Affordance-based reasoning for robot task planning

Iman Awaad
Gerhard K. Kraetzschmar

Bonn-Rhein-Sieg University
and B-IT Center
Grantham-Allee 20
53757 Sankt Augustin, Germany

Joachim Hertzberg

Osnabrück University
and DFKI RIC Osnabrück Branch
Albrechtstrasse 28
49076 Osnabrück, Germany
Enable robots to handle unexpected situations

RESEARCH GOAL
Enable robots to handle unexpected situations

Object substitution

Photo credit: http://blog.comfree.com/2013/05/03/clever-kitchen-storage-solutions/#.Uaekj-uDGJN
Enable robots to handle unexpected situations
Object substitution as tool usage
Enable robots to handle unexpected situations
Performance enhancement

Enable robots to handle unexpected situations

Action substitution
Affordances are opportunities for action provided by a particular object or environment. 

A closed door does not afford passage!

James J. Gibson
Perceived affordances allude to how an object may be interacted with based on the actor’s goals, plans, values, beliefs and past experiences.

It might afford opening and passage!

Don Norman
Perceived affordances allude to how an object may be interacted with based on the actor’s goals, plans, values, beliefs and past experiences.

It might afford opening and passage!

Enable intelligent behavior

Provide a means to represent & use a priori knowledge

Handle underspecified commands

Functional affordances

Reduce the action space

Don Norman

Tuesday, 11 June 13
Enable robots to handle unexpected situations

Show how to model Affordances and use them before Planning, during planning & at execution

Description Logics
real domains, especially in service robotics, are really **hard** to model

1. Model the domain
2. Create the planning problem
3. Generate a plan
4. Execute/Monitor it
Model as much as possible
— difficult, time consuming

Model as little as possible
— could lose solutions

Use domain information to quickly solve hard problems
Convenient representation

How do we know this is possible?

- Heterogenous hardware
- Faulty hardware
The recent semantic web language is the tension of enumeration; hence it can be categorized as Table 2.1: RDF, DAML, and others. Since union \( S \) refers to role definitions, Qualified number restriction \( R.C \) refers to atomic concepts, and \( C \) refers to atomic roles and \( S \) refers to role definitions.

<table>
<thead>
<tr>
<th>Construct</th>
<th>Syntax</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concept</td>
<td>( A )</td>
<td>( \mathcal{F}_{\mathcal{L}_0} )</td>
</tr>
<tr>
<td>Role name</td>
<td>( R )</td>
<td>( \mathcal{F}_{\mathcal{L}} )</td>
</tr>
<tr>
<td>Intersection</td>
<td>( C \cap D )</td>
<td>( \mathcal{A}\mathcal{C} )</td>
</tr>
<tr>
<td>Value restriction</td>
<td>( \forall R.C )</td>
<td>( S )</td>
</tr>
<tr>
<td>Limited existential quantification</td>
<td>( \exists R )</td>
<td>( S )</td>
</tr>
<tr>
<td>Top or Universal</td>
<td>( T )</td>
<td>( S )</td>
</tr>
<tr>
<td>Bottom</td>
<td>( \bot )</td>
<td>( S )</td>
</tr>
<tr>
<td>Atomic Negation</td>
<td>( \neg A )</td>
<td>( S )</td>
</tr>
<tr>
<td>Negation(^{1})</td>
<td>( \neg C )</td>
<td>( S )</td>
</tr>
<tr>
<td>Union</td>
<td>( C \sqcup D )</td>
<td>( S )</td>
</tr>
<tr>
<td>Existential restriction</td>
<td>( \exists R.C )</td>
<td>( S )</td>
</tr>
<tr>
<td>Number restrictions</td>
<td>( (\leq n , R) (\geq n , R) )</td>
<td>( S )</td>
</tr>
<tr>
<td>Nominals</td>
<td>{a_1 \ldots a_n}</td>
<td>( S )</td>
</tr>
<tr>
<td>Role hierarchy</td>
<td>( R \subseteq S )</td>
<td>( S )</td>
</tr>
<tr>
<td>Inverse Role</td>
<td>( R^{-1} )</td>
<td>( S )</td>
</tr>
<tr>
<td>Qualified number restriction</td>
<td>( (\leq n , R.C) (\geq n , R.C) )</td>
<td>( S )</td>
</tr>
</tbody>
</table>

\(^{1}\) \( A \) refers to atomic concepts, \( C \) and \( D \) refers to any concept definition, \( R \) refers to atomic roles and \( S \) refers to role definitions.

\(^{2}\) \( \mathcal{F}_{\mathcal{L}} \) is used for structural DL languages and \( \mathcal{A}\mathcal{C} \) for attributive languages \([BCM+03]\). \( S \) is the name used for the language \( \mathcal{A}\mathcal{C} R^+ \), which is composed of \( \mathcal{A}\mathcal{C} \) plus transitive roles.

\(^{3}\) \( \mathcal{A}\mathcal{C} \) and \( \mathcal{A}\mathcal{C}\mathcal{U}\mathcal{E} \) are equivalent languages, since union \( (U) \) and existential restriction \( (E) \) can be represented using negation \((C)\).

### Modeling the Domain


Knowledge Base

<table>
<thead>
<tr>
<th>Constructor</th>
<th>DL syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>intersectionOf</td>
<td>( C_1 \ldots \cap C_n )</td>
<td>Human \cap Male</td>
</tr>
<tr>
<td>unionOf</td>
<td>( C_1 \ldots \cup C_n )</td>
<td>Doctor \cup Lawyer</td>
</tr>
<tr>
<td>complementOf</td>
<td>( \neg C )</td>
<td>( \neg ) Male</td>
</tr>
<tr>
<td>oneOf</td>
<td>{x_1 \ldots x_n}</td>
<td>{john,mary}</td>
</tr>
<tr>
<td>allValuesFrom</td>
<td>( \forall P.C )</td>
<td>\forall hasChild.Doctor</td>
</tr>
<tr>
<td>someValuesFrom</td>
<td>( \exists R.C )</td>
<td>\exists hasChild.Lawyer</td>
</tr>
<tr>
<td>hasValue</td>
<td>( \exists R{x} )</td>
<td>\citizenOf.{USA}</td>
</tr>
<tr>
<td>minCardinality</td>
<td>( (\geq n , R) )</td>
<td>( \geq 2 ) hasChild</td>
</tr>
<tr>
<td>maxCardinality</td>
<td>( (\leq n , R) )</td>
<td>( \leq 1 ) hasChild</td>
</tr>
<tr>
<td>inverseOf</td>
<td>( R^{-1} )</td>
<td>\hasChild^{-}</td>
</tr>
</tbody>
</table>

Recent semantic web language is the Markup Language enumeration hence it can be categorised as Table 2.1:

$$\text{RDF} \{ \text{Room} \text{isClose} \text{isOpen} \}$$

The doors are inserted into the by the instances of the

$$\text{Room} \text{hasState} \text{isOpen} \text{isClose}$$

The concepts "( ```

$$\text{ALC} \text{FL} \text{Ontology reproduced from R. Hartanto, Fusing DL Reasoning with HTN Planning as a Deliberative Layer in Mobile Robots. PhD thesis, University of Osnabrück, August 2009.}$$

THE DOMAIN

MODELING THE DOMAIN

our initial state is **HUGE**

1. Model the domain
2. **Create the planning problem**
3. Generate a plan
4. Execute/Monitor it

Use DL to infer relevant aspects of the domain
1. Model the domain
2. Create the planning problem
3. Generate a plan
4. Execute/Monitor it
What if we have no watering can?
1. Model the domain
2. Create the planning problem
3. Generate a plan
4. Expand the domain and try again
5. Execute/Monitor it
EXPAND THE DOMAIN

Use affordances

...and Conceptual Similarity

m_WaterPlant(Plant)
precond: empty ?WCan

m_Get(?WCan)

o_GoTo(?WCan,ForGrasping)  o_Grasp(?WCan,ForTransport)

m_WaterPlant(Plant)
precond: full ?WCan

o_GoTo(?Plant,ForWatering)  o_Position(?WCan,ForWatering)  o_Pour(?WCan,ForWatering)

m_Fill(?WCan)

m_Regrasp(?WCan)

o_Place(?WCan,ForRegrasping)  o_Grasp(?WCan,ForPouring)

o_GoTo(?!KitchenSink,ForFilling)  o_Position(?WCan,ForFilling)  o_OpenTap(?ColdTap)  o_CloseTap(?ColdTap)  o_Grasp(?WCan,ForTransport)
**FUNCTIONAL AFFORDANCES**

watering can |ˈwɔːrɪŋˌkæn|
noun
a portable water container with a long spout and a detachable perforated cap, used for watering plants.
Can we determine a relation between quality dimensions and given tasks?

*Conceptual spaces* are composed by *quality dimensions*

Multi-dimensional feature space:
- **Points** denote objects
- **Regions** denote concepts

E.g. Capacity to hold water; handle; spout

Unique Instance
Common Instance
Same Functional Affordance & Conceptually Similar
Same Functional Affordance
Conceptually Similar
Inferred Conceptual Similarity

E.g. only “my_teacup”
Unique Instance

Common Instance

Same Functional Affordance & Conceptually Similar

Same Functional Affordance

Conceptually Similar

Inferred Conceptual Similarity

E.g. closest instance of a “teacup”

Increasing flexibility & Decreasing Constraints
Increasing flexibility & Decreasing Constraints

Unique Instance

Common Instance

**Same Functional Affordance & Conceptually Similar**

Same Functional Affordance

Conceptually Similar

Inferred Conceptual Similarity

E.g. closest object “for drinking from”, that matches “small, bowl-shaped, container, handle” (e.g. “mug”)
Increasing flexibility & Decreasing Constraints

Unique Instance

Common Instance

Same Functional Affordance & Conceptually Similar

**Same Functional Affordance**

Conceptually Similar

Inferred Conceptual Similarity

E.g. closest object “for drinking from” (e.g. “bottle”)
Unique Instance

Common Instance

Same Functional Affordance & Conceptually Similar

Same Functional Affordance

**Conceptually Similar**

Inferred Conceptual Similarity

E.g. “small, bowl-shaped, container, handle” (e.g. “measuring cup”)

Increasing flexibility & Decreasing Constraints
Unique Instance
Common Instance
Same Functional Affordance
& Conceptually Similar
Same Functional Affordance
Conceptually Similar

**Inferred Conceptual Similarity**

E.g. objects used “for drinking from” are usually “small, cylindrical, container, glass” (e.g. “jar”)
incomplete information about the environment

1. Model the domain
2. Create the planning problem
3. Generate a plan
4. **Execute/Monitor it**

Combine generated plans with action behaviors
Use Proximity

Find alternatives in case of failures

Take advantage of opportunities

EXECUTE/ MONITOR PLANS

m_WaterPlant(?Plant)
precond: empty ?WCan

m_Get(?WCan)

o_GoTo(?WCan,ForGrasping) o_Grasp(?WCan,ForTransport)

m_WaterPlant(Plant)
precond: full ?WCan

o_GoTo(?Plant,ForWatering) o_Position(?WCan,ForWatering) o_Pour(?WCan,ForWatering)

m_Regrasp(?WCan)

o_GoTo(?Plant,ForWatering) o_Position(?WCan,ForWatering)

m_Fill(?WCan)

o_Place(?WCan,ForRegrasping) o_Grasp(?WCan,ForPouring)

o_GoTo(?KitchenSink,ForFilling) o_Position(?WCan,ForFilling) o_OpenTap(?ColdTap) o_CloseTap(?ColdTap) o_Grasp(?WCan,ForTransport)

m_WaterPlant(?Plant)
precond: empty ?WCan

Tuesday, 11 June 13
1. Cluster behaviors by their effect on objects
2. Create one operator per cluster
3. Generate plans with these operators
4. Executed as the closest-matching behavior

To reduce complexity during planning
1. Receive command
2. Check plan library
3. Create planning problem
4. Generate Plan
5. Execute and monitor it
Design the plan library (including preferences)
Test domain expansion phase
Extend this to enable action substitution
Enable instantiation of affordance behaviors at execution time using Conceptual Spaces
Enable object substitution as tool usage
Enable the performance enhancement use case

Architecture design
Proof of concept integrating planning with execution & monitoring

Integration into our b-it-bots RoboCup @Home framework
Modeling functional affordances in DL
Abstraction hierarchy for action substitution
Extend planner to lift over functional affordances and use justification structures
THANK YOU