Introduction to Cognitive Robotics

Module 9: The CRAM Plan Language
Lecture 2: Designators, process modules

www.cognitiverobotics.net

The CRAM Language

Based on CRAM documentation
http://cram-system.org/doc

CRAM Designators

- A designator is a Common Lisp object for describing various parameters in the CRAM language
- To the user, designators are objects containing sequences of key-value pairs of symbols

CRAM Designators

- Designators are effectively placeholders
  - require runtime resolution
  - based on the current context of the task action
- Designator resolution is accomplished by
  - Querying a priori knowledge embedded in the plan
  - Querying knowledge in the KnowRob2 knowledge base
  - By accessing sensorimotor data through the Perception Executive

CRAM Designators

Four types of designator

1. Motion designators (e.g. motor command)
2. Location designators (e.g. 3D pose)
3. Object designators (e.g. grasp configuration)
4. Action designators (e.g. goal)

CRAM Designators

- Designators are atomic
  - Properties are not changed by the user
  - Value is persistent
- Designators can be equated
  - for example, if they refer to the same object (at different times)
- Designators are multivalued
  - More than one key-value pair might be a solution

CRAM Designators

User API

- Functions for constructing, equating, and accessing designators
- Other API for resolving designators

CRAM Designators

```
(make-designator parent properties &optional parent)
```

- Constructs a designator of type `class` and the given properties
- If `parent` is specified, it is equated with it

```
((equate parent successor)
```

- Equates the two designators

CRAM Designators

`(desig-equal designator-1 designator-2)`

- Checks if two designators describe the same entity, i.e. if they are equated

`(first-desig designator)`

- Returns the first ancestor in the chain of equated designators

`(current-desig designator)`

- Returns the newest designator, i.e. that one that has been equated last to designator or one of its equated designators.

CRAM Designators

```
(reference designator &optional role)
```

- Tries to dereference the designator
  - Return its data object
  - Throws an error if it is not an effective designator, i.e. one that can be dereferenced
- By specifying a `role` parameter, different algorithms to resolve (i.e. dereference) the designator can be selected

CRAM Designators

```
(next-solution designator)
```

- Returns another solution for the effective designator `designator` or `NIL` if none exists

- The next solution is a newly constructed designator with identical properties that is equated to designator since it describes the same entity

CRAM Designators

```
(designator-solutions-equal solution-1 solution-2)
```

- Compares two designator solutions and returns `T` if they are equal

```
(designator-solutions designator &optional from-root)
```

- Returns the lazy list of all solutions of a designator

CRAM Designators

```
(copy-designator old-designator &key new-designator)
```

- Returns a new designator with the same properties as `old-designator`

- When present, the description parameter `new-designator` will be merged with the old description

- The new description will be dominant in this relation

CRAM Designators

```
(make-effective-designator parent &key new-properties data-object time-stamp)
```

- Returns a new effective designator with the same type as parent
- The parent's properties are used if `new-properties` is not specified
- The internal data slot that is used for dereferencing the designator is set to `data-object`

CRAM Designators

```
(newest-effective-designator designator)
```

- Returns the newest, i.e. the current, equated effective designator

```
(desig-prop-value designator property-key)
```

- Returns the value part of the designator property indicated by `property-key`

CRAM Designators

Designator resolution

- Resolution means to generate real parameters for executing actions from the symbolic key-value pairs of a designator

- Each designator class – `action`, `object`, `location`, `and` `motion` – are resolved differently

CRAM Designators

Object Designators

- Direct interface between CRAM plans and the robot perception sub-system
- The key-value pairs in the designator's properties describe the object that is to be perceived
- The perception sub-system is then responsible for resolving the designator, as follows

CRAM Designators

Object Designators

1. Parse the designator's properties for a description of the object to be found

2. Find the described object

3. Create a new effective designator with the data object bound to an instance of `object-designator-data` containing all relevant information about the object

4. Equate the new effective designator with the original designator

CRAM Designators

Action Designators

- Resolution is normally effected by using an inference engine – Prolog – to convert symbolic action descriptions to ROS action goals or similar data structures

- To implement resolution for an action designator, the user has to provide definitions for the predicate `action-desig ?designator ?solution`

CRAM Designators

Action Designators

For example, the following designator describes the action of moving 1 m forward

The action designator has a type, in this case `(type navigation)`

```
(let ((goal-location
        (make-designator 'location
          `((pose ,(make-pose-stamped
                    "base_footprint" 0.0
                    (make-3d-vector 1.0 0.0 0.0)
                    (make-identity-rotation)))))))
  (make-designator 'action `((type navigation) (goal ,goal-location)))
```

action type

goal pose: resolution requires this pose to be specified

CRAM Designators

Action Designators

To resolve the pose, we define the corresponding predicate `action-desig`

```
(def-fact-group navigation-action-designator (action-desig)
  (<- (action-desig ?designator ?goal)
    (desig-prop ?designator (type navigation))
    (desig-prop ?designator (goal ?goal))))
```

CRAM Designators

Location Designators

- These designators are resolved as a robot pose that is appropriate for manipulating an object
- They can be the most complex to resolve because the computation of poses such as "a location in which to stand for opening a drawer" is non-trivial

CRAM Designators

Location Designators

- Resolution is done in two steps:

  1. Generation of a lazy list of candidate poses

  2. Verification of candidate poses
     (i.e. is the pose actually a feasible solution)

- This allows for a general generation process and specific filter process to remove the invalid solutions

CRAM Designators

Location Designators

- When the `reference` method is called on a location designator:
  - The system first executes generator functions to generate the sequence of candidate poses
  - Generator functions are prioritized: there is an explicit order in which they must be executed
- Each generator function takes a location designator as input and returns the (lazy) list of possible solutions

CRAM Designators

Location Designators

- Next, the system …
    - Appends (concatenates) all lists
    - Validates each solution in turn
    - Until
        - a valid solution is found or
        - a maximum number of solutions has been tried
          (in which case the system throws an error)

CRAM Designators

Location Designators

- Validation functions have two parameters
  - The designator to be resolved
  - The pose candidate to validate
- Depending on the result, the solution can be
  - accepted
  - immediately rejected
  - rejected if no other validation function accepts it

CRAM Designators

Location Designators

To register a location generator function

```
register-location-generator priority function
&optional doc-string
```

`priority` is a `fixnum` used to order all location generators. Common-Lisp distinguishes two types of integer types: `fixnums` and `bignum`. A fixnum is a small integer, which ideally occupies only a word of memory.

Solutions generated with functions with smaller priorities are used first.

`function` is a symbol naming the function that generates a list of solutions.

It takes exactly one argument: the designator

CRAM Designators

Location Designators

To register a location validation function

```
register-location-validation priority function
&optional doc-string
```

`priority` is a `fixnum` used to indicate the evaluation order of all location validation functions

`function` is a symbol naming the validation function.

This function takes exactly two arguments: the designator and a solution.

It returns either
`:accept`,
`:unknown`
(cannot decide),
`:maybe-reject`
(reject if all other validation functions return `:unknown`),
or
`:reject`

CRAM Process Modules

- Process modules provide a high-level abstract interface to navigation, manipulation, and perception functionality
- They hide the details of robot-specific components
- Their input is always an action designator
- They are defined with the `def-process-module` macro

  Similar to to a function definition but only one parameter is declared: the name of the variable that the input designator is bound to

CRAM Process Modules

Process modules operate as follows:

1. The action designator is resolved to get the actual commands to send to the robot hardware
2. These are executed
3. Return a result value or throw an error if the execution fails

CRAM Process Modules

Example:

Define the `navigation` process module,

to be bound to the action designator `input-designator`

```
(def-process-module navigation (input-designator)
  (let ((goal-pose (reference input-designator)))
    (or (execute-navigation-action goal-pose)
        (fail 'navigation-failed))))
```

which (we will assume) resolves to a 3D pose …

… that is passed to the `execute-navigation-action` function

… and throw an error if that navigation action fails

CRAM Process Modules

To send a command to the process module

use `pm-execute`

start up the specified process module

```
(top-level
  (with-process-modules-running (navigation)
    (pm-execute navigation (make-designator 'action
                '((type navigation) (goal, some-pose))))))
```

Here, we assume that `some-pose` has a valid pose value

CRAM Language Resources

CRAM Designators	http://cram-system.org/doc/package/cram_designators
CRAM Process Modules	http://cram-system.org/doc/package/cram_process_modules

Background Reading

G. Kazhoyan, Lecture notes: Robot Programming with Lisp 7. Coordinate Transformations, TF, ActionLib, slides 5-8.
`https://ai.uni-bremen.de/_media/teaching/7_more_ros.pdf`

T. Rittweiler, CRAM – Design and Implementation of a Reactive Plan Language, Bachelor Thesis, Technical University of Munich, 2010.
`https://common-lisp.net/~trittweiler/bachelor-thesis.pdf`