computing: nature, power and limits—robotics applications (cis1.0)
fall 2006—lecture # D.1
monday 16-oct-2006

today:

- event-driven programming
- conditional execution
- robots and agents

event-driven programming

- *event*
  - something that happens while a program is running and provides input to the computer running the program
  - *for example:*
    - user input on a web page (like clicking on a button or in an image map)
    - sensor input to a robot (like bumping into something with a touch sensor)
- *event handler*
  - the part of a computer program that tells the computer what to do when an event happens
  - *for example:*
    - making a window pop up on a web page when a user clicks on a button or in an image map
    - making a robot stop when its touch sensor receives input that it has bumped into something

conditional execution

- *unconditional execution*—
  the computer executes (i.e., "runs") the program, or components of a program, no matter what the user does, or no matter what happens while the program (or component) is running
- *conditional execution*—
  the execution of a program, or component of a program (i.e., the way a program, or component of a program, runs), depends on what happens while the program (or component) is running; the program relies on *feedback* from its *environment*
  the *environment* can be a human user for an interactive web program, or a human interacting with a robot, or a robot's environment (i.e., the room in which it operates) interacting with it
- the classical programming syntax for conditional execution is *if-then-else*;
  in other words, *if* something happens, *then* the program does one thing; *else* (i.e., otherwise) the program does another thing

boolean tests and relational operators

- *Boolean test*
  - binary values can be thought of as: $0 = false$ and $1 = true$
  - these $true$ and $false$ values are called *logical* or *Boolean* values
  - anything that can be evaluated as having a value of $true$ or $false$ is considered a *Boolean test*
- *relational operator*
  - in a computer program, it is common to *compare* two values
  - these comparisons are done using *relational operators*, or "comparison" operators:

$==$	equal to
$!=$	not equal to
$<$	less than
$<=$	less than or equal to
$>$	greater than
$>=$	greater than or equal to

- mathematical statements that use these relational operators are called *boolean expressions*—and will always have a value of either $true$ or $false$

boolean algebra

- *Boolean* algebra was invented by Englishman George Boole (1815-1864)
- the idea behind *Boolean algebra* is to define ways in which logical values can be combined
- there are three basic *Boolean operators*: AND, OR, NOT
- each logical operator is defined using a *truth table*
- AND and OR are called *binary* operators because they take two *arguments*, i.e., two values (i.e., arguments) are combined using each operator

AND	$true$	$false$
$true$	$true$	$false$
$false$	$false$	$false$

OR	$true$	$false$
$true$	$true$	$true$
$false$	$true$	$false$

- NOT is called a *unary* operator because it only takes one argument, i.e., one value is combined with the NOT operator

NOT	$true$	$false$
	$false$	$true$

uses for boolean algebra

- *searching on the web for multiple terms:*
  `search for: "APPLE" AND "ORANGE"`
  returns all documents that have BOTH the word `APPLE` *and* the word `ORANGE` in them
  versus:
  `search for: "APPLE" OR "ORANGE"`
  returns all documents that have EITHER the word `APPLE` *or* the word `ORANGE` in them

- *controlling a robot to respond to multiple events:*
  `stop when: "TOUCH SENSOR IS PRESSED" AND "LIGHT SENSOR SEES DARK"`
  makes the robot stop when BOTH its touch sensor is pressed *and* its light sensor sees something dark
  versus:
  `stop when: "TOUCH SENSOR IS PRESSED" OR "LIGHT SENSOR SEES DARK"`
  makes the robot stop when EITHER its touch sensor is pressed *or* its light sensor sees something dark

examples

evaluate the following Boolean expressions:

1. `true AND false`
2. `true OR false`
3. `true AND (NOT false)`
4. `(NOT true) OR (NOT false)`
5. `(5 == 3) AND (6 > 3)`
6. `(5 == 5) OR (NOT true)`
7. `(1 == 2) OR (1 > 2) OR (1 < 2)`
8. `(1 == 2) AND (1 > 2) AND (1 < 2)`
9. `(1 == 2) OR (1 > 2) AND (1 < 2)`
10. `(1 == 2) AND (1 > 2) OR (1 < 2)`

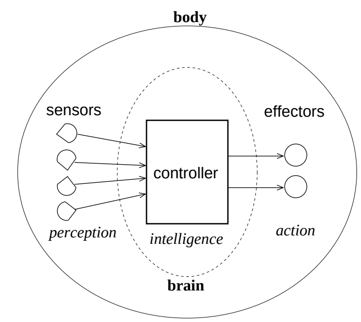
answers

1. `true AND false = false`
2. `true OR false = true`
3. `true AND (NOT false) = true AND true = true`
4. `(NOT true) OR (NOT false) = false OR true = true`
5. `(5 == 3) AND (6 > 3) = false AND true = false`
6. `(5 == 5) OR (NOT true) = true OR false = true`

```
7. (1 == 2) OR (1 > 2) OR (1 < 2) =
   false OR false OR true =
   false OR true = true

8. (1 == 2) AND (1 > 2) AND (1 < 2) =
   false AND false AND true =
   false AND true = false

9. (1 == 2) OR (1 > 2) AND (1 < 2) =
   false OR false AND true =
   false AND true = false

10.(1 == 2) AND (1 > 2) OR (1 < 2) =
   false AND false OR true =
   false OR true = true
```


robots and agents

- a *robot* is an *autonomous embodied agent*
- it is a mechanical device that exists in the physical world
- it has a *body* and a *brain* (i.e., a *COMPUTER* or *microprocessor*—a very small computer)
- it contains *sensors* to perceive its own state and to perceive its surrounding environment
- it possesses *effectors* which perform actions
- it has a *controller* which takes input from the sensors, makes *intelligent* decisions about actions to take, and effects those actions by sending commands to motors



the robots for our labs

- LEGO Mindstorms
- Hitachi h8300 microprocessor called **RCX**
- with an IR (infra-red) transceiver
- and 3 input ports, for:
  - touch sensor—to detect if the robot has bumped into anything
  - light sensor—to detect if the robot is "looking" at something dark or light (or somewhere in between)
- and 3 output ports, for:
  - motors
  - light bulbs

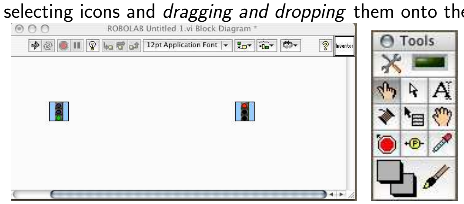
programming the LEGO Mindstorms

- you will write programs on a computer and *download* them to the RCX using an IR transmitter ("communication tower")

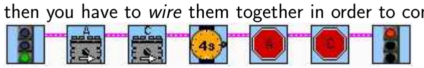
- we will use **RoboLab** — a graphical programming environment

how to program in RoboLab (1)

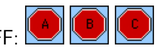
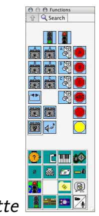
- when you start up RoboLab, you will see a *canvas* on the screen and you will use the menus (*icon palettes*) to select *icons* by clicking on them; you will create *programs* by selecting icons and *dragging and dropping* them onto the canvas



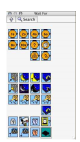
- then you have to *wire* them together in order to complete the program



how to program in RoboLab (2)

- all programs start with:  and end with: 
- *motor* icons turn motors ON: 
  remember to check which ports your motors are connected to on the RCX
- *lamp* icons turn lamps ON: 
- *stop sign* icons turn motors (and lamps) OFF: 
- all of these *icons* appear on the main *functions palette* 

how to program in RoboLab (3)

- these icons appear on the *wait for palette* 
  (click on  *on the functions palette)*
- *wait for* icons tell the program to *wait* (i.e., do nothing) until something (i.e., an *event*) happens
  timers wait for a specified amount of time to elapse: 
- *touch sensor* icons wait for the *state* of the touch sensor to change:
   = wait for the touch sensor to be pushed
   = wait for the touch sensor to be released *(you probably won't need to use this one!)*

case study: vacuum cleaner robots

- read the articles on the web page about *Roomba*
- think about what a robot vacuum cleaner does
- what *events* should it respond to?
- what *conditional* behaviors should it have?