Size matters: Grounding quantifiers in spatial perception

Pauw, S.

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: http://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
Chapter 3
Planning What To Say Next: Grounded Language Processing

This chapter describes a fully operational language interpretation system that is created for robotic communicative interactions called Incremental Recruitment Language (IRL). Meaning in IRL is represented as a network (program) that contains instructions (cognitive operations). These operations can be seen as a set of instructions that allow an agent to fulfill a specific communicative goal. IRL contains a number of mechanisms to define, execute, compose, compare, and repair these IRL-programs. The goal of this chapter is to provide a detailed overview of the most essential IRL mechanisms. For implemented examples, see the IRL tutorial that is shipped with the Babel Lisp-package that can be found on http://www.emergent-languages.org/. This article together with the tutorial aim to provide the reader with enough information to start working with IRL.

This chapter is part of an ongoing effort to keep track of the evolution of the IRL system, as discussed below. Big parts of Section 3.2 and Section 3.3 are taken directly from the following publication: Spranger, M., Pauw, S., Loetzsch, M., and Steels, L. (2012b). Open-ended Procedural Semantics. In Steels, L. and Hild, M., editors, Language Grounding in Robots. Springer, New York.

3.1 Introduction

The main purpose of IRL is to bridge the gap between the continuous data that result from sensorimotor processing and the discrete concepts of language. Although IRL is often described as a language (as the name already suggests) or a formalism (Steels and Bleys, 2005), it might be better to conceive of IRL as a set of conceptualization mechanisms that are typically needed for robotic language experiments. Most of the mechanisms are independent of the rest of the system, so the experimenter can choose to only use the part of IRL that (s)he needs.

The early development of IRL took place at the end of the nineties. A first
implementation by Luc Steels was used in experiments in grammar emergence (Steels, 2000a). A second implementation was made by Wouter Van den Broeck (Van Den Broeck, 2008). This chapter is based on a more recent implementation by Martin Loetzsch, Simon Pauw and Michael Spranger (Spranger et al., 2010a). The current implementation has already been used in language game experiments targeting various domains including color (Bleys, 2008), spatial language (Spranger, 2011), quantifiers (Pauw and Hilfery, 2012) and temporal language (Gerasymova and Spranger, 2012a) on different robotic platforms, including the Sony humanoid (Fujita et al., 2005) and the Humboldt MYON (Hild et al., 2012) robot.

This chapter describes only the most essential mechanisms that make up IRL. The next section shows how meaning in IRL can be represented using networks of cognitive operations and semantic entities. The subsequent section illustrates the execution (i.e., interpretation) of IRL networks and their automatic composition (i.e., planning). The last section discusses the relation between IRL meaning and language, showing how IRL networks can be mapped onto linguistic constructions and in the case of noisy communication channels, how IRL can help to reconstruct the intended message (i.e., flexible interpretation).

3.2 Meaning Representation

IRL can best be seen as a set of independent mechanisms that are useful for modeling semantics grounded in the visual attention of robotic agents. As such, it is an open-ended system in which we can implement many different semantic paradigms. However, in order to understand the working of IRL, we focus our attention on one specific semantic model, which we will illustrate using the example utterance “the red block”.

The implementation of this example is based on robotic communicative interactions, called language games (Spranger et al., 2010b, see Figure 3.1). In this particular language game, the robot on the left takes the role of the speaker. He picks the red block in the center of Figure 3.1 as the topic of the language game. His communicative goal is to find an utterance that, when interpreted by the hearer (the robot on the right), draws the attention to that object. He could for example say: “the red block.” The hearer in turn interprets this utterance and points at the object he thinks the speaker intended. If he points correctly the game is a success.

In order to play such a language game, the robots are provided with computational systems for perception, conceptualization and communication that link language to the sensorimotor interaction of artificial agents. The role of IRL herein is to provide a mapping between the output of the perceptual system and the conceptual structure that the language system translates into an utterance. The perceptual systems for recognizing and tracking the objects in their envi-
3.2. Meaning Representation

Figure 3.1: Robots scan the shared environment (center image) with the cameras in their heads (images top left and top right) and construct world models from these data streams (images bottom left and bottom right). The robot at the left has the role of the speaker and tries to draw the attention of the other robot to the red block by means of the utterance “the red block”).

The world model produced by the vision system consists of a set of objects that are characterized by continuous real-valued features such as color, position, orientation, and dimensions. Part of the job of conceptualization is to move from these continuous values to the discrete categorizations used in language.

The respective world models of two different robots are similar, but never identical because of perceptual noise and different perspectives. Any conceptualization mechanism used in robotic interaction experiments should be robust against this imperfect perception.

In order to find an appropriate description for an object, the speaker tries to find a particular set of operations that, when executed by the hearer, will single out the object from the context. Consequently, the meaning of an utterance is a set of cognitive operations or procedures that the speaker wants the hearer to execute in order to fulfill a communicative goal. The explicit use of operations as part of the meaning is common in procedural semantics (see Winograd, 1971; Johnson-Laird, 1977; Woods, 1981, for original ideas).

More specifically, an utterance encodes a network of cognitive operations as well as the relationships between their arguments. An example network for
Figure 3.2: An IRL network representing the meaning of “the red block”. When executed by the hearer in the interaction shown in Figure 3.1 (right robot), the variable \(?\text{referent}\) (the referent of the utterance) becomes bound to the object \text{obj-252}.

the utterance “the red block” is shown in Figure 3.2. It includes operations such as filtering the context for blocks (\text{filter-set-class}) or finding red objects (\text{filter-by-color}). Every node in the network evokes a cognitive operation, represented by its name and its list of arguments, for example (\text{filter-set-class} ?\text{set-2} ?\text{context} ?\text{class}), evokes the \text{filter-set-class} operation. The arguments can be thought of as variables or slots that are bound to or will contain specific values. These slots are represented as names preceded by question marks. The same variable can re-occur in different cognitive operations, and this is represented by the arrows in the network.

These arrows are purely representational. The actual information about how the operations are linked are provided by the variable names. For example the first argument of \text{filter-set-class} is linked to the second argument of \text{filter-by-color} through the variable ?\text{set-2}, meaning that the hearer should first filter the context for blocks and then find all red objects in this set of blocks (“red block”).

There is a special operation called \text{bind} which introduces concrete semantic entities of a certain type and binds them to argument variables in the network. Semantic entities are categories in the conceptual inventory of the agents. For instance, the statement (\text{bind color-category} ?\text{color} \text{red}) in the example above binds the color category \text{red} to the variable ?\text{color}. The color category itself has its own grounded representation.

It should be pointed out here that IRL, although designed as a procedural semantics, has some aspects that are clearly more declarative in nature. First of all, the order of the operations has no effect on the order of execution, only the relations between the operations influence their execution. Secondly, IRL operations are multidirectional. From the example above, one might conclude that cognitive operations behave like procedures as known from programming language: computing an output from input arguments. But, as we will show in in the remainder of this Section, they can operate in multiple directions, so that IRL can in fact be seen as a constraint language (as pioneered in early programming
3.2. Meaning Representation

language designs of Borning, 1981; Sussman and Steele, 1980; Steels, 1982).

3.2.1 Operations

The basic building blocks of IRL are cognitive operations and semantic entities. The semantic entities represent any conceptual contents—such as prototypes, predicates, relations, and sets. The cognitive operations instruct the agent what to do with these semantic entities. In this section we describe these basic building blocks in detail. The present section discusses the working of the cognitive operations and shows how one can implement such operations. The section to follow focusses on the notion of semantic entities\(^1\).

A cognitive operation implements a specific cognitive function or task, for example filtering a set with a color category, picking elements from a set, categorizing an event, performing a spatial perspective transformation, taking the union of two sets, and so on. This is an example of how an operation for color categorization can be declared in IRL (we will show the full implementation later):

\[
\text{(defoperation filter-by-color ((target-set entity-set) (source-set entity-set) (color color-category))}
\]

This operation has the three arguments target-set, source-set and color, which are respectively of type: entity-set, entity-set and color-category.

In many ways, cognitive operations behave like functions in the sense that they compute a set of output arguments from a set of input arguments. For the present example we assume that color concepts are defined by a set of prototypical colors that carve up the color space. Now, the operation filter-by-color can be defined such that finds all the elements in source-set argument of which the color falls in the color-space region defined by the color argument. Every object for which this is the case, is returned in the target-set.

However, these cognitive “operations” are not really operations, but relations; they are multi-directional. For example the operation filter-by-color can also be defined to infer a color category from a target-set of classified objects and a source-set. And it can be defined to compute combinations of color categories and resulting target-set values from a source-set. As we will show later, this ability to operate in multiple directions is crucial for flexible conceptualization and interpretation of semantic structures.

When an operation is executed, some of its arguments are bound to a value. This value can be any semantic entity (see next Section) with a type that is

\(^1\)It should be noted that the IRL core system does not come with any built-in cognitive operations or semantic entities. The user of IRL will have to implement them for the particular experiment IRL only provides an interface for it.
compatible to the type of the argument specified in the operation. Whether an argument then is input or output of the operation depends on whether it is bound or not. To give an idea how these different cases of input-output relationships are concretely implemented in IRL, we will now show the complete implementation of the filter-by-color operation as used in this example:

```lisp
(defun filter-by-color ((target-set entity-set) (source-set entity-set) (color color-category))
  ;; Case 1
  ((source-set color => target-set)
   (let ((filtered-set (apply-color-category color source-set
                                                  (color-categories ontology))))
    (when filtered-set
     (bind (target-set 1.0 filtered-set))))))

;; Case 2
((target-set source-set => color)
 (loop for category in (all-color-categories ontology)
       when (equal-entity target-set
               (apply-color-category category source-set
                                       (color-categories ontology)))
       do (bind (color 1.0 category))))

;; Case 3
((source-set => color target-set)
 (loop for category in (color-categories ontology)
       for filtered-set = (apply-color-category category source-set
                                        (color-categories ontology))
       when filtered-set
       do (bind (target-set 1.0 filtered-set)
                (color 1.0 category))))

;; Case 4
((target-set source-set color =>)
 (let ((filtered-set (apply-color-category
                       (color-categories ontology))))
  (equal-entity filtered-set target-set))))
```

Most of this is general Lisp code, with the IRL specific code shown underlined. Under the definition of the operation (which we have already explained before) there are four cases, which each implement the behavior of the operation for a
3.2. Meaning Representation

different combination of bound/ unbound arguments. Each case starts with a pattern that defines its applicability: when all arguments before the \( \Rightarrow \) symbol are bound and all arguments after \( \Rightarrow \) unbound, then the code below the pattern is executed. For example, Case 1 specifies the operation of the primitive when source-set and color are bound, but target-set is still unbound.

Each case ‘returns’ values for all its unbound arguments with the bind command. For example in the first case, \((\text{bind } (\text{target-set } 1.0 \text{ filtered-set}))\) assigns the computed value filtered-set with a score of 1.0 to the argument target-set. In addition to that, an operation can call the bind command multiple times and thereby create multiple hypotheses. For example in the third case, the operation computes all possible pairs of values for the color and target-set arguments when only the source-set is bound. When multiple hypotheses are created, the scores are used to discern better from worse solutions.

It is also possible that an operation does not compute a value for an output argument. For example in the second case above, it can happen that the operation is not able to infer a color category which can account for a categorization of source-set into target-set. The operation will then simply not call the bind command, which invalidates the values bound to its input arguments. Finally, when all arguments of an operation are bound, then the operation does not bind any values at all but returns information on whether its arguments are consistent. In the fourth case, the operation checks whether the color category applied to the source-set is indeed the same as the given target-set.

3.2.2 Entities

The values that are bound to the arguments of cognitive operations are called semantic entities. These can be any kind of data representations, including items in the conceptual inventory of an agent (e.g. image schemata, categories, prototypes, relations, roles, etc.), representations of the current context (e.g. the world model, discourse information, etc.), and intermediate data structures that are exchanged between cognitive operations (e.g. sets of things, constructed views on a scene, etc.). In the example above, a semantic entity of type color-category consists of three numeric values that represent a prototypical point in the YCbCr color space. The memory of the agent contains several instances of color-category, for example red is represented by the point in the color space \([16, 56, 248]\). Furthermore, a semantic entity of type entity-set represents a list of objects, which each again contain numerical values computed by the vision system. The world model of an agent is also represented as an entity-set so that it can be used by operations such as filter-by-color.

Semantic entities are typed, which makes it possible to explicitly model intuitive distinctions between different cognitive representations. Such distinctions could for example be rooted in a perceptual system which already distinguishes between objects and events because they are recognized by different sub-systems.
Or it could be the difference between a color category and a discourse role, which clearly are meant to operate in different domains. Furthermore, types can be organized in hierarchies, which allows it to treat entities with a common super-type the same. Technically, type hierarchies are represented using the standard class inheritance system of Lisp (Kiczales et al., 1991), that is new types are defined by creating classes that are directly or indirectly derived from the IRL class entity. Using class inheritance additionally allows it to inherit properties from other classes of semantic entities and can be used for software engineering, in particular, designs with reuse.

An example of such a type hierarchy is shown in Figure 3.3. It shows semantic entities that were chosen for the examples in this chapter. As mentioned before, the vision system used here has different mechanisms for recognizing robots, boxes and blocks. Consequently, there are different classes of semantic entities for the output of these sub-systems, namely robot, box and block, which all are indirectly derived from a common ancestor sensory-entity. In addition to that, there is the class entity-set for sets of objects and discourse-role, color-category, spatial-category, selector for categories that work with different kinds of cognitive operations. However, this is only an example. How concrete type hierarchies are implemented and how they interface with the rest of an agent architecture is left to the user of IRL.

Type information is used in IRL in three different ways. First, it constrains what semantic entities can be bound to arguments of cognitive operations: only entities of the same type or of a sub-type of the type of the argument or can be bound to the argument of an operation. Second, it constrains the way in which cognitive operations can be combined in networks (see Section 3.3.2). And third, they can provide a seed for semantic and syntactic categories in the grammar that expresses semantic structures: an distinction on the semantic level between objects and events could be reflected in categories such as noun and verb (see Bleys, 2008; Spranger and Steels, 2012, for experiments in this direction).
3.2. Meaning Representation

It is common practice to provide agents with an ontology that can be used to access semantical entities. Every semantic entity (every object that inherits from the object entity) has an identifier field (id). These id’s are used by the semantic structure to refer to the semantic entities. In this example, the semantic entity red, has the id red. The statement (bind color-category ?color red) in Figure 3.2 uses this id to access that semantic entity.

3.2.3 Networks

Above we provided an intuitive explanation of the IRL network in Figure 3.2, representing the utterance “the red block”. The network is repeated below as a Lisp S-expression. We continue with this example to explain the mechanisms for evaluation and construction of this network in more detail.

```
((unique-entity ?referent ?set-1)
  (bind color-category ?color red)
  (filter-set-class ?set-2 ?context ?class)
  (bind object-class ?class block)
  (get-context ?context))
```

It contains four cognitive operations: unique-entity, filter-by-color, filter-set-class and get-context, and two semantic entities: red and block. The arguments of the operations are connected via variables (starting with a ?). Two or more operations are linked when they share the same variable. For example in the network above the first argument of the filter-set-class operation is connected to the second argument of filter-by-color through the variable ?set-2.

Semantic entities are introduced in a network with bind statements (starting with the bind symbol) and they are also linked to cognitive operations through variables. For example (bind color-category ?color red) binds the red color category to the color argument of filter-by-color via the ?color variable. The first parameter of the bind statement (here: color-category) declares the type of the semantic entity.

Figure 3.4 (repeated from Figure 3.2) shows the graphical representation of the network, with the links between operations and bind statements are drawn as arrows. Note that although the arrows suggest directionality, they only represent a canonical direction of execution, which nevertheless is often very different from the actual data flow in the network. Furthermore, the order of operations and bind statements in a network is not meaningful at all. All that matters is how operations and semantic entities are linked. Two networks are equivalent when both have the same set of operations and bind statements and when the structure of the links between them is the same.
Figure 3.4: Graphical representation of an IRL network underlying “the red block”. The get-context operation binds the set of all objects contained in the world model to the variable ?context. Then filter-set-class filters this set for all objects of class block and binds the result to ?set-2. This set is then filtered by filter-by-color for objects that match the red color category into ?set-1. Finally, unique-entity checks whether ?set-1 contains only one object and binds the result to ?referent.

3.3 Meaning Processing

The previous section shows how to define IRL-networks and all its components. This section focuses on the mechanisms for executing and automatically composing such networks.

3.3.1 Interpretation

Which particular red block in the example in Figure 3.4 is referred to, i.e. which object is bound to the variable ?referent, is found by executing the network within the current context of the interaction. Execution is the process by which values are bound to the variables in the network. A set of variable-value bindings is considered a solution, if it is complete and consistent. Complete means that all variables are bound. A set of bindings is consistent if all operations in the network have been executed.

The execution process starts by executing the bind statements to yield a list of initial bindings. The semantic entities expressed in bind statements are retrieved via their id and bound to the respective variables in the network. All other variables are assigned an empty value (unbound). As shown in the leftmost node of Figure 3.5, the initial bindings for the execution of our example network map the semantic entity red to the ?color variable and block to ?class, with the rest of the variables remaining unbound.

Execution of the network proceeds by executing all cognitive operations in the network. In each step, an operation is picked from the list of not yet executed operations and it is checked whether the operation can be executed given the current set of bindings for its arguments, i.e. whether it has implemented a case for that particular combination of bound and unbound arguments. If such a case
Figure 3.5: Example of an execution process. The network from Figure 3.4 is executed by the hearer in the interaction of Figure 3.1 (right robot). From left to right, each node represents a step in the execution process. From top to bottom, the executed operation, the node status, and the current list of bindings of each node are shown. A consistent solution with bindings for all variables is found in the last node, and the value obj-252 is indeed a unique red block (compare Figure 3.1).

exists, then the operation is executed (see Section 3.2.1) and newly established bindings are added to the list of bindings. If not, then another operation is tried. A consequence of this procedure is that the particular order in which operations are executed, the control flow, can not be determined by the structure of a network alone. Rather, IRL execution is data-flow driven and execution order depends on how data spreads between cognitive operations.

In the example of Figure 3.5, the only operation that can be executed given the initial bindings is get-context (it doesn’t require bound input arguments) and it introduces the entity set context-3 as a value for ?context. Then filter-set-class can be run, and so on. Each added binding enables the execution of more operations, until the unique-entity adds a binding for the last remaining unbound variable ?referent. The set of bindings in the right-most node of Figure 3.5 is a consistent solution of the execution process, because all operations in the network have been successfully executed and all variables are bound.

Of course there can be also other outcomes of executing operations than in the example above (see Section 3.2.1). First, it can happen that an operation returns multiple hypotheses for its unbound arguments. IRL will then add each hypothesis to a copy of the current bindings list and then further process these lists in parallel. Second, when all arguments of an operation are bound, then its execution amounts to a verification or checking of consistency. If that fails, then the complete set of bindings is invalidated and not further processed. And third, when an operation is not able to bind a value for an unbound argument, then the whole bindings set is also invalidated.

To illustrate this, we will now look at the execution of a second network. It has the same operations and the same connections between them as the previous example, but does not contain bind statements for ?color and ?class. Instead, the ?referent variable is bound to object obj-268 (the red block in the world model of the speaker, see Figure 3.1):
Figure 3.6: Example of an execution process with parallel processing of multiple hypotheses.

### 3.3.1. Example.

```lisp
((bind sensory-entity ?referent obj-268)
  (unique-entity ?referent ?set-1)
  (filter-set-class ?set-2 ?context ?class)
  (get-context ?context))
```

It is unlikely that such a semantic structure will be the result of parsing an actual utterance, but as we will see in the next section, the execution of such networks is heavily used in conceptualization. (The goal of conceptualization is to find a set of concepts that best describe a specific objects.) The execution process for this network in the world model of the speaker in Figure 3.1 is shown in Figure 3.6. Execution again starts with the get-context operation, but then another case of filter-set-class is executed: because both its source-set and class arguments are unbound, the operation creates combinations of object classes and resulting filtered sets, which leads to a branching of the execution.
3.3. Meaning Processing

process. The first two of these branches (Figure 3.6 top) immediately become invalidated, because \textit{filter-by-color} cannot apply color categories to boxes and robots. The third case, however, is further branched by \textit{filter-by-color}, because the set \texttt{block-set-5} bound to \texttt{set-2} contains both yellow and red objects. The first of these two hypotheses is then invalidated by \textit{unique-entity}, because \texttt{entity-set-16} contains more than one object. A consistent solution is then found with the node at the bottom right of Figure 3.6.

3.3.2 Conceptualization

We have seen how compositional semantics are represented and executed in IRL and will now turn to the use of these mechanisms in communicative interactions, i.e. how meanings are constructed and interpreted and how underspecified semantic structures can be completed.

For structured procedural meanings such as IRL programs, conceptualization is the process of constructing a network that, when executed, can achieve a specific communicative goal. For instance, the communicative goal can be to discriminate \texttt{obj-268} in Figure 3.1 (i.e. the red block). This goal can be achieved by the following network:

\begin{verbatim}
3.3.2. Example.
((unique-entity ?referent ?set-1)
 (filter-set-class ?set-2 ?context ?class)
 (get-context ?context)
 (bind object-class ?class block)
 (bind color-category ?color-prototype red))
\end{verbatim}

The mechanism that takes care of finding such a network is called the \textit{composer}. The composer is implemented as a standard best first search algorithm. Starting from an initial (usually empty) network, cognitive operations are recursively added and linked until a useful network is found. Moreover, the composer can also use complete or incomplete networks in the process of composition.

An example of such a composition search process is shown in Figure 3.7. Each node in the search tree contains an (intermediate) IRL program together with a target variable and a set of open variables and a number indicating the cost of that node.

The target variable of the chunk in composition is the variable that is linked to the first slot of the first operation that is added by the composer (thus there is always only one target variable per network). Open variables are all other variables in the network that don’t link cognitive operations. Additionally, the types of the slots of cognitive operations that are connected to target variables and open variables are also stored with the network. The cost of a node is used
Figure 3.7: Example of a search process involved in the construction of an IRL program. Analogous to previous examples, the goal for this conceptualization process is to find a program that can identify the red block in the scene of Figure 3.1. Each node represents one processing step and branches in the tree indicate multiple possibilities for expansion. Node labels show the order in which nodes were created, a score that determines which node should be expanded next, and a list of the cognitive operations that have been incorporated into the network so far. Starting from an empty network (node 1), cognitive operations are recursively added and the resulting programs are tried out (nodes 2-3, 7, 9), until finally a solution is found that can achieve the goal of identifying the red block (node 14). By then, some nodes have not been tested yet (nodes 6, 10, 12, 13, 16-18) and some cannot be further expanded (nodes 5, 8, 11, 15, 19).

to determine which node to expand next. The one with the lowest cost is the first to be expanded. Example 3.3.3 shows the internal network of node ‘4’ in Figure 3.7:

3.3.3. Example. Node 4

\[
((\text{unique-entity} \ ?\text{topic} \ ?\text{set-1})
\quad (\text{filter-by-color} \ ?\text{set-1} \ ?\text{set-2} \ ?\text{color}))
\]

The target variable is ?topic (of type sensory-entity) and the open variables are ?set-2 (type object-set) and ?color (type color-category).

The search process starts from an initial node. The content of the initial node depends on the communicative goal but should always contain at least one open variable. In our example the first node contains nothing but the open variable ?topic.

Every iteration of the search procedure consist of two phases. In a first phase the composer checks if the current networks can achieve the communicative goal.
3.3. Meaning Processing

For this, the conceptualizing speaker takes itself as a model for the hearer and executes the program using his own set of categories and his own perception of the world. This is a form of re-entrance (Steels, 2003). If one of the current networks can achieve the communicative goal then the composer is done and the solution is returned. The execution of a network can generate additional bindings. These additional bindings become part of the solution. Most of the time the networks will not provide a solution.

If no solution has been found yet, the composer tries to extend the network of the node with the lowest cost. The composer tries to add a cognitive operation to the existing network and links the target slot of the cognitive operation to one of the open variables of the node. This variable can only be linked if its type is compatible with the type of the target slot. For each possible extension, a child node is created with the extended network, the now connected variable is removed from the list of open variables and new open variables for the other slots of the added operation are created.

A solution of the conceptualization process is found when the execution of a node’s network results in a set of bindings. The processing of nodes stops and the found program together with the bindings from execution is returned. However, often there is more than one solution and sometimes the first solution found is not the best solution. Therefore it is possible to ask the composition engine for multiple (or even all) solutions up to a certain search depth.

We will turn to an example to illustrate the expansion of a node. In Example 3.3.3, the open variable \( \text{?set-2} \) (of type \( \text{object-set} \)) of node ‘4’ can be connected to the three different operations \( \text{filter-set-class} \), \( \text{get-context} \) and \( \text{filter-by-color} \) because their target slot is of type \( \text{object-set} \) (which is the same as and thus compatible with the type of \( \text{?set-2} \)). Consequently, three child nodes are created for the three resulting networks (nodes 9-11). Node 9 contains the following expansion:

3.3.4. Example. Node 9

\[
\begin{align*}
&((\text{unique-entity} \ ?\text{topic} \ ?\text{set-1}) \\
&\quad (\text{filter-by-color} \ ?\text{set-1} \ ?\text{set-2} \ ?\text{color})) \\
&\quad (\text{filter-set-class} \ ?\text{set-2} \ ?\text{set-3} \ ?\text{class}))
\end{align*}
\]

Its open variables are \( ?\text{color} \) (of type \( \text{color-category} \)), \( ?\text{set-3} \) (type \( \text{object-set} \)) and \( ?\text{class} \) (type \( \text{object-class} \)). The expansion of node ‘4’ removed \( ?\text{set-2} \) from the list of open variables but added \( ?\text{set-3} \) and \( ?\text{class} \). This network does not yet compute the topic. In order for the operation \( \text{filter-set-class} \) to compute something it requires a value for \( ?\text{set-3} \). But, \( ?\text{set-3} \) is still an open variable. However, a further expansion of node 9 into node 19 does give a solution:

3.3.5. Example. Node 19

\[
((\text{unique-entity} \ ?\text{topic} \ ?\text{set-1})
\]

...
(filter-set-class ?set-2 ?set-3 ?class))
(get-context ?set-3))

For the topic of this example (the red ball – obj-268 in Figure 3.1). IRL finds an unambiguous set of bindings containing the values red and block for the variables ?color and ?class respectively, which was already hinted at in Example 3.3.2.

The composition process of IRL is highly customizable to the specific needs of particular learning scenarios. Most importantly, the order in which nodes are processed can be influenced by providing a function that ranks them depending on the composed program and their depth in the search tree. Nodes with a lower rank will be processed first (see the second number in the node representations in Figure 3.7). By default, networks with a low depth in the tree, few duplicate cognitive operations and a smaller number of open variables are preferred, resulting in a ‘best first’ search strategy. But this scoring mechanisms can also be used to implement depth-first or breadth-first searches.

3.4 Communication

So far we have only looked at meaning representation an processing, but we have not yet discussed how all of this relates to language. In this section we address the question how IRL-networks are translated into utterances and vice versa. IRL is independent of any specific language formalism, and there is more than one way IRL meaning could be mapped onto language. However, IRL is developed in lockstep with Fluid Construction Grammar (FCG, Steels et al., 2012a), which makes FCG the go-to candidate for parsing and producing IRL-networks. But any construction grammar would serve equally well (ECG for example, Bergen and Chang, 2005), and in many cases it would even be possible to use IRL in conjunction with categorical grammars (e.g., Steedman and Baldridge, 2009). Rather than focussing on the implementation of IRL meaning using a specific formalism, this section aims to illustrate at a conceptual level some common practices in IRL grammar design.

The second part of this section focusses on an IRL-native system called flexible interpretation. IRL is designed for robotic language games. Inherent to these language games is that communication is not always perfect. The hearer is often presented with an incomplete message—either due to transmission noise or because the hearer doesn’t know all the words or syntactic constructions that the speaker uses. In such cases, the hearer can only retrieve part of the intended IRL-network. The flexible interpretation mechanism in IRL is designed to recover the missing parts of the network.
3.4.1 Grammar

In order to understand how IRL meaning can be mapped onto language, it is useful to point out that there are three different sources of information encoded in the semantic structure: semantic entities (bind-statements), cognitive operations, and the links between operations/bind-statements. Typically, these different sources of information all have their own particular role to play in the description of the grammar: Bind-statements are expressed by content words. For example, the word block maps onto the bind-statement (bind object-class ?class block). Cognitive operations and the variable links are expressed by grammatical constructions. For example, the operation (filter-set-class ?set-2 ?context ?class) is associated with the grammatical class CN.

![Diagram of syntactic structure of “the red block”](image)

Figure 3.8: Syntactic structure of “the red block”. The example grammar contains four grammatical categories: common noun (CN, e.g., block), adjective (ADJ, e.g., red), determiner (DET, e.g., the) and noun phrase (NP, e.g., “the red block”). The grammar also contains two rules for contracting utterance: The CN→ADJ/CN rule that constructs complex common nouns out by adding chaining adjectives to a simple common noun (e.g., “red block”), and the NP→DET/CN rule that constructs noun phrases by adding a determiner to a common noun (e.g., “the red block”).

The syntactical side of the grammar that produces the utterance “the red block” can be represented as a rewrite grammar containing rules to produce complex common nouns and noun phrases. Figure 3.8 shows an example of the tree structure that this grammar produces. The grammar contains a rule (CN→ADJ/CN) that combines the ADJ red and the CN block into the complex CN “red block”. Another rule (NP→DET/CN) adds the DET the to the common noun to create the NP “the red block”.

The question remains: How does such a grammar map onto IRL-networks? Traditional categorical grammars would link the grammatical categories to lambda expressions that when combined will produce the desired meaning. Although it would not be impossible to create IRL-networks this way, practice shows that categorical grammars are too restrictive. Due to their more flexible nature, construction grammars turn out to be more practical for this purpose. The essence
of construction grammars is the direct link between any linguistic construction, be it a word order constraint, a lexical item, a marker or any other syntactic construction, and the meaning. Any linguistic construction maps onto part of the semantic content.

Figure 3.9 shows the mapping between syntax and meaning for the NP “the red block”. In the case of content words this mapping is fairly straightforward. The word block maps onto the bind-statement that introduces the block prototype (bind object-class ?class block). Similarly the word red maps onto the bind-statement (bind color-category ?color red). The grammatical categories themselves provide the instructions of how the content words would be used. For example, the fact that block is used as a noun introduces the part of the network that picks all the blocks from the environment ((get-context ?context) (filter-by-class ?set-2 ?context ?class)). The grammatical constructions that create the syntactic structure of the utterance provide the links between different parts of the network. For example, CN→ADJ/CN provides the variable link (?set-2) that is responsible for linking the parts of the network from the CN block and the ADJ red.

This is just one example of how semantic structure in IRL can be mapped onto syntactic structure. More elaborate examples can be found in Gerasymova and Spranger (2012b) for temporal language, in Spranger and Loetzsch (2011) for spatial language and in Bleys (2008) for color. Approaches differ in their implementational details, but on a conceptual level they all follow the present outline.

### 3.4.2 Flexible Interpretation

One advantage of using FCG over any other language formalisms to encode language is its robustness against imperfect communication. When parts of an utterance are not recognized, by the language system, FCG proceeds to parse as much of the utterance as it possibly can. So it will still recover a partial meaning. Such robustness is only helpful if the agent have some mechanism to repair this partial meaning. The presence of such a mechanism (called flexible interpretation) is one of the thinks that makes IRL such a useful system for robotic communication.

Let us consider an example. The speaker says “the yellow block right of you”, however the hearer for some reason hears the following phrase:

3.4.1. Example. “grrgh yellow block right krkks you ”

When the hearer knows English, this utterance has some recognizable elements for the hearer. Like “block”, “yellow”, “you” and “right”, But misses “the”, and “of”. A language system that parses this sentence can at best retrieve only some of the intended network. Figure 3.10 shows an example of a network that might be the result of parsing this utterance.
4.2. Flexible Interpretation

will show at a conceptual level how IRL meaning is typically mapped to language
serve equally well (ECG for example).

3. Meaning Processing

– Cognitive operations and the variable links are expressed by grammatical con-
– Typically, bind-statements are expressed by content words. E.g., the word
– Any language formalism that can encode these types of information is in prin-

3.1. Interpretation

• Three types of information in semantic structure: semantic entities

3.2. Conceptualization

• Grammatical Constructions

(c) Grammatical Constructions

Figure 3.9: The mapping between syntax and meaning. Content words provide the bind-statements (a). The grammatical categories provide the cognitive operations that link to those bind-statements (b). And the rules that that create syntactic structure provide the linking between the previously created subnetworks (c).
Chapter 3. Language Processing

Figure 3.10: A partial network for "block yellow you".

Executing this network leads to no result (solution). However, the hearer can actively reconstruct possible meanings using the composer. The composer starts with an empty network, just as in conceptualization, and gradually adds cognitive operations building up more and more complex IRL networks. If an IRL network matches with the meaning obtained so far, then it is a possible interpretation of the phrase. If such the IRL network, furthermore, executes then the result of execution is considered a possible solution.

Matching

The most important operation in interpretation is matching. The matching algorithm tests whether a particular IRL network built by the composer is compatible with the meaning that the language engine parsed from an utterance. In the next few paragraphs the word meaning is reserved for the structure parsed by the language engine (so not the reconstructed meaning). Meaning is always matched against the (intermediate) IRL networks constructed in composition.

The interpretation mechanism has to recover networks that were not properly communicated. The solution that is matched with the meaning can therefore contain additional cognitive operations and variable links. While allowing the composer to add information, the matching process has to preserve the information provided by the meaning. In other words, the IRL network that the composer finds is only a valid interpretation if it contains at least as much information as provided by the utterance.

To understand the matching process, recall that, during parsing, language constructions add meaning to the overall interpretation in the form of 1) bind statements, 2) cognitive operations, and 3) variable links. A solution in interpretation found by the composer can include additional information, but must
preserve these three aspects from the parsed meaning. Consequently, the composer has to find a network that contains at least the cognitive operations and the variable links of the meaning. In addition, the open variables have to match the bind statements of the meaning. These intuitions are captured by the following definition:

A meaning \( n \) **trivially matches** an IRL network \( c \) iff (1) for each bind statement \((\text{bind type } \?\text{variable entity})\) in \( n \) there is a open variable \((\?\text{variable . type})\) in \( c \) and (2) every primitive \( p \) in \( n \) is in \( c \).

A meaning \( n \) **matches** IRL network \( c \) iff there is a function \( f \) from the variables in \( n \) to the variables in \( c \) such that \( n' = f(n) \) trivially matches \( c \).

where \( f(n) \) is the meaning \( n' \) that is the result of substituting every variable \( x \) in \( n \) for \( f(x) \).

For example, the parsed meaning for the utterance “block” is \((\text{bind object-class } \?\text{class block})\). This matches Network 3.4.2, but not Network 3.4.3. This is because the open variable \( ?\text{class} \) in Network 3.4.2 is of type \text{object-class} which matches the object class of the bind statement. The type of the open variable \( ?\text{color} \) in 3.4.3 is not correct.

3.4.2. Example.

\[
((\text{unique-entity } \?\text{referent } \?\text{set-1})
  \ (\text{filter-set-class } \?\text{set-1 } \?\text{context } \?\text{class})
  \ (\text{get-context } \?\text{context}))
\]

3.4.3. Example.

\[
((\text{unique-entity } \?\text{referent } \?\text{set-1})
  \ (\text{filter-by-color } \?\text{set-1 } \?\text{set-2 } \?\text{color})
  \ (\text{get-context } \?\text{context}))
\]

As a second example, consider the utterance “the block.” FCG computes the following meaning:

3.4.4. Example.

\[
((\text{unique-entity } \?\text{referent } \?\text{set-1})
  \ (\text{filter-set-class } \?\text{set-1 } \?\text{context } \?\text{class})
  \ (\text{bind object-class } \?\text{class block})
  \ (\text{get-context } \?\text{context}))
\]

This meaning matches Network 3.4.5, but it does not match 3.4.6 and 3.4.7. In Network 3.4.5, all the primitives and variable bindings of the meaning are also in the network, and the open variable \( ?\text{class} \) matches the bind statement \((\text{bind object-class } \?\text{class block})\) from the meaning. Network 3.4.6 does not match because the primitive \( \text{filter-set-class} \) is present in the meaning but not in the network. Network 3.4.7 does not match because it does not have the variable link between \( \text{filter-set-class} \) and \( \text{get-context} \).
3.4.5. Example.

((unique-entity ?referent ?set-1)
 (filter-set-class ?set-1 ?context ?class)
 (get-context ?context))

3.4.6. Example.

((unique-entity ?referent ?context)
 (get-context ?context))

3.4.7. Example.

((unique-entity ?referent ?set-1)
 (filter-set-class ?set-1 ?set-2 ?class)
 (get-context ?context))

3.5 Discussion

IRL has been built to support the embodied, multi-agent experiments in language evolution. This chapter discusses the mechanisms provided for autonomous conceptualization and interpretation. Namely, a mechanism for the evaluation (or execution) and composition of semantic structure, and a mechanism to reconstruct incomplete semantic structure. IRL, thus, provides the needed connection between the sensorimotor systems and the language systems, at the same time allowing for learning and open-ended adaptation.

As suggested earlier in this chapter, IRL can be seen as a procedural semantics. The semantic building blocks are procedures that conduct the hearer into achieving a communicative goal. And, although IRL is a novel approach to language modeling, the notion of procedural semantics is not. The notion of procedural semantics dates back to the seventies, to the first attempts to make computers understand human language (Winograd, 1971). The program sparked some heated debate Johnson-Laird (1978); Fodor (1978, 1979); Winograd (1975), but also culminated in a number of fruitful practical endeavours with actual systems being build. One example of such a complete system for parsing and interpreting natural language utterances is SHRDLU Winograd (1971); Hewitt (1969).

The main qualm its opponents have with procedural semantics is the lack of formal rigor (Fodor, 1978, 1979). More recent approaches in relevance theory, addressed this issue and provide a more formal version of procedural semantics (Blakemore, 1992; Wilson and Sperber, 1993; Blakemore, 2002). But, these approaches loose a lot of the procedural force. They consider procedures to be guiding the inference process of otherwise propositional meaning. Procedures, in this view are mostly an afterthought. IRL is in that sense much more reminiscent of SHRDLU then of contemporary approaches. And indeed this come at the
cost of less formal rigor. I think an important future The only argument we can make in defense simply not required for the purpose of IRL. IRL is not meant as theory, it is not designed to make predictions about language and language use. IRL is meant as a collection of mechanism that are useful for conducting grounded language experiment. And this it does well.