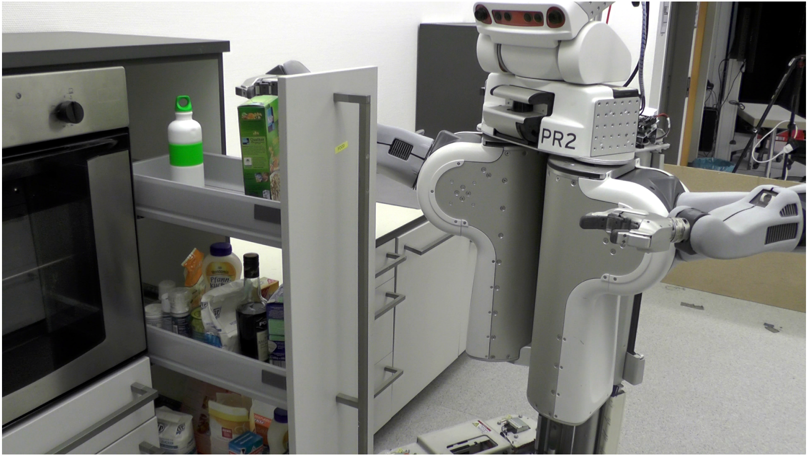
Fall School 2021 - Day 3

Mobile Manipulation with Cognitive Robot Abstract Machine (CRAM)

Alina Hawkin, Arthur Niedzwiecki
Institute for Artificial Intelligence
University Bremen

November 10, 2021

Demonstration



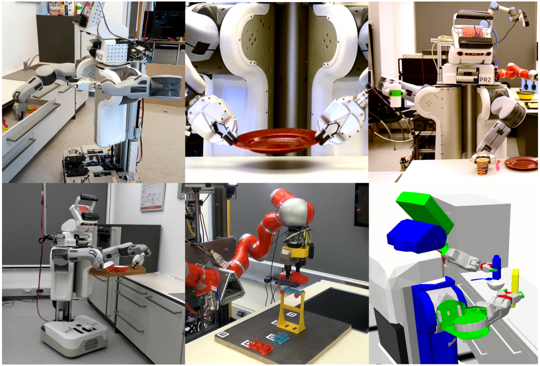
Agenda

1. Abstract Machine
2. CRAM Plan Executive
   - Motions
   - Action Hierarchy
   - Parameters
3. Tutorials
4. Failure Handling

Agenda

1. Abstract Machine

Motivation



One plan to accomplish all variations of fetch and place:

- different *objects*, *environments*, *robot platforms*, *applications*.

Abstract Machines in Computer Science

Adapted from Pedro Domingos: "What's Missing in AI: the Interface Layer"

Field	**Interface Layer**	**Below the Layer**	**Above the Layer**
Operating Systems	virtual machines	hardware	software
Programming systems	high-level languages	compilers, optimizers, ...	programming
Databases	relational model	query optimization, db design, transaction mgmt	enterprise applications

Abstract Machines in Computer Science

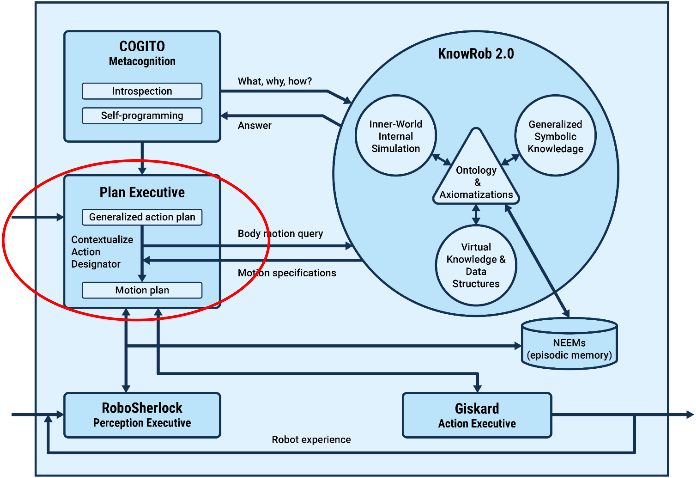
Adapted from Pedro Domingos: "What's Missing in AI: the Interface Layer"

Field	**Interface Layer**	**Below the Layer**	**Above the Layer**
Operating Systems	virtual machines	hardware	software
Programming systems	high-level languages	compilers, optimizers, ...	programming
Databases	relational model	query optimization, db design, transaction mgmt	enterprise applications
Personal robotics	**CRAM**	*grounding in robot, AI tools, the nuts and bolts of intelligent robotics, ...*	*robot application programming*

Raise the conceptual level at which service and personal robot applications are programmed!

Agenda

CRAM General Overview



The CRAM 2.0 system.

Challenges Tackled by the Plan Executive

1. Define which actions to execute to achieve the goal.
2. Infer which parameters to use for each action.
3. Monitor task execution and react to failures.

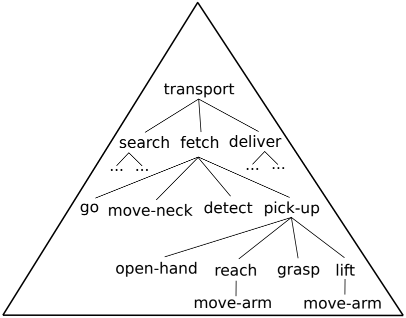
Primitives: Motions and Percepts

Primitives of Mobile Pick and Place for PR2-like Robots

Primitive	**Description**
going	drive or walk or fly to the goal pose
moving-torso	move torso to the goal joint position
moving-neck	move the neck to direct the gaze
moving-arm	execute a trajectory in Cartesian or joint space
grasping/releasing	move the fingers to grasp or release an object
opening-hand/cl.	move the fingers to open or close the hand
monitoring-joints	monitor the positions of robot body parts in space
detecting	perceive the described object in the environment
moving-eye	move the eye in the socket to direct the gaze
...	

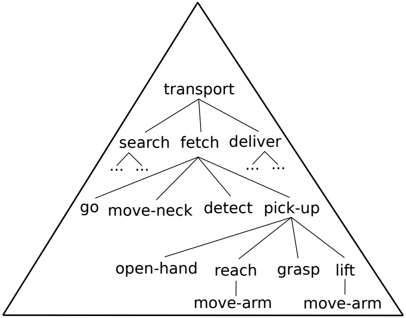
Action Model

Model of Mobile Pick & Place and a Simple Plan Written in CPL



Action Model

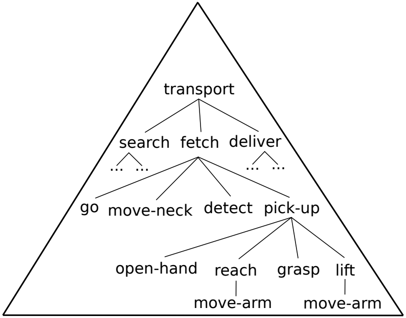
Model of Mobile Pick & Place and a Simple Plan Written in CPL




```
(a primitive
    (                    )
    (                    )
    ...)
```

Action Model

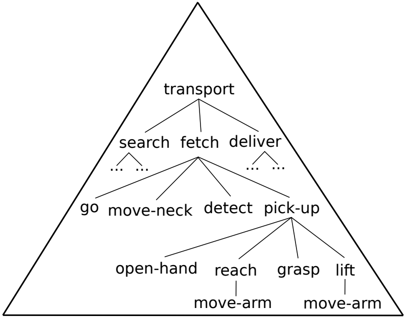
Model of Mobile Pick & Place and a Simple Plan Written in CPL




```
(a primitive
    (type                    )
    (                        )
    ...)
```

Action Model

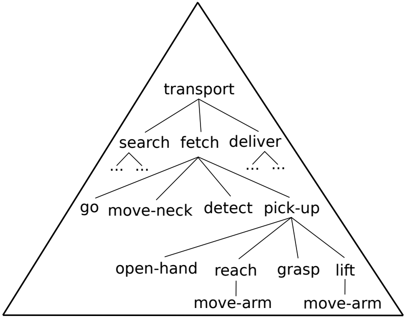
Model of Mobile Pick & Place and a Simple Plan Written in CPL




```
(a primitive
    (type moving-arm)
    (                )
    ...)
```

Action Model

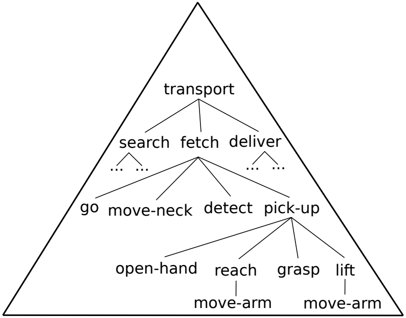
Model of Mobile Pick & Place and a Simple Plan Written in CPL




```
(a primitive
    (type moving-arm)
    (cart-goal ?goal)
    ...)
```

Action Model

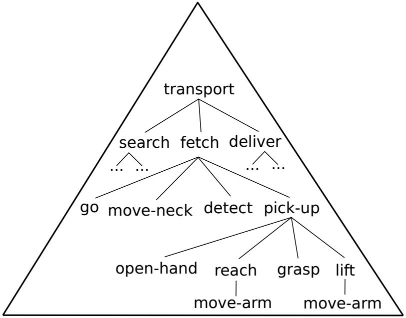
Model of Mobile Pick & Place and a Simple Plan Written in CPL




```
perform (a primitive
          (type moving-arm)
          (cart-goal ?goal)
          ...)
```

Action Model

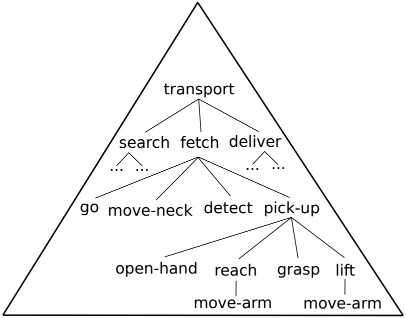
Model of Mobile Pick & Place and a Simple Plan Written in CPL




```
def_plan reach (          )
  perform (a primitive
               (type moving-arm)
               (cart-goal ?goal)
               ...)
```

Action Model

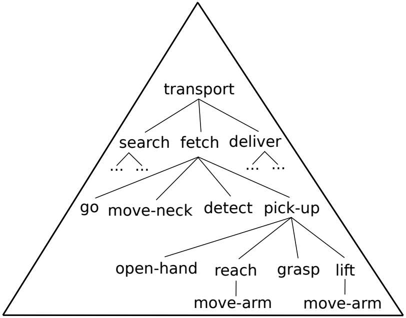
Model of Mobile Pick & Place and a Simple Plan Written in CPL




```
def_plan reach (goal ...)
  perform (a primitive
              (type moving-arm)
              (cart-goal ?goal)
              ...)
```

Action Model

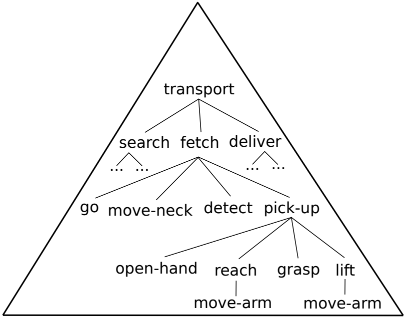
Model of Mobile Pick & Place and a Simple Plan Written in CPL




```
def_plan reach (goal ...)
  perform (a primitive
              (type moving-arm)
              (cart-goal ?goal)
              ...)

def_plan pick_up (...)
```

Action Model

Model of Mobile Pick & Place and a Simple Plan Written in CPL



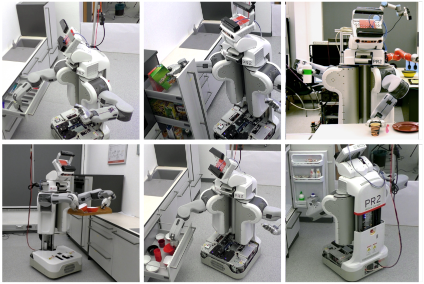

```
def_plan reach (goal ...)
  perform (a primitive
              (type moving-arm)
              (cart-goal ?goal)
              ...)

def_plan pick_up (...)
  perform (a primitive
              (type opening-hand))
  perform (an action
               (type reaching) ...)
  perform (a primitive
              (type grasping) ...)
  perform (an action
               (type lifting) ...))
```

Parameters of Motion and Perception Primitives

Primitive	Parameters
going	goal_pose, ..., speed, ...
moving-torso	goal_position, ...
moving-neck	goal_positions, goal_coordinate_to_look_at, ...
moving-arm	goal_pose_for_hand, goal_positions, collisions, ...
grasping/releasing	hand, grasping_force, object_properties, ...
opening-hand/cl.	hand, ...
monitoring-joints	joint_name, joint_value, monitoring_function, ...
detecting	object_description, ...

Calculating parameter values that maximize success probability: heuristics, learning from experience, imitation learning, ask a human

Choice of Parameter Values is Crucial For Success



- Often very many possible values to choose from

  Example: from which side and with which hand to grasp?

- Effects can be:
  - immediate
  - short-term
  - long-term

Agenda

1. Abstract Machine
2. CRAM Plan Executive
   - Motions
   - Action Hierarchy
   - Parameters
3. Tutorials
4. Failure Handling

Before the Tutorial: Setup and Changing the Workspace

Please download the CRAM-VM if you haven't done so yet, and follow the setup steps described in the tutorial:

`http://cram-system.org/tutorials/demo/fetch_and_place`

Open a terminal (Ctrl-Alt-T) and within the terminal do:

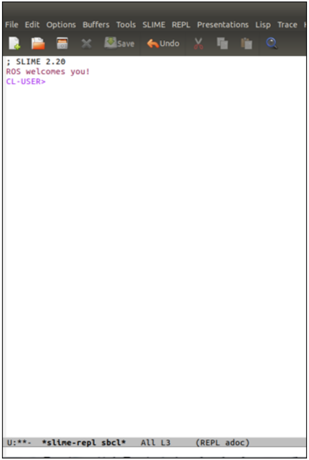
- *gedit .bashrc*
- Go to the bottom of the file
- Comment-out the line starting with *source* for day 1 & 2
- Comment-in the line for day 3
- Save and close the .bashrc file
- Do *source .bashrc*

Now if you type *roscd* you should be in the *cram_tutorial* workspace.

Exercise 1: Orc-Battle (Getting familiar with Emacs)

start emacs from the terminal with *roslisp_repl &*

- This is the REPL (Read-Eval-Print-Loop)
- *Cl-USER* = shows you the package you are currently located in. *CL-USER* is the default.
- you can type any lisp code here, and after you press *enter*, it will be evaluated. try: *(+ 2 3)*



LISP in a nutshell

- everything has to be in parenthesis (. . .)
- prefix notation takes a while to get used to:
  (+ 2 3)
  (defvar *var* 0.0)
- everything is a list:
  (list 1 2 3)
  '(1 2 3)
- the four disguises of an empty list:
  '() , () , 'nil , nil → all of these will evaluate to nil
- there is no *return* statement. The REPL will output the result of the function automatically by default.

LISP in a nutshell

- you can define a new function with *defun*:
  (defun test (parameters) . . . *fancy code* . . .)
- define a **global** variable with *defparameter* and put the name into asterisks (convention).
  (defparameter *my-var* 12)
  (defparameter *my-var-nil* nil)
  (defparameter *my-var-str* "string")
- change the variable value with *setf*
  (setf *my-var* 10)
- lisp variables are not typed (everything is a list anyway)

LISP in a nutshell

- local variables are defined with *let*:
  ```
  (let ((a 10)
  (b 5))
  (+ a b))
  ```
- difference to let*
  ```
  (let* ((a 10)
  (g 5)
  (d (+ 3 g)))
  (print d))
  ```

Exercise 1: Orc-Battle (Getting familiar with Emacs)

- open a file with *Ctrl-x Ctrl-f*
  - look for Downloads/orc-battle.lisp and hit Enter
- compile the whole file with *Ctrl-c Ctrl-k*
- switch buffer with *Ctrl-x b*
  - use up and down keys to find **slime-repl sbcl**, then press *Enter*
- execute *(orc-battle)*

Emacs Keybindings

The following notation is used in Emacs for keyboard shortcuts:

- C for <Ctrl>
- M for <Alt>
- SPC for <Space>
- RET for <Enter>
- '-' for when two keys are pressed together (e.g. C-x for <Ctrl>+x)

- Open a file: `C-x C-f`
  TAB auto-completes, RET opens
- Switch buffer: `C-x b`
  Up/Down keys: browse buffers
- Split view horizontally: `C-x 2`
- Split view vertically: `C-x 3`
- Switch between tabs: `C-x o`
- Close current tab: `C-x 0`
- Cut: `C-w`
- Copy: `M-w`
- Paste (yank): `C-y`
- Compile section: `C-c C-c`
- Compile whole file: `C-c C-k`

More Keybindings

- Cancel command mid-way: `C-g`
  Or hit ESC 3 times.
- Kill buffer: `C-x k`
- Jump to definition: `M-.`
- Jump back from definition: `M-,`
- Select code within parentheses:
  `C-M-SPC`
  When at an opening parenthesis
- Exit Emacs: `C-x C-c yes`

While in the REPL

- Delete current input:
  `C-M-Backspace`
- Get last command: `C-UP`
- Break line: `C-j`

Exercise 2: Fetch and Place Plans

Goal: To write a plan for fetching and delivering objects.
Follow the section 'Simple Fetch and Place' for the
`http://cram-system.org/tutorials/demo/fetch_and_place`

- Load the tutorial package:
  ```
  (ros-load:load-system "cram_pick_place_tutorial":cram-pick-place-tutorial)
  ```
- Change into the tutorial package:
  ```
  (in-package :cram-pick-place-tutorial)
  ```
- load your VM
- head to
  `http://cram-system.org/tutorials/demo/fetch_and_place`
- follow the instructions there

Exercise 3: Failure Handling

- Goal: Robot should be able to reliably perform pick and place tasks
- Many things could go wrong:
  - object could slip out of the gripper
  - sensor noise can cause faulty grasping poses
  - ...
- in short: the more complex the plan, the more could go wrong!
  $\rightarrow$ how can we avoid/fix this?

Perceiving Goal States and Detecting Failures

Ensuring that the goal was achieved can be done through:

- extrinsic perception of the scene
  - after placing the object, perceive the scene to ensure that it is actually there

Perceiving Goal States and Detecting Failures

Ensuring that the goal was achieved can be done through:

- extrinsic perception of the scene
  - after placing the object, perceive the scene to ensure that it is actually there
- intrinsic perception of the robot's body part positions with respect to each other
  - ensure that the arm / base / neck reached the goal
  - ensure that the gripper did not close completely if an object was expected to be grasped

Perceiving Goal States and Detecting Failures

Ensuring that the goal was achieved can be done through:

- extrinsic perception of the scene
  - after placing the object, perceive the scene to ensure that it is actually there
- intrinsic perception of the robot's body part positions with respect to each other
  - ensure that the arm / base / neck reached the goal
  - ensure that the gripper did not close completely if an object was expected to be grasped
- other kinds of perception
  - estimate the weight of the object in the hand
  - use tactile perception
  - react to sounds, smells, etc.

Failure Types

- Low-level (primitive) vs high-level (action) failures
  - low-level failures are thrown if a primitive was not successful, e.g., *going_low_level_failure*, *arm_low_level_failure*, *hand_low_level_failure*, *neck_low_level_failure*, *torso_low_level_failure*, *perception_low_level_failure*, ...
  - high-level failures are thrown if an action did not succeed, e.g., *picking_up_failure*, *searching_failure*

Failure Types

- Low-level (primitive) vs high-level (action) failures
  - low-level failures are thrown if a primitive was not successful, e.g., *going_low_level_failure*, *arm_low_level_failure*, *hand_low_level_failure*, *neck_low_level_failure*, *torso_low_level_failure*, *perception_low_level_failure*, ...
  - high-level failures are thrown if an action did not succeed, e.g., *picking_up_failure*, *searching_failure*
- Planning time vs execution time vs post-execution failures
  - planning time failures are thrown if the robot anticipates that the action will fail **before** executing it, e.g., by using simulation
  - execution time failures are signaled if a deviation from the intended course of action is detecting **during** action execution
  - post-execution failures are those that are detected **after** action execution has finished

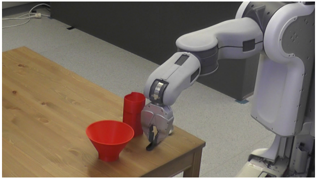
Strategy 1: Retrying Without Change

The sequence of actions that aims to negate the unwanted effects of the failure, together with the new sequence of actions that leads to success, is the **failure handling strategy**.

Strategy 1: Retrying Without Change

The sequence of actions that aims to negate the unwanted effects of the failure, together with the new sequence of actions that leads to success, is the **failure handling strategy**.

- The real world is non-deterministic:
  executing the same action the same way can have different effects.
- Simplest strategy: simply retry executing the action the same way.
- Example: if grasping failed, simply try to perceive and grasp again.



Retrying Without Change Strategy in CPL

```
perform (an action
          (type detecting)
          (object (an object
                   (type spoon)))
          ...)
perform (an action
          (type picking-up)
          (object (the object
                    (type spoon))))
          (hand right-hand)
          (grasp top-grasp)
          ...)
```

Retrying Without Change Strategy in CPL

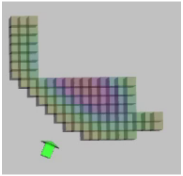
```
with_failure_handling

    perform (an action
              (type detecting)
              (object (an object
                        (type spoon)))
              ...)
    perform (an action
              (type picking-up)
              (object (the object
                         (type spoon))))
              (hand right-hand)
              (grasp top-grasp)
              ...)

  catch grasping_failure

      retry
```

Retrying Without Change Strategy in CPL

```
with_retry_counters grasp_counter = 3
  with_failure_handling

      perform (an action
                (type detecting)
                (object (an object
                          (type spoon)))
                ...)
      perform (an action
                (type picking-up)
                (object (the object
                          (type spoon))))
                (hand right-hand)
                (grasp top-grasp)
                ...)

    catch grasping_failure
      do_retry grasp_counter
        retry
```

Strategy 2: Retrying with Changing the Parameters

- Strategy: pick another parameter value and retry
- Parameter values can be represented using a probability distribution
- Choosing the next parameter: random sampling, gradient descent, A*, any other type of search
- If the action has multiple parameters, do a search over all the parameters, the search tree grows exponentially



Retrying with Changing the Parameters in CPL

```
robot_base_location = (a location
                          (to open)
                          (object (an object
                                      (type refrigerator)))))
```

Retrying with Changing the Parameters in CPL

```
robot_base_location = (a location
                         (to open)
                         (object (an object
                                    (type refrigerator)))))



      perform (an action
                 (type going)
                 (target ?robot_base_location))))
      perform (an action
                 (type opening)
                 (object (an object
                            (type refrigerator)))
                 (hand right-hand)
                 ...))
```

Retrying with Changing the Parameters in CPL

```
robot_base_location = (a location
                         (to open)
                         (object (an object
                                    (type refrigerator)))))

  with_failure_handling

      perform (an action
                  (type going)
                  (target ?robot_base_location))))
      perform (an action
                  (type opening)
                  (object (an object
                              (type refrigerator)))
                  (hand right-hand)
                  ...))

    catch grasping_failure
```

Retrying with Changing the Parameters in CPL

```
robot_base_location = (a location
                        (to open)
                        (object (an object
                                  (type refrigerator)))))

  with_failure_handling

      perform (an action
                (type going)
                (target ?robot_base_location))))
      perform (an action
                (type opening)
                (object (an object
                          (type refrigerator)))
                (hand right-hand)
                ...))

    catch grasping_failure

        robot_base_location = next(robot_base_location)

          retry
```

Retrying with Changing the Parameters in CPL

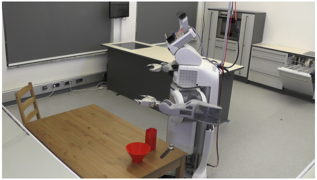
```
robot_base_location = (a location
                         (to open)
                         (object (an object
                                    (type refrigerator)))))

  with_failure_handling

     perform (an action
                (type going)
                (target ?robot_base_location))))
     perform (an action
                (type opening)
                (object (an object
                           (type refrigerator)))
                (hand right-hand)
                ...))

    catch grasping_failure
      if exists next(robot_base_location)
        robot_base_location = next(robot_base_location)

          retry
```

Retrying with Changing the Parameters in CPL

```
robot_base_location = (a location
                        (to open)
                        (object (an object
                                   (type refrigerator)))))
with_retry_counters grasp_counter = 3
  with_failure_handling

      perform (an action
                   (type going)
                   (target ?robot_base_location))))
      perform (an action
                   (type opening)
                   (object (an object
                                (type refrigerator)))
                   (hand right-hand)
                   ...))

    catch grasping_failure
      if exists next(robot_base_location)
        robot_base_location = next(robot_base_location)
        do_retry grasp_counter
          retry
```

Strategy 3: Retrying with Changing the Actions

- Sometimes retrying the action is not sufficient:
  one needs additional actions
  - If stuck, wiggle yourself to get unstuck.
  - If object cannot be found, move the torso to a different configuration.
  - If everything fails, ask a human for help.
- One can put these if-else cases explicitly as a part of the plan and not the failure handling.
  - For efficiency, skip actions, where the goal has already been achieved.



Strategy 3 in CPL

```
with_retry_counters perception_counter = 3
  with_failure_handling

      perform (an action
                 (type detecting)
                 (object (an object
                            (type spoon)))
                 ...))

    catch perception_failure
      perform (a primitive
                (type moving-torso)
                (joint-position highest)))
      do_retry perception_counter
        retry
```

Exercise 3: Failure Handling

Brief introduction to Failure Handling concepts in CRAM + Tutorial:
`http://cram-system.org/tutorials/demo/fetch_and_place_3`

Extra Tutorial: Advanced Failure Handling

An extra Tutorial with a few more exercises:
`http://cram-system.org/tutorials/demo/fetch_and_place_4`

If we could spark your interest, here are some more Tutorials:
`http://cram-system.org/tutorials`

Have fun!