environment Assumptions and Motion/Action Requirements

We distinguish between two types of assumptions/requirements: **Safety** and **Liveness**. In the following, for environment assumptions $\varphi = \phi_{env}$ and for robot requirements $\varphi = \phi_{robot}$. This means that the sensor and robot propositions **cannot** be mixed in these basic instructions.

2.1 Safety

Safety assumptions and requirements specify behaviors that must always be met, for example (in natural language): "Never enter the haunted room with a cake".
Valid and equivalent safety sentences are:

- "Always φ"
- "Always do φ"
- "Do φ"

$\rightarrow$ LTL equivalent: $\Box\,\varphi$

2.2 Liveness

Liveness assumptions and requirements specify goals that always must eventually be met, thus describing situations that must be reached infinitely often. For example (in natural language) "Go to the haunted room infinitely often".
Valid and equivalent liveness sentences are:

- "Go to φ"
- "Visit φ"
- "Infinitely often do φ"
- "Infinitely often φ"

$\rightarrow$ LTL equivalent: $\Box\Diamond\,\varphi$

2.3 Go To and Stay

We allow one more type of robot requirement, **Go To and Stay**:

- "Go to ϕ_{robot} and stay there"
- "Go to ϕ_{robot} and stay"

While we allow ϕ_{robot}, the intended use of this sentence is for statements such as "Go to the haunted room and stay there."

$\rightarrow$ LTL equivalent: $(\Box \Diamond \varphi) \wedge \Box \left(\varphi \rightarrow \bigwedge_{\phi_i \in RP} [\bigcirc \phi_i \Leftrightarrow \phi_i] \right)$, where RP is the set of region propositions.

3 Conditionals

There are three types of conditional sentences that are allowed:

- "If *condition* then *requirement*"
- "*requirement* unless *condition*"
- "*requirement* if and only if *condition*"

$\rightarrow$ LTL equivalent: For a requirement $\Delta\phi_r$, where $\Delta \in \{\Box, \Box \Diamond\}$, and condition ϕ_c, the resulting LTL formulas will be $\Delta\,(\phi_c \rightarrow \phi_r)$, $\Delta\,(\neg\phi_c \rightarrow \phi_r)$, and $\Delta\,(\phi_c \Leftrightarrow \phi_r)$, respectively.

3.1 Requirements

A requirement is a either a **safety** or a **liveness** sentence. As such, it should contain either sensor propositions or robot propositions, but not both.
A requirement can also be "**Stay there**" or "**Stay**".

3.2 Conditions

Conditions are built up by combining basic conditions using *and* and *or* (which are also case insensitive). The structure of a basic condition is "*Phrase* ϕ", where *Phrase* is one of the following:

	Positive	**Negative**
Past	'The robot was in'	'The robot was not in'
	'You were in'	'You were not in'
	'It was in'	'It was not in'
	'Were in'	'Were not in'
	'Was in'	'Was not in'
	'The robot sensed'	'The robot did not sense'
	'You sensed'	'You did not sense'
	'You were sensing'	'You were not sensing'
	'It sensed'	'It did not sense'
	'Sensed'	'Did not sense'
	'The robot activated'	'The robot did not activate'
	'You activated'	'You did not activate'
	'You were activating'	'You were not activating'
	'It activated'	'It did not activate'
	'Activated'	'Did not activate'
Present	'The robot is in'	'The robot is not in'
	'You are in'	'You are not in'
	'It is in'	'It is not in'
	'Are in'	'Are not in'
	'Is in'	'Is not in'
	'The robot is sensing'	'The robot is not sensing'
	'You are sensing'	'You are not sensing'
	'It is sensing'	'It is not sensing'
	'Are sensing'	'Are not sensing'
	'Is sensing'	'Is not sensing'
	'The robot is activating'	'The robot is not activating'
	'You are activating'	'You are not activating'
	'It is activating'	'It is not activating'
	'Are activating'	'Are not activating'
	'Is activating'	'Is not activating'

A few notes about the tenses:

- Tenses are used to decide whether the 'next' operator ($\bigcirc$) needs to be attached to a proposition. Due to the structure of the LTL formulas we consider, the 'next' operator is *not* used in liveness requirements, therefore, both the past and present tenses are the same for such propositions.
- For safety requirements (including 'stay'), the past tense relates to formulas that are true in the current state while the present tense relates to formulas that should be true in the next state.
- For environment safety (again, due to the structure of the formula and the nature of the synthesis algorithm) all robot propositions *must* be in the past tense. If they are not, the parser will translate them into the past tense and issue a warning.
- One should keep in mind that the past tense only relates to formulas (situations) that are true in the current state. If the desired behavior needs to depend on events that took place in the past, an auxiliary proposition should be created to 'remember' that the event took place. See Section 4 for some macros.

4 Macros

Macros are defined to provide a succinct way to specify some useful notions.

4.1 Memory Propositions

As mentioned in Section 3.2, if the specification depends on events that happened in the past, and not just in the previous state, an auxiliary proposition needs to be defined and used as memory.
To assist with that we define the macro:

- "*MemProp* is set on ϕ_{set} and reset on ϕ_{reset}"
- "*MemProp* is set on ϕ_{set} and reset on false"

where *MemProp* is the auxiliary proposition (which is a robot proposition) and ϕ_{set} and ϕ_{reset} are boolean formulas in the set of all propositions.

MemProp becomes true whenever ϕ_{set} is true *and* ϕ_{reset} is false. This means that if both formulas are true, *MemProp* will remain false. Once it is true, the proposition remains true while ϕ_{reset} is false and once it is false it remains false while ϕ_{set} is false. In other words, the proposition is true from the time (state) ϕ_{set} is true until, but not including, the time (state) in which ϕ_{reset} is true.

In the case of "*MemProp* is set on ϕ_{set} and reset on false", once *MemProp* becomes true it stays true forever.

As an example, if the desired behavior is to beep from the time the robot sensed Mika in Room 3 until it senses her mother, the specification would look like this:

Propositions: { `Mika, Mother, Room3, Beep, NeedToBeep` }

```
# Describing the behavior of the auxiliary proposition NeedToBeep:
NeedToBeep is set on Room3 and Mika and reset on Mother
```

```
# Describe the beeping:
Beep if and only if you are activating NeedToBeep
```

$\rightarrow$ LTL equivalent:

$$\begin{aligned} & \Box\,(\phi_{set} \wedge \neg\phi_{reset} \rightarrow \bigcirc\, MemProp) \\ \wedge & \Box\,(\phi_{reset} \rightarrow \neg \bigcirc\, MemProp) \\ \wedge & \Box\,(MemProp \wedge \neg\phi_{reset} \rightarrow \bigcirc\, MemProp) \\ \wedge & \Box\,(\neg MemProp \wedge \neg\phi_{set} \rightarrow \neg \bigcirc\, MemProp) \end{aligned}$$

4.2 Toggling Propositions

Similar to the memory propositions in Section 4.1, a macro is available to toggle the value of a system proposition when a specified event $\phi_{trigger}$ occurs.
This macro takes the following form:

- "*ToggleProp* is toggled on $\phi_{trigger}$"

where *ToggleProp* is the auxiliary proposition (which is a robot proposition) and $\phi_{trigger}$ is a boolean formula in the set of all propositions.

If *ToggleProp* is currently false and $\phi_{trigger}$ occurs, *ToggleProp* will be set to true in the next step. Likewise, if *ToggleProp* is currently true and $\phi_{trigger}$ occurs, *ToggleProp* will be set to false in the next step. If $\phi_{trigger}$ does not occur, the value of *ToggleProp* will not change.

→ LTL equivalent:

$$\begin{aligned}
&\square\,(TogglePropwedge\phi_{trigger} \rightarrow \neg \bigcirc ToggleProp) \\
\wedge\, &\square\,(\neg TogglePropwedge\phi_{trigger} \rightarrow \bigcirc ToggleProp) \\
\wedge\, &\square\,(TogglePropwedge \neg\phi_{trigger} \rightarrow \bigcirc ToggleProp) \\
\wedge\, &\square\,(\neg TogglePropwedge \neg\phi_{trigger} \rightarrow \neg \bigcirc ToggleProp)
\end{aligned}$$

4.3 Proposition Change Conditions

In some cases, you may wish to have something occur only when a proposition changes value. To make this easier, macros are defined to specify change conditions.
This macro takes the following forms:

- **Change from False to True:** "start of *Prop*", "beginning of *Prop*"
- **Change from True to False:** "end of *Prop*"

Please note the following restrictions:

- Change conditions cannot be used in liveness specifications
- Only a stand-alone proposition can be used inside a change macro
- One cannot refer to the change of a system proposition in an environment safety specification

4.4 Region Quantifiers

Groups of regions can be defined as follows:

- "Group *GroupName* is *Region1*, *Region2*, *Region3*"

One can then use the quantifiers "*any*" and "*all*" with respect to these groups (i.e. "any *GroupName*" or "all *GroupName*" where one would normally use a single region name). The way in which the quantifiers are evaluated depends on their context:

	any	***all***
Condition	$\bigvee_{\phi_i \in R} \phi_i$	$\bigwedge_{\phi_i \in R} \phi_i$ (*not useful*)
Safety Requirement ϕ_Q in $\square(\phi_C \rightarrow \phi_Q)$	(*not implemented*)	$\bigwedge_{\phi_i \in R} \square(\phi_C \rightarrow [\phi_Q]^{\phi_i})$
Liveness Requirement ϕ_Q in $\square\diamond(\phi_C \rightarrow \phi_Q)$	$\bigvee_{\phi_i \in R} \phi_i$	$\bigwedge_{\phi_i \in R} \square\diamond(\phi_C \rightarrow [\phi_Q]^{\phi_i})$

where R is the set of regions in the group, ϕ_Q is a requirement containing exactly one quantifier and no implications, and $[\phi_Q]^{\phi_i}$ denotes ϕ_Q with ϕ_i substituted in the place of the quantifier statement.
Note: Currently, only one quantifier can be used per line.

4.5 Non-projective Locative Propositions

The grammar also provides simple support for the use of the non-projective locative propositions “between”, “within”, and “near”.
Anywhere where you would normally specify a region, you can instead specify one of the following:

- “between *Region1* and *Region2*”
- “within *Distance* of/from *Region*”
- “near *Region*”

where *Distance* is specified in the coordinate system used inside region files, and “near *Region*” is defined to be equivalent to “within 50 of *Region*.”
The map will be automatically decomposed to create regions corresponding to your prepositional statements.