Boolean Logic

Truth Values

Robots don't like ambiguity when making decisions. They need to know, very clearly, which choice to make under what circumstances. As a consequence, their decisions are always based on the answers to questions which have only two possible answers: yes or no, true or false. Statements that can be only true or false are called **Boolean statements**, and their true-or-false value is called a **truth value**.

Fortunately, many kinds of questions can be phrased so that their answers are Boolean (true/false). Technically, they must be phrased as **statements**, not questions. So, rather than asking whether the sky is blue and getting an answer yes or no, you would state that "the sky is blue" and then find out the truth value of that statement, **true** (it is blue) or **false** (it is not blue).

Note that the truth value of a statement is only applicable **at the time it is checked**. The sky could be blue one minute and grey the next. But regardless of which it is, the statement "the sky is blue" is either true or false at **any specific time**. The truth value of a statement does not depend on **when** it is true or false, only **whether** it is true or false **right now**.

(Conditions)

ROBOTC **control structures** that make decisions about which pieces of code to run, such as **while loops** and **if-else** conditional statements, always depend on a (condition) to make their decisions. **ROBOTC (conditions) are always Boolean statements.** They are always either true or false at any given moment. Try asking yourself the same question the robot does – for example, whether the value of the Ultrasonic Sensor is greater than 45 or not. Pick any number you want for the Ultrasonic Sensor value. The statement "the Ultrasonic Sensor's value is greater than 45" will still either be true, or be false.

Condition	Ask yourself...	Truth value
`SensorValue(sonarSensor) > 45`	Is the value of the Ultrasonic Sensor greater than 45?	**True**, if the current value is more than 45 (for example, if it is 50). **False**, if the current value is not more than 45 (for example, if it is 40).

Some (conditions) have the additional benefit of ALWAYS being true, or ALWAYS being false. These are used to implement some special things like "infinite" loops that will never end (because the condition to make them end can never be reached!).

Condition	Ask yourself...	Truth value
`1==1`	Is 1 equal to 1?	**True**, always
`0==1`	Is 0 equal to 1?	**False**, always

Boolean Logic

Comparison Operators

Comparisons (such as the comparison of the Ultrasonic sensor's value against the number 45) are at the core of the decision-making process. A well-formed comparison typically uses one of a very specific set of operators, the "comparison operations" which generate a true or false result. Here are some of the most common ones recognized by ROBOTC.

ROBOTC Symbol	Meaning	Sample comparison	Result
==	"is equal to"	`50 == 50`	**true**
		`50 == 100`	**false**
		`100 == 50`	**false**
!=	"is not equal to"	`50 != 50`	**false**
		`50 != 100`	**true**
		`100 != 50`	**true**
`<`	"is less than"	`50 < 50`	**false**
		`50 < 100`	**true**
		`100 < 50`	**false**
`<=`	"is less than or equal to"	`50 <= 50`	**true**
		`50 <= 100`	**true**
		`50 <= 0`	**false**
`>`	"is greater than"	`50 > 50`	**false**
		`50 > 100`	**false**
		`100 > 50`	**true**
`>=`	Greater than or equal to	`50 >= 50`	**true**
		`50 >= 100`	**false**
		`100 >= 50`	**true**

Evaluating Values

The "result" of a comparison is either true or false, but the robot takes it one step further. The program will actually substitute the true or false value in, where the comparison used to be. Once a comparison is made, it not only is true or false, it literally **becomes** true or false in the program.

```
if (50 > 45) ...

if (true) ...
```

Boolean Logic

Use in Control Structures

"Under the hood" of all the major decision-making control structures is a simple check for the Boolean value of the (condition). The line `if (SensorValue(bumper) == 1)...` may read easily as "if the bumper switch is pressed, do...", but the robot is really looking for `if(true)` or `if(false)`. Whether the robot runs the "if true" part of the if-else structure or the "else" part, depends solely on whether the (condition) boils down to `true` or `false`.

```
if (50 > 45) ...

if (true) ...
```

Logical Operators

Some (conditions) need to take **more than one thing** into account. Maybe you only want the robot to run if the traffic light is green AND there's no truck stopped in front of it waiting to turn. Unlike the comparison operators, which produce a truth value by comparing other types of values (is one number equal to another?), the **logical operators** are used to **combine multiple truth values into one single truth value**. The combined result can then be used as the (condition).

Example:
Suppose the value of a Light Sensor named **sonarSensor is 50**, and at the same time, the value of a Bumper Switch named **bumper is 1** (pressed).

The Boolean statement `(sonarSensor > 45) && (bumper == 1)` would be evaluated...v

```
(50 > 45) && (1 == 1)

true && true

true
```

ROBOTC Symbol	Meaning	Sample comparison	Result
&&	"AND"	`true && true`	**true**
		`true && false`	**false**
		`false && true`	**false**
		`false && false`	**false**
||	"OR"	`true || true`	**true**
		`true || false`	**true**
		`false || true`	**true**
		`false || false`	**false**