How grounded defeasible rules and exceptions emerge:

Conceptualizing the environment in a population of agents

Heiko Harders

March 2010

Master Thesis
Artificial Intelligence
Department of Artificial Intelligence

First advisor:
dr. B. Verheij (Artificial Intelligence, University of Groningen)
Second advisor:
prof. dr. L.C. Verbrugge (Artificial Intelligence, University of Groningen)

university of groningen

Abstract

We are able to use complex concepts that describe our environment. Exceptions are part of these concepts. Think for example about the platypus that we consider as a mammal but not as a bird, although the platypus has some of the typical properties of a bird. How are such exceptions created? In this work an agent model was developed to investigate how defeasible rules with exceptions can emerge in a population. The agents learn rules and exceptions about their environment by playing games. Using these rules a set of concepts is learnt that is grounded in the environment. The novelty of this model is the use of defeasible rules and exceptions. These rules have simple forms, with a conjunction of properties in the antecedent and a single concept as conclusion. Rules can be defeated under exceptional circumstances. In that case the conclusion of the rule cannot be reached anymore by using the defeated rule. A method was developed to iteratively learn and update such rules. Experiments with the model show that the agents are able to reach communicative success using the defeasible rules. Concepts that contain exceptions emerge in some occasions from the interactions.

Contents

List of Tables

List of Figures

List of Games

Chapter 1

Introduction

We have conceptualized our world to a great extent. In every day language we use the concepts we have created to interact with other people, thereby making clear what our intentions are and what we are talking about. While interacting with other people we expect them to understand us and thus we expect others to understand the concepts we are using to describe everything we are talking about. We use concepts for everything that occurs in our world and we have even concepts for abstract things. Example of such concepts are things we have created ourselves such as doors, windows, buildings and houses. Other examples are living things such as plants, animals, mammals, birds, vertebrates. More abstract examples are mathematics, laws and religions.

We are able to understand and use an enormous amount of concepts. For example, when we are talking about a door, we expect others to understand what a door is. Because there is a large variety of different doors, the concept door must be a very general one that can even be used for doors we have never seen before. When we are giving somebody directions to go into the hallway on the right and take the third door on the left we expect them to understand what we mean without explaining what the hallway looks like or explaining in detail what each door in the hallway looks like.

Not only are we able to understand and use concepts, we also know how concepts relate to each other and when something is an exception to a concept. Take for example the platypus which shows features that are shared with birds, like having a beak and laying eggs. However the platypus also has features that are shared with mammals, like having mammary glands. Therefore there are reasons to consider the platypus as an example of the concept bird, but also reasons to consider it as an example of the concept mammal. However, we know that the platypus is a mammal and not a bird. Therefore this animal is an exception to birds. There are many other examples, we know that doors can be opened unless they are locked. Or we can switch on the light unless the light is broken or unless there is a power failure.

How can we explain this extreme amount of conceptualization of our world? And what is the base of our knowledge that allows us to make these concepts? One way to explain the ability to use very general concepts but at the same time allow exceptions is using defeasible rules. A general rule describing mammals could look like: ‘if an animal has mammary glands, it is a mammal’. For birds there could be a rule that states: ‘if an animal lays eggs, it is a bird’. There

are several advantages using such rules. The most important advantages are that these rules are both simple and general. They are simple because only one property of the animals is involved and they are general because they work for almost all animals that are mammals or birds. When we assume that the simplest, most general rules are also the easiest rules to learn, there is a huge advantage of having easy to learn rules that cover many objects that belong to a concept.

More specialized rules are needed for the exceptional situations. An example of an exceptional situation is given by looking at the platypus. When talking about the platypus we are talking about a very specific example of animals. The general rule about birds that was given above will fail when used for the platypus. However this is not a very big deal because animals for which the rule does work are observed far more often. To correct the situation the general rule about birds should be prevented to be used for the platypus. An exception can be used to achieve this. Such an exception could be: 'if an animal lays eggs and it has mammary glands it can not be concluded that it is a bird'. This exception involves checking two properties and thus the exception is more complex than the general rule about birds and it applies to a more specific set of objects. Being more complex, one could argue that the exception is harder to learn. However, as stated before, this is not a big issue because situations in which specific rules are needed occur less often. To understand each other it is most important to learn the general rules quickly. Rules for situations that do not occur as often do not contribute much on understanding each other in the most occurring situations.

The afore mentioned reasons suggest that using defeasible rules could be a good way to create robust concepts that are easy to learn and use. To test this hypothesis an agent model was developed that simulates the emergence and use of concepts based on defeasible rules and exceptions. In this model the defeasible rules and exceptions that are learnt by agents are based on the logical system DefLog that was proposed by Verheij (2003). In this system rules are prima facie justified, but there could be a reason to reject a rule. In case an exception exists that rejects a rule, the rule is said to be defeated and can no longer be used to reach a conclusion. Therefore the system is non-monotonic and it can be used to create general rules that do not apply in more specific situations. It also allows creating exceptions that defeat the general rules when necessary.

To investigate the emergence of concepts when using defeasible rules and exceptions, it is undesirable to provide prior knowledge to the system. Instead concepts must emerge without relying on any prior knowledge. An excellent way of emerging knowledge in an agent based system is described by Steels (1999). In his Talking Heads experiment, Steels uses robots to generate language in a shared environment by letting the robots play language games. An advantage of Steels' experiment is that new words in the language are grounded in the environment because the robots are playing the language games in and about their environment. Similar games will be used in the current work to let a population of software agents evolve a set of concepts that can be used in their environment. This allows research about concepts that evolve when relying on defeasible rules and exceptions to describe the concepts. In this work the emergence of concepts will be investigated, looking at the meaning of concepts, the spreading of a concept through a population and the emergence of concepts

with exceptions.

Another feature of Steels' Talking Heads experiment is uncertainty about meanings. When two robots are playing a language game, they never directly communicate the meaning of words they are using. The only thing they can observe is whether they agree with each other on the outcome. This means that two players are not aware of the rules that form the meaning of a word, they only know for which object the word was used. This causes uncertainty between robots about the meaning of words. Similar uncertainty will be used in the currently discussed work. The advantage is that concepts undergo an evolution in their development. When agents see more objects that are examples of a concept, they will gain certainty about the meaning of the concept. They will never be entirely sure though whether they mean the same as other agents in the simulation.

Uncertainty about rules and concepts means that some rules can turn out not to be working very well. These are rules that cause a lot of disagreement with other agents. Therefore agents will be measuring the quality of the rules they are using. A rule that often leads to success will have a high quality while a rule that often leads to disagreement will have a low quality. Agents can use this quality measure to prefer rules with a high quality to improve their chances on successful outcomes of games. Because each agent plays games with multiple other agents, each agent will have a unique history of interactions. Combined with the uncertainty about rules for concepts, the advantage is that each agent has to develop the rules for a certain concept in different circumstances. If the agents succeed to agree about a concept, this will be the result of a broad range of different interaction histories. Using larger populations will mean that more different agents that each have a unique history of using and learning rules about the concept are involved.

To force agents to come up with efficient rules, each agent will have a limit on the number of rules it can remember. Remembering less rules means more competition between rules: if a rule has a low quality there is a high chance it will be forgotten. At that point there will be room for a new rule which hopefully lets the agent agree more often with other agents. Knowing less rules therefore could turn out to be an advantage for agents.

The features described above allow a model that can be used to investigate many properties of using defeasible rules and exceptions to conceptualize a shared environment. Many of these features have been used separately in other research. But to my knowledge an attempt to combine them in a single model has not been made before. The novelty of this work therefore is creating a plausible model for the emergence of concepts using defeasible rules and exceptions, while allowing very general concepts but also very specific concepts at the same time. It is hoped to find evidence that defeasible rules and exceptions provide a solid base for the emergence of concepts.

To find this evidence several experiments will be performed with the model. In these experiments the effects of different population sizes, different memory sizes and different environment sizes for games will be investigated. Larger populations have the ability to explore more rules and concepts which could lead to better performing concepts. On the other hand spreading of rules and concepts will be harder in larger populations. It is expected that smaller memory sizes will lead to more efficient rules, which in turn causes higher average success in games played by agents. Having more objects in the environment will make

games more complex for agents, therefore creating successful rules will probably be harder in these cases. Besides looking at the performance of agents in several different circumstances, the emergence of concepts and exceptions will be discussed in detail.

The main question in this work will be whether there is evidence that using defeasible rules and exceptions can form a plausible base for the emergence of successful concepts grounded in the environment. An answer to this question is hoped to be found by creating a model and experimenting with the model that incorporates:

1. Defeasible rules and exceptions to describe concepts.
2. Simple general rules and more complex rules for specific concepts.
3. Grounding of concepts in a shared environment.
4. A population of agents that all have a unique interaction histories.
5. Uncertainty of agents about rules, they can only observe the effect of rules.
6. Agents that can measure the quality of rules.
7. A limit on the memory of agents to force creating effective rules.

It is hypothesized that creating a model based on these key features will result in successfully communicating populations of agents that use concepts to describe their environment. Therefore it is expected that using defeasible rules and exceptions forms a plausible base for emerging and using concepts about the environment.

This work continues with discussing existing research that incorporates several of these key features separately. Thereafter the created model will be explained in detail, the performed experiments will be explained followed by a discussion of results, possible improvements and suggestions for further research.

Chapter 2

Literature

In the first chapter a list was presented containing key-features and ideas that were considered important in emerging concepts using defeasible rules and exceptions. Some of these features were previously described in other work. In the following sections an overview will be presented of work that is related to defeasible rules and other key-features of the model that is described in the current work. The first topic that will be covered is which techniques can be used to create emergent behaviour. Secondly some work about defeasible rules is described.

2.1 Emergent behaviour in multi-agent models

When creating a multi-agent model that incorporates emergent behaviour, it is often hard to predict the exact behaviour that follows from actions performed by agents. While these actions often seem fairly simple themselves, the behaviour of the entire population can be surprisingly complex. To control the behaviour of a population of agents, one has to consider the consequences of local interactions between agents. While predicting the behaviour of the population is hard, there are many ways to give direction to the emergence of behaviour. To make sure the desired behaviour can be reached by a population of agents, a model must be build that steers the behaviour of agents by rewarding correct behaviour or punishing behaviour that leads to failure. If the correct rewarding schemes are used in a model, this will cause the reinforcement of desired behaviour which in turn leads to the emergence of the desired behaviour. A lot of research has been done on exploiting emergent behaviour in multi-agent models. This research can be used to acquire some important insights in successful strategies for achieving the desired behaviour in populations of agents. A selection of existing work will be discussed in the following sections. This work is related to the key-features and ideas that were discussed in the previous chapter.

2.1.1 Evolving languages

Some excellent work on emergence of language is done by Steels. In (Steels, 1998) the Talking Heads model is described that allows agents in a population to learn language with a syntax. The language is obtained through different

games the agents play in a shared environment. This causes the language to be grounded in the environment. Agents are computer programs that are placed into an actual robot that is equipped with a camera. Two of these robots are placed in a scene and they can perceive objects in their environment using their camera's. The games played by the agents are adaptive language games, meaning that the agents adapt their internal structures to be more successful in future games.

The agents in these games perceive properties of the objects in their environment. For example the x and y positions of the objects in the perceived image, or the area the object covers in the image. One of the objects in a scene will be the topic of the game that is played by the agents. The task in all games is to distinguish the topic from all other objects in the game. This is done using binary discrimination trees that segment the perceived properties of the objects into regions. An object can be discriminated from other objects if the set of properties falls into a set of unique regions of the discrimination trees. In case the properties of another object fall into exactly the same regions of the discrimination trees, both would be indistinguishable. In case there are no unique feature sets for the object, a discrimination tree can be refined so that distinguishing becomes possible.

The discrimination trees are only descended as far as necessary to make a distinction between objects. So the most general region that enables the agent to make a distinction is used. From the two agents that play a game together, one is assigned the role of speaker, while the other one gets the role of hearer. The speaker will start segmenting the image of the scene and tries to create a discriminating feature set for the (randomly chosen) topic. Words are used to describe a feature-set. The speaker of a game must utter a word that describes the features of the topic such that the hearer will be able to discriminate the object that is the topic. In case the speaker does not know a word yet for the set of features that discriminates the topic, it will invent a new word.

At first the hearer will not know the word used by the speaker, because it never heard it before. The game therefore fails, communication was not successful. The speaker will indicate by pointing which object it meant. This is seen by the hearer and it can inspect which feature-set is distinctive for the topic and match the word it just heard to that set of features. Notice that it could be the case that this is not the same set of distinctive features as the set used by the speaker (it might very well be that multiple distinctive feature sets exist). Therefore this model incorporates uncertainty about the meaning of an utterance.

In later games there will be occasions where agents do understand each other because they both know the word to describe the set of features. In case the communication in a game was successful both agents will increase the score of the association between the word and the feature-set. This makes it more likely that the same word will be used again in a future game. However, in case the hearer did know the word, but pointed to the wrong object, both agents will decrease the score of the association. In this case it will be less likely that the same word is used again for this set of features in a future game. Hereby both agents align their lexicon to improve the chances of success in future games.

This forms the base of the guessing game. When played many times, agents will slowly build up a shared lexicon and will be able to understand each other. The language is created through emergent behaviour. A method to evolve syn-

tax in the language in a similar way is also described. Syntax allows the use of multi-word sentences and a syntax that was used successfully will be rewarded. The exact process of evolving syntax will not be described here because it is not relevant for the current work.

More work on the language games is done by Steels (1999) and Steels and Kaplan (1999). The guessing game and other language games are covered extensively. The setting is roughly the same as in Steels (1998), but the scenes are kept somewhat simpler. A scene is a collection of primitive forms on a white-board that can have different colours, shapes, positions and sizes. Also, words used to name objects with multiple features seem not to be used. Only in one game type, the discrimination game, using multiple features is mentioned (Steels, 1999). The goal of the discrimination game is to build the binary discrimination trees. This game is played by a single agent and the game ends successfully when the topic can be discriminated from the other objects by using the discrimination trees. For this game some examples are given in which multiple features are used to distinguish the topic, but this game does not involve creating words for the discrimination trees. Examples of interacting agents that use words for conjunctions of properties were not found in the currently discussed work.

Steels does show that emergent behaviour can explain how a language might have evolved. His work shows some interesting properties like the grounding of words in the environment of agents. However from Steels' work that I have found and that is discussed in the current work, it is not entirely clear what the role of more complex meanings is in the Talking Heads experiment. The interactions between agents in the model that will be described in the next chapter is very similar to the interaction used in Steels' Talking Heads experiment. There are large differences in the type of rules and the way the agents learn and use rules though. In the current work more attention will be given to complex meanings and defeasible rules and exceptions will be used instead of binary trees to determine what concept an object belongs to.

Human language is a popular topic for research and in Kirby (2002) an overview is given about research on human language. According to Kirby human language is one of the most complex problems in science and Artificial Life can contribute to this research because of its ability to combine theory and give predictions by testing these theories with models. Kirby argues that cultural evolution may play an important role for the evolution of language. This is further explored in (Brighton & Kirby, 2001) in which a model is discussed with agents that learn language from agents from a previous generation. The authors use the term Iterated Learning for their model. In this model each generation contains one agent that learns a language from the agent in the previous generation. The instructing agent shows some examples for a small set of random meanings from the language. Because the instructing agent only shows a small subset of the language, the learning agent has never observed the complete language. After learning the agent must pass on the language to the next generation, again by showing a limited set of the meaning/signal pairs. If the agent has to express a signal for a meaning it did not see before, it has to invent and express a new word.

The limitation represents the poverty of the stimulus, learning a language by seeing only a small subset of all possible constructions. In case the language contains regularities that allow the agent to generalize over the language, the

agent might be able to express a signal for the meaning it did not see before. The authors therefore hypothesize that languages that can be generalized can be learnt. Such languages must contain some sort of regularity. A random language cannot be learnt because there is no generalization possible from a subset of the language. The authors argue that compositional languages, in which the parts of a signal have a relation to the parts of the meaning can be learnt. In contrast a random language is the opposite of a compositional language. In such languages the parts of the signals have no relation to the parts of the meaning and therefore a random language cannot be learnt.

In their model, Brighton and Kirby use compression to catch regularities in language. Compression of data exploits regularities found in the data by using a short description for parts of the data that occur often, while using a long description for parts of the data that do not occur often. Therefore, the authors argue, compression is a good hypothesis for the language. If a language can be compressed very well, it must contain a lot of regularity and thus it is a language that can be learnt well. The best hypothesis for a language is the compression that causes the compressed data combined with the description of the data to be as small as possible, thereby catching as much of the regularity in the language as possible.

In an experiment the authors show that by using compression as hypothesis for a language, combined with the poverty of the stimulus, languages are created that can be learnt very well. Thereby the authors conclude that cultural evolution can explain the emergence of syntactic language.

More work on the Iterated Learning Model is done by Kirby (2001). The model is expanded to explain not only regularity in human languages, but also stable irregularity which occurs as well. The original Iterated Learning Model was able to produce a very regular language, far more regular than is observed in real languages (which contain a good chunk of regularity, but also some irregularity). Kirby expands the model with two minor adjustments. The first adjustment is that if a meaning can be expressed by multiple signals, an agent will always choose the shortest description. The second adjustment is a small chance of a character in a signal being mispronounced, meaning the character is skipped. However, Kirby notices that these adjustments are not enough to explain stable irregularity. The changes cause irregularity, but the irregularity is soon removed from the language and replaced by regular signals. Thus there is no stable irregularity. Kirby thereafter makes one final adjustment, he notices that in real language some of the most used words are irregular (for example some of the most used English verbs, which have an irregular past tense). However these irregular words are also very stable. The adjustment that is made is that the meanings will be offered to the agents with a different frequency, to reflect the frequency distribution of words used in human languages. This final adjustment causes a striking result: the most used words now have irregular forms in the created language! There also seems to be a relation between a short signal length and meanings that are often used. In other words, meanings occurring very often, have short signals.

It is important to notice is that the Iterated Learning Model is a simplified model. Meaning and signal can be tied together directly because the agents always match up an entire meaning with the entire signal. Therefore, there cannot be any misunderstanding about the signal and its meaning. It is always clear that the signal refers to all components of the given meaning. More

importantly, the meaning has an order in the model and the signal uses the exact same order. For example the meaning (a_1, b_4) can be encoded as "beaqol" in which "bea" stands for a_1 and "qol" stands for b_4. In language games like those described in (Steels, 1998) there is always uncertainty about which part of the meaning gives reason for a certain signal, for example: when an agent in a language game refers to a red triangle and says "beaqol", the listening agent does not know whether it means: "the object that is a triangle" or "the object that is red". The important thing to notice is that the language in the Iterated Learning Model captures all meaning that is available, something that would be impossible in the real world. There is no way to describe all tiny details of your pet dog when you want to refer to him. Neither are all dogs exactly the same, and thus it would be impossible to catch all meaning in the word "dog" because the meaning of "dog" is slightly different for each dog. The meaning "dog" therefore must catch some of the most important features that each dog has. Suppose your pet dog is a Dalmation, then your dog would be white with black spots. But the meaning "white with black spots" is not covered by the word "dog", however we are still able to recognize Dalmations as dogs.

It is very interesting though that using such a simple principle as compression allows the emergence of a language that can be learnt. This research suggests that we allowed ourselves to be so expressive in our language because the poverty of the stimulus caused the language to emerge in such a way that it became easier to learn.

Another interesting topic is discussed by Skyrms (2004). He suggests a mechanism that could explain the evolution of inference. Skyrms describes a situation in which multiple individuals are located in the same environment. The individuals have developed specific signals for several different approaching dangers. It might be the case that signals also emerged for less specific information, for example if an individual wants to warn for danger but is unsure about the source of the danger. A signal could than be used that means: "a leopard or a snake is approaching". Skyrms calls such signals 'proto-truth functions', a connection of two simple elements which is true in case only one of its base elements is true.

Skyrms now continues and suggests that multiple of these proto-truth functions could emerge in a population. This makes possible the situation in which two agents detect danger at approximately the same time. One of them uses the signal "snake or leopard" while the other uses the signal "not-leopard". On average, agents who take the right actions when hearing these emergency signals have a higher chance to survive when there is danger. Such a mechanism could cause evolving abilities to make use of logical inference of such signals according to Skyrms.

This idea is interesting. The example given by Skyrms incorporates some of the most important features that allow emergent behaviour. First of all there are multiple agents who are interacting with each other. There is also uncertainty because partial information is used and the mechanism is grounded in a shared environment. The example given by Skyrms only is a suggestion though, the mechanism was not implemented and thus not tested.

2.1.2 Increasing skills of agents

A problem that occurs in multi-agent simulations with emergent behaviours is that the skill level of agents can rise to master a certain task, but thereafter agents are not able to improve their skills further because there is nothing for the task that has to be learnt. This limits research on multi-agent simulations to relatively easy to accomplish tasks, while the agents might be able to learn more complex tasks if scaffolding would be used in the learning process. In other words: begin with a simple task and once it is mastered continue with more complex tasks.

Steels proposes a technique that helps increasing the challenge for cognitive agents when they feel confident they are able to cope with a certain task (Steels, 2004). This allows agents to learn simple things first and when they master this, they can move on to learn more complex things. The technique is based on the autotelic principle that states that people can learn very fast if they enjoy doing a certain skill. Because they are enjoying the skill, they have a strong motivation to keep doing it, thereby improving their skills. Examples of such activities are mountain climbing and painting. Steels proposes a method to use this principle to gradually increase challenge levels for cognitive agents. Hereby the agent must be able to monitor the performance of each of its components. When the average performance level of the components gets very high, the task is getting too easy and the agent gets bored. At this point the challenge level will be increased and the agent can start improving its skills. On the other hand, if the performance is low the agent can try to repair, for example by giving one of its components more resources (thereby hopefully being able to master the task). If the performance is low and is not improving, the agent can decide to decrease the skill level again and thereby get the chance to learn an easier task first.

This technique is further explored by Steels and Wellens (2007). Agents that are playing language games improve their skills gradually by raising the challenge as soon as they master a certain task. They start out talking about single objects at first, a task that requires a simple lexicon to be successful. When the agents master this task, they start talking about two objects at the same time. The performance drops and the agents seek ways to increase their skills. Agents are now starting to bind syntactical categories to words and a grammar is developed that allows multi-word utterances with syntactical sentences. Again agents increase their skills and look for even more challenging tasks. Performance drops at first and because no more repair possibilities were given in the experiment the agent is not able to master the new challenge.

This technique gives some useful tools to gradually build up the skills of agents. It allows them to develop some basic skills first and move on to more complex skills that are built upon the basic skills. This behaviour emerges from the interacting agents because they aim at finding a balance between the skill level and their performance.

2.1.3 More complex rules

Research that is aimed at generating a complex set of rules is discussed by Grefenstette, Ramsey, and Schultz (1990) and Grefenstette (1992). A model called SAMUEL, able to generate strategies containing multiple complex rules, is described. This model can cope with delayed feedback in the sense that feedback

on the success of a sequence of used actions is only applied after the sequence is completed. SAMUEL is used in a simulation of the evasive manoeuvres problem. In this simulation an agent controls an air-plane that must avoid a missile that steers towards the air-plane. The agent can steer the air-plane and it uses input from a couple of sensors that for example indicate the range towards the missile or the heading of the missile. Based on the values of its sensors, the agent will perform actions that try to avoid the missile. The pay-off for this task is given when the air-plane escapes the missile or when the air-plane is hit by the missile (in which case a lower pay-off is rewarded).

The system works at two different levels. There is the strategic plan level and there is the rule level. A strategic plan contains multiple rules. At each trial a certain strategic plan is used that will be evaluated. Depending on the sensor readings, a rule from the plan is chosen that matches most sensor values. If multiple rules all match an equal amount of sensor values, one of them must be chosen. In this case the rule with the highest strength is chosen. All rules that match the most sensor values and that result in the same action as the rule with the highest strength, are active. All active rules will be updated when pay-off is rewarded.

Rewarding rules is done by removing a fraction of the strength of the rule while adding the same fraction off the reward to the rule. Using this reward scheme a rule that has a strength that correctly predicts its reward will retain the same strength. A rule overestimating its reward will decrease in strength while a rule that underestimates the reward that follows will increase in strength. The rules in the system therefore emerge to have a strength that correctly predicts the pay-off. This is advantageous for the agent because it can choose the rule that predicts the highest pay-off and if the pay-off is predicted correctly that rule will give the agent the highest chance for survival.

Genetic algorithms are used to create new rules and strategic plans. New rules are introduced by mutating existing rules. At the strategic plan level new plans are generated by crossover of existing plans (combining some rules of two strategic plans into one new plan). The system is able to reach high scores, on average a 90% score was achieved. Even when noise is introduced in the sensor values, the system can still reach an average score of about 80%.

This model allows complex rules that covers multiple sensor readings. In other words: it allows rules about combinations of observed properties. The quality of rules is measured as well and there is a limit on the memory size because tactical plans can only contain a limited amount of rules. Another interesting feature is the rewarding scheme for delayed pay-off that allows agents to select the rule with the highest expected reward. While the model allows rules that are inserted into the system by human users, it can also discover the rules by itself, by letting the agents (the one steering the plane and the missile) play the evasive manoeuvres game and let the rules emerge.

Another method that covers rewarding more complex structures is described by Steels, Trijp, and Wellens (2007). They discuss a method that allows rewarding rules on multiple levels while competing rules on the same level must be punished. An example of this is the competition between synonyms. When a word is successfully used, the synonyms for that word could be punished to evolve a single word for a meaning. In this work agents are able to use words, but also patterns of words. Patterns themselves can also be part of another pattern. If updating on a single level was performed, a word that looses the

competition with a synonym might still be used in a strong pattern of words which can cause the language to loose coherency and structure. The solution proposed is updating words and patterns of words on multiple levels. When a pattern or word is updated, all elements that are part of the pattern must be updated as well. Also each pattern that includes the current pattern (or word) must be updated. This way, when a single word is successfully used, all patterns that include that word are also updated. When a pattern is successfully used, all words in the pattern are updated. This causes coherency between patterns and words and it is shown that this approach works. Updating the elements at different levels at the same time causes no interference between words and patterns.

2.1.4 Emergence of categories

Besides multi-agent research on the emergence of language, there is also work done on the emergence of categories. Often this is combined by some kind of language emergence because agents must somehow be able to refer to these categories.

In the work of Steels and Hanappe (2006) a model is proposed in which agents help human users to search data. Each user owns an agent and the agents mediate when users are trying to exchange data. The agents form categories from examples of data shown to them by their owners. They look for properties in the example data that do not show up in counterexamples (other data form the same set of data that was not given as example). They will request similar data from the same category from agents owned by other users via the Internet for example. In case the agent that received the request for data does not know the category, it asks the requesting agent for some examples. The agent with the request will send the examples that were given to him by its owner and the agent who is trying to provide information makes its own interpretation of the category that fits the given examples. When information is exchanged the user that receives the data can give feedback on whether the data was what he expected, or that he expected something else. Using this feedback, the mediating agents can reward the categories used, align their categories or come up with better fitting categories.

This work has much in common with the language games. It shows uncertainty about the categories because an agent is not directly told by its user what the category is. The agent must infer this information from the examples it was shown. The agents also communicate with each other and have as goal to align their categories so they can help their owners finding the information they requested. By rewarding the agents for successful information transfer, the emergence of categories that work well is stimulated.

A problem that occurs when having to infer a meaning from an example (or a set of examples) is that there is uncertainty about the exact meaning. Often there are multiple possibilities, but it is unknown which one is the correct one. A possible strategy is to keep multiple competing hypotheses in memory and hope that the correct one will win in the end. Another possible strategy is described by Wellens, Loetzsch, and Steels (2008). They propose a method in which words have a weighted connection with possible properties instead of remembering multiple competing hypotheses about the meaning of the word. Each weight represents the certainty that the property belongs to the meaning

of the word. This method works well when objects have many properties. It also allows the emergence of words about combinations of properties. This is much harder when remembering competing word meanings in memory because words about combinations of properties are used less often by definition (they are more specific and thus the possibility to use them occurs less often) and therefore loose the competition.

In the work of Puglisi, Baronchelli, and Loreto (2008) the emergence of colour categories is investigated. Language games are used in a population of 100 agents to develop names for colours the agents perceive. The colours are represented as a value in the range $[0, 1]$. In each game a minimum of two objects is shown to the agents that play the game. These objects have a minimum colour distance in between, this means the agents never have to discriminate between two colours that are very similar. The agent with the role of speaker chooses the word that describes a colour that discriminates the topic from the other objects in the game. A colour category describes a range of perceived colours. The hearer in the game tries to find out which object is the topic by using its own definitions of the colour ranges. Colour ranges are adapted when a game fails, or invented by the speaker if it cannot discriminate the topic. When a game is successful, competing colour categories are forgotten.

The results of this research show that only a couple of linguistic categories are needed to reach communicative success in a real valued domain with an unlimited amount of different values. This work is very similar with Steels his Talking Heads experiment (Steels, 1998, 1999; Steels & Kaplan, 1999). It shows agents playing a game very similar to the guessing game in a shared environment. The categories that are learnt, are quite simple though. Because the categories are mutually exclusive there is no overlap or hierarchy in the categories. As soon as a colour is used successfully, all competing colours are forgotten by the agent, therefore also no complex quality measure is needed.

2.1.5 Using defeasible rules

So far only models using monotonic rules have been discussed. In most models competing rules are possible and some of the models use probabilities to define a likelihood for the conclusions of rules. However none of them uses a defeasible reasoning process to reach a conclusion. As far as I could determine there is no existing work that allows the emergence of defeasible rules and exceptions.

There is some work that tries to capture knowledge using defeasible reasoning though. Especially for the legal domain, there are applications that use defeasible reasoning. Some work exists that tries to find a minimal set of defeasible rules to capture knowledge from the legal domain (Johnston & Governatori, 2003). In this work the authors argue that a minimal set of rules that covers the data contains rules that are more predictive compared to a larger set of ruled covering the same data. Therefore, the minimum rule-set should be the best generalisation of the data. The same argument was also used by Brighton and Kirby (2001) to argue that the best hypothesis for a language was the smallest compression of that language. However in the work of Johnston and Governatori (2003) there is no agent based model that creates a set of rules. Instead a search algorithm is used that expands the rule-set with the rule that increases the accuracy of the rule-set most.

Another example of using defeasible rules to cover knowledge is discussed by

Governatori and Stranieri (2001). This work comes also from the legal domain and it uses association rules to obtain knowledge from a dataset. These rules state how often certain properties occur given that another property was observed, in other words: if you know that property A is true, in what percentage of the cases is property B true as well? Again no agents nor emergent behaviour is involved, instead the data is searched with a brute-force algorithm to obtain the best association rules.

2.2 Relations to the current work

The work that has been discussed in this chapter shows some very interesting properties. First of all the games as proposed by Steels (1998, 1999) and Steels and Kaplan (1999) provide an ingenious way to emerge language that is grounded in the environment. These interactions provide a natural way of evolving words that are based on properties observed in the environment. Steels' model also incorporates the uncertainty about meaning that causes diversity in language. A similar approach will be used in the current work to let agents learn concepts using defeasible rules and exceptions.

In the work of Kirby (2002) and Brighton and Kirby (2001) exploiting generalisation of language was discussed. This work also involved complex meanings because words used in the language were related to multiple properties of objects. The type of language is quite different than the concepts that will be learnt in the current work though because the parts of words used by Brighton and Kirby (2001); Kirby (2002) have meaning too. Words also refer to the entire meaning and therefore there is no uncertainty about which part of an object gives reason for using a certain word. A similarity with the current work is that there is a preference for small hypotheses of the language. This is somewhat similar to using a memory limit for the number of rules, although the used mechanism is different. Both, preferring small languages and using a memory limit for rules, are mechanisms to enforce efficient rules.

Using sequences of complex rules and update them in an appropriate way was discussed by Grefenstette et al. (1990) and Grefenstette (1992). The agents that steered an air-plane to avoid missiles used multiple rules before the outcome of the game they played was known. This will also be the case in the current work. To make decisions in a game agents will use several rules and interactions before the outcome of the game is known. Thereafter the used rules have to be updated appropriately.

Wellens et al. (2008) discussed a memory model was used that allowed learning complex concepts by using weighted connections to properties that possibly were important for the meaning of the concept. This is quite different from the mechanism that will be used in the currently discussed model in which multiple hypotheses for concept rules will be used. In the current work there will be no preference for general or specific concepts and the most specific rule that can be used to determine the concept will be rewarded. This should allow the emergence of complex concepts even when having multiple hypotheses for concepts in memory.

Some work in which defeasible rules were used was discussed as well, although this work involved mining of defeasible rules instead of emerging and learning them. An approach in which defeasible rules emerged from interacting agents

in a population was not found and as far as I am aware this work will be the first attempt to emerge defeasible rules and exceptions that are grounded in the environment. To achieve this the model will incorporate the features that were mentioned in the first chapter and that were discussed in relation to other work in the current chapter. The details of how these features are implemented in the model will be discussed in the next chapter.

Chapter 3

Modelling the conceptualization game

In the first chapter several key ideas and features were presented that are considered to be basic elements for creating an successful multi-agent model in which agents would be able to learn concepts by using defeasible rules and exceptions. The second chapter provided some more insight in existing research that makes use of these features. The current chapter will describe the actual implementation of the model that combines all these features and explains how they will be exploited to obtain concepts based on defeasible rules and exceptions.

The overview of the features is listed again below for reference. First their usage in the model will be shortly discussed. In later sections they will get more attention and they will be explained in more detail. The features that will be implemented by the model are:

1. Defeasible rules and exceptions to describe concepts.
2. Simple general rules and more complex rules for specific concepts.
3. Grounding of concepts in a shared environment.
4. A population of agents that all have a unique interaction histories.
5. Uncertainty of agents about rules, they can only observe the effect of rules.
6. Agents that can measure the quality of rules.
7. A limit on the memory of agents to force creating effective rules.

The first feature is fulfilled because the agents in the population will be learning defeasible rules and exceptions that are based on the rules of the logical system DefLog (Verheij, 2003). The rules and exceptions are used to determine which concepts are valid for objects perceived by the agents. These rules can be general which means that only a few properties have to be observed in an object to conclude that it is an example of the concept. But rules can also be more specific, in which case more properties are needed to be able to use the rule. Hereby the first and the second features from the list are covered.

The interactions between agents will be games about objects in their environment. These games are very similar to the games used by Steels (1999), but

there are some differences. Population sizes of at least three agents will be used and for each game two agents are randomly selected from the population. This causes each agent to have its own unique history of interactions, which increases diversity in the simulation. This covers the third and the fourth features in the list.

Objects used in the games have multiple properties and agents learn rules that match properties to concepts. In the interactions between agents that are performed in the games, only a concept will be communicated and at the end of a game the agents know about which object they were talking. Therefore they cannot observe which rule the other agent used during the game, they can only guess which property or properties in the object gave reason to selecting the concept. Hence the fifth feature in the list, uncertainty about rules, is fulfilled.

Each agent can only remember a limited amount of rules, depending on the size of the agent's memory. Once an agent's memory is full, it can only learn a new rule if an existing rule is forgotten by the agent. To determine which rule must be forgotten, the agent keeps track of the quality of each rule it knows. This is done by remembering the number of times a rule was used and the number of times the rule was successfully used. These two counts together determine the quality of the rule and this gives the agent the ability to decide which rule to forget when more room is necessary. The same measurement also helps the agent deciding which rule to use when it has to find a concept for an object. Hereby the last two features of the list are also covered.

3.1 An overview of the game

The interactions between agents form the base of this model. The interaction is a game called the conceptualization game and it is similar to the guessing game presented by Steels (1999). The most important part of this game is to make sure that after the game was played, the agents that participated in the game have increased their shared beliefs about concepts. If on average in each game the participating agents increase their shared beliefs and if it is assumed that the game does not cause a decrease of shared beliefs with other agents in the population, the shared beliefs about concepts in the entire population should increase over time. This is the base of emergent behaviour and thus the protocol of the game must be shaped in such a way that the agents increase their shared beliefs.

The game is played by two agents, both are randomly selected. One of the agents is assigned the role of speaker, while the other is assigned the role of hearer. The game is played in a scene in which two or more objects are placed that can be seen by both agents. One of the objects in the scene is the topic of the game. The following steps are taken during the game:

1. The speaker is informed which object is the topic and by using its known rules and exceptions it determines which concept describes the topic. If no concept could be found for the topic, the speaker creates a new concept and a rule to match it to the topic.

2. The speaker communicates the concept is has chosen to the hearer.

3. The hearer hears the concept and by using its rules and exceptions it starts looking for an object that is an example of the concept.

4. The hearer will point to the example it has chosen or it will shrug if it could not find an example.

5. The speaker now has to verify whether the example that was chosen by the hearer is indeed an example of the concept. The speaker uses its rules and exceptions to determine whether this is the case. In case the speaker agrees with the example chosen by the hearer, the game ended successful.

6. If the speaker did not agree that the example was an example of the concept or if the hearer shrugged to indicate it could not find an example, the speaker points to the topic of the game. In both cases the game failed.

7. The speaker and the hearer update the rules they used during the game. In case the game failed there is a chance that the hearer will create a new rule.

The step in which the speaker verifies the example differs from the games used by Steels (1999). Steels requires the agents to point to the topic, but in this case that would be too restricting. If only the topic would be considered as a correct answer, agents would be forced to make specific concepts that can be used to distinguish objects in scenes from each other. By allowing agents to point to different examples of the concept, there is less restriction on the concepts that are made.

These games are played repeatedly by agents that were randomly selected from the population. Each agent has its own memory to store rules and exceptions. Only a limited number of rules and exceptions can be remembered by each agent. The objects that are used during the game are chosen from a fixed set of objects and each object has several properties. These properties can be observed by the agents. This allows them to reason about the objects, they can apply rules and exceptions to the objects, which in turn allows them to find concepts for the objects. The details of each of the steps taken during a game are discussed in the following sections of this chapter.

3.2 Initialization

Before any games can be played, the necessary components to play games must be initialized. Because the games will be played by agents, a population of agents must be created from which participants can be selected. The variable n_{agents} defines the size of the population. The agents are created without initial knowledge and they are all identical. Each agent has a unique identifier which is used to track them during the experiments.

Games are played in an environment that contains several objects, therefore a set of objects must be available from which objects can be selected for a game. The objects are specified in an XML document which is read before the experiment starts. Objects are represented by an identifier string, a frequency and a series of at least one property. Properties are also represented by strings. An example of an XML specification of an object is:

```
<object identifier="platypus" frequency="5">
  <property>hasMammaryGlands</property>
  <property>hasSpinalColumn</property>
```

Game 3.1: *An example of part of the output of an experiment, showing the set-up of a single game. In this game* `agent_7` *fulfils the role of speaker, while* `agent_0` *fulfils the role of hearer. It is the 114th game played during this experiment and the scene contains two objects: a blackbird and a dog. For both objects their properties are listed for reference purposes when interpreting the rest of the output of the game. The object that is listed first, the blackbird, is the topic of this game.*

```
game: 114 | speaker: agent_7 | hearer: agent_0 | topic: o1 | scene:
 o1: blackbird = [canFly, canSing, eatsAnimals, eatsPlants, hasBeak,
                  hasBlackColour, hasSpinalColumn, hasWings, laysEggs]
 o2: salmon = [canSwim, eatsAnimals, hasFins, hasGills, hasSpinalColumn,
               inFreshWater, inSeaWater, laysEggs]
[...]
```

```
  <property>hasBeak</property>
  <property>laysEggs</property>
</object>
```

All objects are read from the XML file and put into an object library. Each object must be distinguishable from other objects by their properties. So each object must have a unique set of properties. The frequency is a value that only has a meaning when compared to frequencies of other objects. The relative frequency of an object determines the chance of an object to be seen. If for example object A has a frequency of 6, while object B has a frequency of 2, the chance of seeing object A is three times as high as seeing object B.

For each game two different agents are randomly selected from the population. In this way agents with different backgrounds and therefore different knowledge are matched up and knowledge can spread through the population. The first agent that was selected gets the role of speaker during the game, while the second agent that was selected gets the role of hearer. Because the agents are selected randomly in the first place, this means the roles during the game are random as well.

A set of different objects which are randomly selected from the object library, with their chance of being selected depending on their frequency, form the scene in which a game takes place. The number of objects in a scene can be specified to be a static value, or it can be a random amount between two specified limits. The difference in number of objects causes different chances of being able to recognize at least one object in the scene. With more objects in the scene it is also more likely that ambiguous situations occur. From the objects that are chosen, the first object that was chosen always acts as the topic of the game.

The two agents thereafter start playing the conceptualization game in the scene. This process is repeated over and over again until the maximum number of games (n_{games}) are played after which the simulation is finished. An example of the setup of a game as shown in the output of the model is shown in Game 3.1.

3.3 Ordinary rules and exceptions

The knowledge of agents is a set of defeasible rules that can be applied to objects to determine which concept can be used to describe the object. All rules have

a conjunction of properties in the antecedent and a conclusion in the form of a single concept. Properties and concepts cannot be interchanged and therefore concepts cannot be used in the antecedent of another rule. So far the terms rules and exceptions were used to refer to both types of defeasible rules. However from now on I will use the terms *ordinary rule* and *exception*. The term *rules* refers to both: ordinary rules and exceptions.

Ordinary rules are used to select a concept for a topic, but they can be defeated by exceptions which exclude a certain concept from being derived. An ordinary rule has the following general form:

$$\texttt{p1} \wedge \ldots \wedge \texttt{pn} \rightarrow \texttt{C}$$

In this rule `pi` stands for a property, and C is the concept that is defeasibly derived. If all properties in the antecedent of the rule are found in an object, that object is an example of the concept C mentioned in the consequent of the rule. The antecedent of the rule may only contain properties, and there must be at least one property in the antecedent. A rule may contain an unlimited number of properties in the antecedent. The consequent must always contain a single concept. Note that properties and concepts are strictly separated in this model. Concepts and properties are not interchangeable and therefore a concept cannot occur in the antecedent of a rule.

A rule that represents an exception has the following general form:

$$\times(\texttt{p1} \wedge \ldots \wedge \texttt{pn} \rightarrow \texttt{C})$$

In this exception `pi` are properties like in the ordinary rule. Again only properties are allowed in the antecedent of the rule and the consequent must be a single concept. The meaning of this exception is that if all properties in the antecedent are part of an object, the concept C can not be derived for the object using an ordinary rule that is less specific than the exception. Again, like with the ordinary rules, this conclusion is defeasible.

A difference with the ordinary rules is, that an exception must contain at least two properties in its antecedent, otherwise it cannot be an exception to an existing ordinary rule. An exception containing only one property in the antecedent would be an exception to an ordinary rule with an empty set of properties in the antecedent, which is not allowed. Therefore an exception with only one property is not allowed either.

3.3.1 Applicability of rules

Agents will be applying rules to objects during the game. Not all rules can be used for all objects. Whether a rule a rule can be used for an object depends on the properties of the object. When a rule can be used for an object the rule is said to be applicable to the object. Rules can be used for an object in case all properties in the antecedent of the rule are observed properties of the object. Consider the following examples about an object with properties `hasBeak`, `hasMammaryGlands`, `laysEggs` and `isVenomous`. An example of an ordinary rule that is applicable to this object would be:

$$\texttt{hasBeak} \wedge \texttt{laysEggs} \rightarrow \texttt{Bird}$$

This ordinary rule is applicable because both properties in the antecedent are part of the object. An example of an exception that would be applicable to this object would be:

$$\times(\texttt{hasBeak} \wedge \texttt{laysEggs} \wedge \texttt{hasMammaryGlands} \rightarrow \texttt{Bird})$$

For this exception it is also the case that it is applicable because all properties in the antecedent of the exception are part of the object. An example of a rule that would not be applicable would be:

$$\texttt{hasBeak} \wedge \texttt{eatsAnimals} \rightarrow \texttt{Carnivore}$$

This rule is not applicable because the property `eatsAnimals` is not part of the object.

3.3.2 Defeasibility

Even when a rule is applicable to an object, it might stil be the case that the rule cannot be used if there is an exception that defeats the rule. Therefore rules are said to be defeasible. The main advantage of using defeasible rules is that it allows the creation of general rules for concepts that grasp the most important aspects of that concept. A monotonic rule often cannot be general, because it must be specific enough to exclude counter examples. As general rules are not very restrictive, they are likely to cover a large part of the set of objects that are examples of the concept. This makes defeasible general rules efficient to use. Because they are general they can be used often and therefore they can be learnt faster. A general rule can also be a simple rule that contains only a single property in its antecedent. Simple rules are expected to spread easier through a population because there are only a few possible rules with one property in the antecedent, while the number of possible different rules with two or more properties in the antecedent is much larger. The chances are therefore higher that two agents derive the same simple rule, than the chance that two agents derive the same complex rule.

Consider the following rule:

$$\texttt{hasBeak} \wedge \texttt{laysEggs} \rightarrow \texttt{Bird}$$

This rule covers a large part of the animal kingdom as we know it. We know birds have beaks and lay eggs, so in many cases this rule works very well. On the other hand we can come up with an exception. The platypus is known for its beak, and it lays eggs as well, but the animal is not a bird. To cover this properly an exception to the general rule must be made:

$$\times(\texttt{hasBeak} \wedge \texttt{laysEggs} \wedge \texttt{hasMammaryGlands} \rightarrow \texttt{Bird})$$

So if an animal has a beak, it lays eggs and it has mammary glands, it is no longer considered to be a bird. The first rule is thus defeated by the more specific exception. More specific means that the properties in the antecedent of the more specific rule are a strict superset (the set is larger) of the properties in the antecedent of the less specific rule. To defeat a rule using a more specific rule, the concepts in the consequents of both rules should be the same as well.

So a rule can only be defeated by another rule if the other rule is about the same concept and the set of properties of the other rule is more specific.

If two rules have the exact same set of properties in the antecedent and one of the rules is an ordinary rule, while the other rule is an exception, the exception defeats the ordinary rule. Thus exceptions are stronger than ordinary rules. Ordinary rules can defeat exceptions only if the rule covers the same concept and the set of properties in the antecedent is a strict superset of the properties in the antecedent of the exception.

3.4 Strength of rules

The agents that are playing the game must be able to determine the quality of a rule to increase the chance of choosing a rule with a high likelihood of having success in the game. On the other side, when an agent forgets a rule, the rule that is least successful should be forgotten because that rule is most likely to cause games to fail. To determine the strength of rules the agents keep track of the number of times a rule was used and the number of times a rule was used successfully. Using these counts the strength of a rule is calculated. The strength of a rule is a value between 0 and 1, with the most successful rules having strength 1 and the least successful rules having strength 0.

The strength of a rule is dependent on the ratio between the total number of uses of a rule and the number of successful uses of a rule. This causes rules that are often successful to have a high strength. The strength of rules is also dependent on the absolute difference between the total number of uses and the number of successful uses. This has as effect that rules which are used many times and that consistently cause some games to fail loose strength over time. This means that rules that are not optimal keep loosing strength and the chance they will be forgotten increases over time. A situation like this could be corrected by the agent by inventing a more specific rule that is used instead in the situations where the older general rule fails.

Equation 3.1 shows how the strength of a rule is calculated:

$$S = \left(\frac{s_b + s_n}{a_b + a_n}\right)^{\gamma} \quad \text{with} \quad \gamma = 1 + \alpha\left((a_b + a_n) - (s_b + s_n)\right) \tag{3.1}$$

In this equation, the strength S depends on the accuracy of the rule. That is, the number of times it is successfully applied (s_n), divided by the number of times it was applied in total (a_n). The parameters s_b ($s_b = 1$) and a_b ($a_b = 64$) are used to give a rule its base strength when it is created. Only s_n and a_n change during the experiment. This proportion causes a rule that was mostly used successful to be stronger than a rule that mostly failed when used.

In the exponent the number of times a rule was successfully used is subtracted from the total number of times a rule was used. This part represents the absolute difference between usages and successful usages of the rule. The strength of a rule will decline if this absolute difference in the exponent starts to grow. This way rules that are used often, but do not perform well in all games, will get a penalty in the long run. At some point their strength may have decreased by such a large amount, that the agent forgets the rule and tries to find a better fitting rule that succeeds in more different situations. The parameter α

($\alpha = 0.015$) determines how large the penalty is for rules that consistently fail in some games.

Using this measure for the strength of rules, the rule will always have a strength between 0 and 1 (assuming $0 \leq s_b \leq a_b$, $a_b > 0$ and $\alpha \geq 0$).

3.5 Matching concepts and objects

To communicate about objects using concepts, the speaking agent must be able to find a concept that describes the object that is talked about, while the listening agent must be able to match an object to the concept it heard. To achieve this goal, agents use a set of rules and exceptions they have learnt. The rules and exceptions are aimed at matching properties of objects and concepts to describe the object. In the protocol used in this model, there are several tasks during a game in which object and concept matching is involved. These tasks are:

- The speaker chooses a concept to describe the topic of the game.
- The hearer chooses an object that is an example of the concept mentioned by the speaker.
- The speaker verifies the example of the concept that was selected by the hearer.

For these three tasks object and concept matching is done in the same manner. Which concept is chosen for an object depends on the ordinary rules and exceptions that are known by an agent and their strengths. Depending on the current task of an agent, some specific bounds might be used in the process of selecting a concept.

3.5.1 How concepts are chosen using rules and exceptions

An agent uses the set of known ordinary rules and exceptions to choose a concept for an object. This involves several steps in which the agent will be looking for rules it can use to find a concept and thereafter it starts applying the rules it has found. These steps involve the following actions:

1. Determine which subset of ordinary rules and exceptions must be used to find a concept for an object. When the speaker tries to find a concept for the topic the subset of rules includes all rules the agent knows. In other tasks only rules about the concept used during the game are considered.
2. From the subset of rules that is considered, remove all ordinary rules and exceptions that are not applicable to the object that currently is being looked at.
3. From the rules that are left, remove all rules of the same type for which a more specific counterpart exists. After this step only the most specific ordinary rules that apply to the object and the most specific exceptions that apply to the object are left.

4. Now start applying rules selected from the set of rules that is left to the object that is currently being looked at. This is done by selecting ordinary rules and trying to defeat them with exceptions. This is repeated until an ordinary rule is found that could not be defeated, or until no more ordinary rules are left.

The first step depends on the current task of the agent and is explained later in task specific examples. Once the agent has determined which subset of its known rules to use, it proceeds to find all rules that are applicable to the object. An ordinary rule or exception is applicable to an object if all properties that occur in the antecedent of the rule, are observed in the object. For example, if the following object is considered:

```
cow = [hasSpinalColumn, hasMammaryGlands]
```

And after step 1 the agent determines that the following rules are used to find a concept for the object:

r1. `hasSpinalColumn` $\rightarrow$ `Vertebrate`
r2. `hasSpinalColumn` $\wedge$ `hasMammaryGlands` $\rightarrow$ `Mammal`
r3. $\times$(`hasSpinalColumn` $\wedge$ `hasMammaryGlands` $\rightarrow$ `Bird`)
r4. `hasSpinalColumn` $\wedge$ `hasMammaryGlands` $\wedge$ `hasBeak` $\rightarrow$ `Mammal`

From these rules, rules `r1`, `r2`, `r3` are all applicable to the cow because the properties in the antecedents of these rules are all observed as properties that belong to the cow. Rule `r4` is not applicable, since property `hasBeak` is not a property of the cow. Therefore in step 2 rule `r4` is removed from the set of rules used to determine which concept can be used for the cow.

Only the most specific rules are used when choosing a concept. By only using the most specific rules that apply the possibility to create complex concepts is introduced into the model. The reason for this is that each time the agent has the choice between a general rule and a more specific rule for a certain concept, it will use the more specific one in case it can be applied but the more general one in case the specific rule cannot be applied. This ensures that the more specific rule is used whenever possible. In contrast, suppose that there would be no preference for using the general or the specific rule. In that case by definition the more general rule would be used more often because by definition there are at least as many objects to which the general rule applies as there objects to which the more specific rule applies. The general rule would be used in all cases where the more specific rule does not apply and it would be used in 50% of the cases where the more specific rule also applies. This would give the more general rule an advantage because by definition it would have a higher chance of being used, which in turn allows the more general rule to gain strength faster than the more specific rule. This problem was referred to by Wellens et al. (2008) but it can be solved by always using the most specific rule that is applicable.

Step 3 involves removing all rules for which more specific counterparts are available as well. Consider the following object and applicable rules:

```
platypus = [hasSpinalColumn, hasMammaryGlands, hasBeak, laysEggs]
```

The applicable rules for the platypus:

r1. hasSpinalColumn → Mammal
r2. hasSpinalColumn ∧ hasMammaryGlands → Mammal
r3. hasBeak → Bird
r4. ×(hasBeak ∧ hasMammaryGlands → Bird)
r5. ×(hasBeak ∧ hasSpinalColumn ∧ hasMammaryGlands → Bird)

In this case rule r1 is discarded because rule r2 is an ordinary rule that is more specific. Rule r2 is a most specific ordinary rule and therefore it is not removed. Rule r3 is also a most specific ordinary rule, there are no other ordinary rules with a set of properties in the antecedent that is a superset of the set of properties in rule r3. Rule r4 is removed because r5 is a more specific exception. There are no more specific exceptions than rule r5 which is therefore not removed. This leaves the agent with a set of rules that includes r2, r3 and r5 to determine which concepts are valid for the platypus.

The agent now starts applying rules from this set to the object, to find a concept that can describe the object. Therefore the agent selects randomly an ordinary rule from the set. The chance of a rule being selected depends linearly on the strength of the rule. A rule with a higher strength has a higher chance of being selected than a rule with a lower strength. After an ordinary rule was selected the agent tries to defeat the rule with an exception. Again, if multiple exceptions can be used to defeat the rule, one is chosen randomly with the strength of the exception determining the chance of it being selected. If the ordinary rule was defeated, the agent will try to find another ordinary rule until if finds one that cannot be defeated or until all ordinary rules that apply have been tried by the agent.

This process results in a sequence of rules that were used during the process of trying to find a concept that is valid for the object. This sequence includes ordinary rules that were possibly defeated by exceptions. Note that at most one ordinary rule that is not defeated can occur in the sequence. If there is an undefeated ordinary rule in the sequence, it will always be the last rule in the sequence because the process of finding a rule stops when an undefeated ordinary rule is found. In case the sequence is empty or when it ends with an exception, the agent was not able to find a concept for the object.

The following example shows how rules are selected and used when searching for a concept for an object. Suppose the agent must find a concept for the following object:

```
platypus = [hasSpinalColumn, hasMammaryGlands, hasBeak, laysEggs]
```

The most specific ordinary rules and most specific exceptions that will be used during the concept finding process are:

r2. hasSpinalColumn ∧ hasMammaryGlands → Mammal
r3. hasBeak → Bird
r5. ×(hasBeak ∧ hasSpinalColumn ∧ hasMammaryGlands → Bird)

To find a concept for the platypus the agent has to choose between rule r2 and rule r3 because these are the only available ordinary rules. Suppose rule r3 was chosen. The agent now has to check whether rule r3 can be defeated by an exception. This is indeed the case because rule r5 is a more specific exception for rule r3. So rule r3 is defeated by rule r5. The agent now tries to find another ordinary rule that can be used to choose a concept for the platypus. The only

rule left is `r2` and thus the agent chooses this rule. There is no exception that defeats `r2` and thus the agent has found a concept that is valid for the platypus. The sequence of rules used by the agent to find the concept for the platypus is: `r3`, `r5`, `r2`. The last rule in the sequence is an undefeated ordinary rule and thus the concept that follows from this rule can be used for the platypus. The agent concludes that the platypus is a Mammal.

This finalizes the concept finding process of an agent. In summary, the agent selects the rules that should be considered during the process. Rules that do not apply to the object are removed. Less specific rules are removed and in the end a sequence of alternating ordinary rules and exceptions is created. This sequence represents the rules that the agent tried to use to find a valid concept for the object. Th sequence either ends with an ordinary rule in which case the agent was able to find a concept for the object or the sequence is empty or ends with an exception in which case the agent could not come up with a concept for the object. The next sections will show some more specific examples in which this process is applied during the different steps in the game.

3.5.2 The speaker chooses a concept to describe the topic

The first task during a game in which a concept has to be found for an object is performed by the speaker of the game. The speaker has to find a concept to describe the topic of the game. In Game 3.2 an example is shown in which the speaker uses several rules to select a concept for the topic of the game. During this task, there is a deviation from the general concept finding process. The difference from the general process is found in the part where the agent starts applying rules. The agent will not start by selecting an ordinary rule, instead it starts selecting a concept. This can be any concept that follows from one of the ordinary rules that can be applied to the topic. The chance of selecting a certain concept depends on the strength of the strongest ordinary rule applicable to the topic for each concept.

This selection process is inspired by the work from Grefenstette et al. (1990) in which a similar rule selection process was used. The mechanism ensures that concepts for which many weak rules exists do not have an equally high chance of being selected as a concept for which a single strong rule exists. Therefore concepts for which strong applicable rules exists have a preference over concepts for which only weak rules exist.

After a concept is chosen the speaker sticks to using rules for that concept until it has found an undefeated ordinary rule or until all ordinary rules for the concept were defeated. In case all ordinary rules for a certain concept are defeated the agent will select another concept, again depending on the strongest applicable ordinary rule for each concept.

In Game 3.2 the speaker knows six rules (the rules `s1`, ..., `s6`). The strength of each rule is shown as well in the example. From these rules, there are three ordinary rules that are applicable to the topic. These are rules `s1`, `s3` and `s6`. These rules are all about different concepts. The first thing the speaker does is choosing a concept that it will try to use for the topic. Because for each concept that has an ordinary rule that matches to the topic there is only one rule, these rules determine the chance that the concept will be chosen to be used. Because rule `s3` is the strongest rule, the accompanying concept *Higholk* has the highest chance of being selected. The chance that the speaker will use

Game 3.2: *Example game showing how a speaker uses several rules to find a concept for the topic. Some parts that are irrelevant for the example are left out in this listing, indicated by [...].*

```
game: 300 | speaker: agent_1 | hearer: agent_6 | topic: o1 | scene:
 o1: salmon = [canSwim, eatsAnimals, hasFins, hasGills, hasSpinalColumn,
               inFreshWater, inSeaWater, laysEggs]
 o2: african elephant = [eatsPlants, hasMammaryGlands, hasSpinalColumn,
                         hasTrunk]
rules known by the speaker (sn) and rules known by the hearer (hn):
  s1. canSwim -> Freirtais                                        [0.011]
  s2. hasWings -> Slounbrym                                       [0.013]
  s3. laysEggs -> Higholk                                         [0.035]
  s4. hasMammaryGlands -> Higholk                                 [0.010]
  s5. x(inSeaWater laysEggs -> Higholk)                           [0.000]
  s6. laysEggs -> Vroor                                           [0.000]
[...]
rules matching topic used by the speaker:                  [s3, s5, s1]
the speaker uses rule s1 and says 'Freirtais'
[...]
```

the concept *Freirtais*, following from rule `s1` is somewhat smaller and the chance it will use the concept *Vroor* is virtually 0.

In this case the speaker chooses the concept that has the highest chance to be chosen, it chooses to use the concept *Higholk*. The only ordinary and applicable rule for *Higholk* is rule `s3` and thus this rule is chosen to be used first. The speaker now has to check whether there is an exception that causes rule `s3` not to be a valid reason for the concept *Higholk*. There is indeed such an exception. Rule `s5` is applicable to the topic and is more specific than `s3`, thus rule `r3` is defeated by `s5`. The speaker tries to stick with the concept *Higholk* and starts looking for another ordinary rule about concept *Higholk* that is applicable to the topic. Such a rule is not known by the speaker and thus it has to find another concept for the topic. The only two concepts that are left are *Freirtais* and *Vroor*. By chance the speaker chooses *Freirtais*, again the chance depends on the strongest ordinary rule that applies to the topic.

From all rules about *Freirtais* that are applicable, the speaker chooses rule `s1`. This is not surprising because this is the only ordinary applicable rule for the topic about *Freirtais*. Again the speaker must check whether there are any applicable exceptions that prevent the conclusion from `s1` to be valid. In this case no such exception exists and thus the speaker has found a valid concept for the topic.

In the output from the game, the sequence of rules that is used by the speaker to find the concept for the topic is listed. This sequence consists of rules `s3`, `s5` and `s1`. In the end rule `s1` was used to conclude that the topic can be described by the concept *Freirtais* and thus the speaker utters the word *Freirtais*.

The first turn in the game is now completed. The speaker used its ordinary rules and exceptions to find a concept it could use to describe the topic. The speaker succeeded and used three rules in the process. By uttering the concept the speaker completed its turn and in the next turn the hearer has to find an example of the concept.

Game 3.3: *Example game with four objects in the scene. This example is used to show how the hearer decides which of the objects in the scene are examples of the concept that was used by the speaker. Some parts that are irrelevant for the example are left out in this listing, indicated by `[...]`.*

```
game: 363 | speaker: agent_2 | hearer: agent_4 | topic: o1 | scene:
 o1: evening bat = [canFly, doesHibernate, eatsAnimals, hasMammaryGlands,
                    hasSpinalColumn, hasWings]
 o2: dog = [eatsAnimals, eatsPlants, hasMammaryGlands, hasSpinalColumn]
 o3: blackbird = [canFly, canSing, eatsAnimals, eatsPlants, hasBeak,
                  hasBlackColour, hasSpinalColumn, hasWings, laysEggs]
 o4: cow = [eatsPlants, hasFourStomachs, hasMammaryGlands,
            hasSpinalColumn, isRuminant]
rules known by the speaker (sn) and rules known by the hearer (hn):
[...]
  h1. laysEggs -> Leancrear                                        [0.037]
  h2. isRuminant -> Judseif                                        [0.000]
  h3. hasWings -> Judseif                                          [0.075]
  h4. hasSpinalColumn -> Wrel                                      [0.033]
  h5. x(doesHibernate hasWings -> Judseif)                         [0.000]
  h6. laysEggs -> Krartcil                                         [0.003]
  h7. x(hasSpinalColumn inFreshWater -> Wrel)                      [0.001]
  h8. canSing -> Judseif                                           [0.000]
[...]
the hearer hears 'Judseif'
rules about Judseif matching o1 used by the hearer:              [h3, h5]
rules about Judseif matching o2 used by the hearer:                [none]
rules about Judseif matching o3 used by the hearer:                  [h3]
rules about Judseif matching o4 used by the hearer:                  [h2]
objects the hearer believes are examples of Judseif:             [o3, o4]
the hearer chooses o4 as example of Judseif and points to o4
[...]
```

3.5.3 The hearer searches for an example of the concept

After the speaker selected a concept for the topic, it is the turn of the hearer to find an object in the scene that matches the concept used by the speaker. In Ga e 3.3 an example is shown in which four objects are used in a scene. In this game the concept *Judseif* was used by the speaker to describe the topic.

When selecting an example for the concept, there is also a deviation from the general concept selection process. In this case the hearer is only interested in rules about the concept it just heard. So all rules that are not about the concept used by the speaker are discarded and will not be used when looking for an example object for the concept.

For each object in the scene the hearer determines whether the object is an example of the concept. The hearer selects all rules about the concept that are applicable to the currently considered object. In the example in Game 3.3 for the evening bat these rules are `h3` and `h5`. From these rules the hearer selects an ordinary rule it will try to use to determine whether the object is a possible example of the concept. In this case rule `h3` is the only ordinary rule that applies to the evening bat, so this rule is chosen. Now the hearer has to determine whether there are any exceptions that prevent the conclusion from

Game 3.4: *An example of a game in which the hearer is not able to find an example of the concept used by the speaker. Some parts that are irrelevant for the example are left out in this listing, indicated by* `[...]`.

```
game: 7254 | speaker: agent_0 | hearer: agent_7 | topic: o1 | scene:
 o1: blackbird = [canFly, canSing, eatsAnimals, eatsPlants, hasBeak,
                  hasBlackColour, hasSpinalColumn, hasWings, laysEggs]
 o2: mute swan = [canFly, canSwim, eatsPlants, hasBeak, hasSpinalColumn,
                  hasWebbedFeet, hasWhiteColour, hasWings, laysEggs]
rules known by the speaker (sn) and rules known by the hearer (hn):
[...]
  h1. eatsAnimals -> Freis                                        [0.451]
  h2. hasSpinalColumn -> Steg                                     [0.890]
  h3. x(eatsAnimals eatsPlants -> Freis)                          [0.071]
  h4. expelsInk -> Freis                                          [0.000]
[...]
the hearer hears 'Freis'
rules about Freis matching o1 used by the hearer:               [h1, h3]
rules about Freis matching o2 used by the hearer:                 [none]
objects the hearer believes are examples of Freis:                [none]
the hearer shrugs and the speaker points to o1
[...]
```

rule `h3` to be reached. This is the case, rule `h5` is an exception that is more specific than rule `h3` and thus defeats `h3`. The hearer does not know more rules about *Judseif* that can be applied to the evening bat, so it must conclude that the evening bat is not an example of *Judseif*.

The same process is repeated for the dog, the blackbird and the cow. The hearer does not know any rules about *Judseif* that can be applied to the dog, so it concludes that the dog is not an example of *Judseif* either. For the blackbird there is an ordinary rule about *Judseif* that is applicable. Rule `h3` can be used and is not defeated by any exception that can be applied to the blackbird. So the blackbird is a valid example of *Judseif*. The cow also is a valid example because rule `h2` can be used for the cow and there is not an exception applicable to the cow that defeats `h2`.

So the hearer sees two valid examples of *Judseif* in the scene, the blackbird and the cow. It now has to choose which of these two objects will be the example that it will point at. Each possible example has an equal chance of being chosen, and in this case the hearer chooses the cow to be the selected example. The hearer thereafter points to the cow to indicate that it believes the speaker meant the cow when talking about a *Judseif*.

This ends the turn of the hearer. A choice has been made and now it is the turn of the speaker to verify whether the selected example is indeed a valid example of the concept. Another example is shown in Game 3.4. In this game only two objects are in the scene. For the first object, the blackbird, the hearer knows two rules that can be applied and that is about the concept used by the speaker. However, one of these rules is an exception that defeats the other rule. As the hearer has no rules left about the concept *Freis*, it concludes that the blackbird is not an example of *Freis*. The hearer does not know any rules about *Freis* that can be applied to the mute swan and thus the hearer can not select an example of *Freis*. It therefore shrugs to indicate that it does not know what

Game 3.5: *An example of a game in which the speaker has to verify the example that was chosen by the hearer for the concept Crymkis. In this game each agent can remember ten rules and two objects are shown in the scene. The speaker uses several rules to conclude that the object the hearer pointed to was indeed an example of Crymkis. Some parts that are irrelevant for the example are left out in this listing, indicated by [...].*

```
game: 1172 | speaker: agent_0 | hearer: agent_5 | topic: o1 | scene:
 o1: chimpanzee = [eatsAnimals, eatsPlants, hasMammaryGlands,
                   hasSpinalColumn, livesInGroup]
 o2: mute swan = [canFly, canSwim, eatsPlants, hasBeak, hasSpinalColumn,
                  hasWebbedFeet, hasWhiteColour, hasWings, laysEggs]
rules known by the speaker (sn) and rules known by the hearer (hn):
  s1. hasSpinalColumn -> Crymkis                                    [0.089]
  s2. laysEggs -> Searnstil                                         [0.037]
  s3. eatsAnimals -> Dumhot                                         [0.194]
  s4. x(hasSpinalColumn hasWings -> Crymkis)                        [0.033]
  s5. eatsPlants -> Slefstrean                                      [0.070]
  s6. canSwim laysEggs -> Slefstrean                                [0.004]
  s7. canFly -> Slefstrean                                          [0.010]
  s8. inSeaWater -> Crymkis                                         [0.001]
  s9. hasWings -> Cralkplourp                                       [0.000]
  s10. canFly -> Crymkis                                            [0.001]
[...]
the speaker uses rule s1 and says 'Crymkis'
[...]
the hearer chooses o2 as example of Crymkis and points to o2
rules used by the speaker to verify o2:                    [s1, s4, s10]
the speaker agreed that o2 is an example of Crymkis
[...]
```

the speaker meant, although it had heard about concept *Freis* before.

In this case the turn of the hearer ended and as the hearer was not able to find an example of the concept, the game has failed without the need for the speaker to verify an example.

3.5.4 The speaker verifies the selected example

After the hearer selected an example of the concept, the speaker must verify whether it believes that this example is a valid example of the concept. In case the hearer pointed to the topic the decision is an easy one. As the hearer pointed to the object the speaker had in mind when uttering the concept, the speaker was successful indicating which object it meant. No further action is required by the speaker when this happens.

If the hearer pointed to an object that is not the topic of the game, the speaker must verify whether the selected example is also a valid example of the concept it used for the topic. Again this is done by using a slightly modified version of the general concept matching model. In this case the adaptation means that the speaker will only use rules about the concept used in the game to see if it agrees with the example chosen by the hearer.

In Game 3.5 an example is shown of the verification process. In this game the

chimpanzee is the topic of the game, the speaker used the concept *Crymkis* to describe the chimpanzee. However the hearer did not point to the chimpanzee, but it pointed to the mute swan. Therefore the speaker has to verify whether the mute swan is also an example of *Crymkis*. To do so the speaker selects all rules it knows about the concept *Crymkis* and that are applicable to the mute swan. These rules are `s1`, `s4` and `s10`. Using these rules the speaker now will determine whether the mute swan is a valid example of the concept *Crymkis*.

The speaker can choose between rules `s1` and `s10` since rule `s4` is an exception. Rule `s1` is the strongest rule of the two and thus it has the highest chance of being selected. In this case the speaker selects this rule and if there are no exceptions for this rule, it would mean that the mute swan is indeed an example of the concept *Crymkis*. However rule `s4` is an exception that defeats rule `s1` and therefore the speaker cannot use `s1` to conclude that the hearer was right. The speaker searches through its other rules to see whether there is another ordinary rule that can be used for the mute swan. Rule `s10` is such a rule and because this rule is not defeated, the speaker agrees with the hearer that the mute swan is a valid example of the concept *Crymkis*.

In this case the speaker agreed and thus the game ended successfully. The next step in the game will be updating the ordinary rules and exceptions that were used by the agents during this game.

3.6 Rule updating

After the game was finished both agents start updating their rules. By updating their rules the agents increase or decrease the chance of reusing a rule in a future game. Therefore rules that helped making the game a success should be updated in such a way that they are more likely to be chosen again in a future game. In contrast, rules that caused a game to fail should be updated in such a way that they are less likely to being used again in a future game. Updating rules also changes the chances of a rule being forgotten.

When a rule is updated, the counts that measure how often a rule was used and how often the rule was successfully used are increased. How these counts change the strength of a rule is determined by Equation 3.1, for which the default values $s_b = 1, a_b = 64, \alpha = 0.015$ were used. If a rule was used 100 times and it was successful in 95 games, the strength of this rule would be:

$$\begin{aligned} S &= \left(\frac{s_b + s_n}{a_b + a_n}\right)^{(1+\alpha((a_b+a_n)-(s_b+s_n)))} \\ &= \left(\frac{1+95}{64+100}\right)^{(1+0.015((64+100)-(1+95)))} \\ &\approx 0.339 \end{aligned}$$

When a rule was used with success and the chance of using it again in a future game should be increased, the rule is said to be updated positively. When a rule is updated positively the values a_n and s_n are increased. A rule that led to failure should be updated negatively, in this case the chance of using the rule again in a future game is lowered. When a rule is updated negatively the only value that is increased is a_n.

Suppose that the rule that was used 100 times from which it was successful on 95 occasions, is now updated negatively. The value of a_n now becomes 101, while the value of s_n stays 95. The new strength of the rule becomes:

$$\begin{aligned} S &= \left(\frac{1+95}{64+101}\right)^{(1+0.015((64+101)-(1+95)))} \\ &\approx 0.332 \end{aligned}$$

If the rule on the other hand would be updated positively, the value of a_n would become 101 and the value of s_n would become 96. The new strength of the rule now becomes:

$$\begin{aligned} S &= \left(\frac{1+96}{64+101}\right)^{(1+0.015((64+101)-(1+96)))} \\ &\approx 0.342 \end{aligned}$$

The differences in the strength of a rule do not change by much if it is only updated once. However updating a rule negatively or positively repeatedly has a much larger impact and the chance on using or forgetting a rule will thereby change considerably.

3.6.1 Updating a sequence of used rules

As was shown in section 3.5 agents use sequences of rules to select a concept for the topic, to select an example of a concept, or to verify an example. All rules in these sequences are involved in the success or the failure of a game. Therefore all the rules in a sequence should be updated in a sensible way. A sequence of rules can contain just a single ordinary rule, or it can contain multiple rules, including exceptions. Each rule contributes in its own way to the result of the current game.

To increase shared knowledge in the population, agents should increase the chance they make successful decisions again in future games. The chance they make decisions that led to failure in a game should be lowered in future games. A sequence of rules that was used for an object the agents agreed upon should therefore be rewarded, while a sequence of rules led to disagreement about an object should be punished.

Updating a sequence of rules involves updating ordinary rules and updating exceptions. Ordinary rules are only updated if they were not defeated by exceptions since an ordinary rule that was defeated is effectively not used during a game. Exceptions that occur in a sequence of rules are always updated because exceptions that occur in a sequence of used rules are never defeated. The reason for this is that only the most specific ordinary rules and exceptions that apply to an object are used in the conceptualization process. An exception therefore only occurs in the sequence of used rules if it defeated an ordinary rule that is already maximally specific and thus there is not a more specific ordinary rule that can defeat the exception.

Updating an ordinary rule

Updating an undefeated ordinary rule is straightforward. An undefeated ordinary rule is always the last rule in the sequence of rules because if an ordinary rule is not defeated a decision is made about the object. Therefore the ordinary rule in the sequence of rules for an object directly caused success if the agents agreed upon the object or failure if the agents disagreed about the object. If the sequence of rules ends with an ordinary rule, this rule should be updated positively if the sequence of rules is rewarded while it should be updated negatively if the sequence of rules is punished.

Updating exceptions

For the exceptions in a sequence of rules, the updating is less straightforward. It depends on the concept the exception is about how the exception is updated. Exceptions are updated differently depending on whether the exception was about the same concept the speaker used to describe the topic or not. Notice that only the speaker is able to use exceptions about other concepts when deciding which concept it will use for the topic. When the hearer has to find an example of the used concept, it already knows which concept it should use and therefore it will only use rules about this concept. The same goes for the speaker if it has to verify the example selected by the hearer, in this case the speaker will only use rules about the concept it used to describe the topic. So exceptions about concepts other than the concept used for the topic are only used when deciding how to describe the topic.

Any exceptions about the concept in a sequence of rules that ends with an ordinary rule, counteracts the conclusion (that the concept is valid for the object) of the sequence of rules. Therefore, when a sequence of rules is rewarded because its conclusion was correct, all exceptions about the concept that are in the sequence should be updated negatively. The reason is that these exceptions prevented reaching a conclusion that proved to be valid. An example of a sequence with such an exception would be: `hasBeak` $\rightarrow$ `Bird`, $\times$`(hasBeak` $\wedge$ `hasMammaryGlands` $\rightarrow$ `Bird)`, `laysEggs` $\rightarrow$ `Bird`. In this sequence the first rule was defeated by the second rule, but the concept Bird could still be concluded from the third rule. The exception in the sequence of rules counteracts the conclusion reached by the agent. If the agent was successful by using this sequence of rules it has to update the exception negatively because the first rule proved to be correct after all and should not have been defeated by the exception. On the other hand, if the agent was unsuccessful with this sequence, the exception proved to be correct and should be rewarded.

When the speaker tries to find a concept for the topic it can occur that it tries to use a concept, but that concept is defeated by an exception. In this case a sequence of rules is created that contains an exception that is not about the topic of the game (and this is the only case in which this situation can occur because for the other tasks the agents will only consider rules about the concept used for the topic). A sequence of rules in which such an exception occurs could look like this: `hasBeak` $\rightarrow$ `Bird`, $\times$`(hasBeak` $\wedge$ `hasMammaryGlands` $\rightarrow$ `Bird)`, `hasMammaryGlands` $\rightarrow$ `Mammal`. If the concept *Mammal* proved to be successful the exception helped making the game a success by preventing the concept *Bird* from being used. The exception should therefore be updated positively. If on

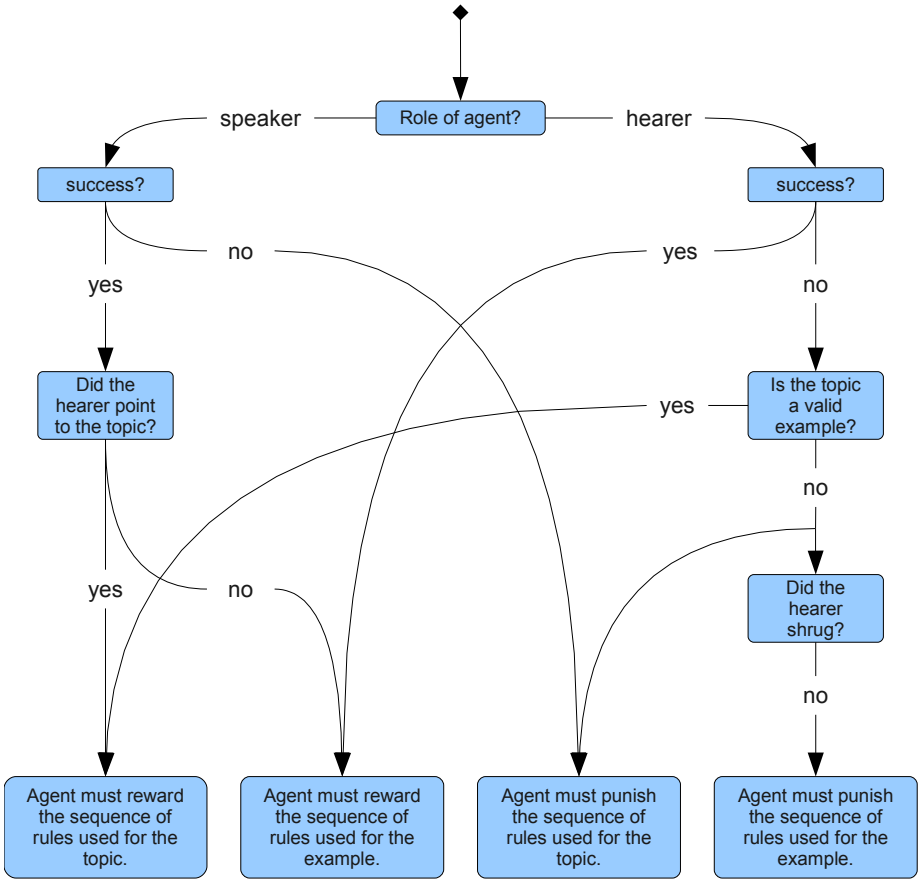

Figure 3.1: *Scheme showing how sequences of rules should be updated.*

the other hand the concept *Mammal* proved to be unsuccessful, the exception should be updated negatively because. In this case it prevented a possibly successful concept (the concept *Bird*) from being used.

3.6.2 Which sequences of rules are updated

The update process is aimed at increasing shared knowledge by increasing the chances of repeating successful decisions and decreasing the chance of repeating decisions that led to failure. Depending on the actions of both agents and the outcome of the game, agents decide which sequences of rules they are updating. Figure 3.1 shows the decision tree for updating the correct sequences of rules.

Sequences of rules should be updated if the agents communicated about the object for which the rules were used. As can be seen in Figure 3.1 the speaker rewards the sequence of rules for the topic only if the hearer actually pointed to the topic. By doing this, both agents agree about the topic and therefore the rules for the topic should be rewarded. If the hearer chooses another example for the concept and the speaker verifies this object as being an example of the concept, both agents will update the sequence of rules for the example instead

of the topic. In this case the agents exchanged information about the example selected by the hearer, but not about the topic. Therefore the speaker does not know what the hearer believes about the topic.

When the game fails the hearer will punish the rules for the example if it did not shrug. Because the speaker did not agree if a game failed and therefore pointed to the topic, the hearer learnt that the example it has chosen was wrong. Therefore the sequence of rules used for the example must be punished. A failing game will cause the speaker to punish the rules for the topic because the concept it used caused confusion (either the hearer pointed to a wrong object, or it did not point at anything at all). It might be the case however that the hearer considered the topic as a possible example of the concept as well (but did choose another object as example). In this case the hearer will reward the sequence of rules it used for the topic because it learnt that it agreed with the speaker about the topic. This increases the hearers chance of selecting the topic in a future game.

Contradicting update of rules

The model for updating sequences of rules has been changed a lot to make sure the emergent behaviour of the agents worked well and the correct rules were rewarded or punished. The heavy changes of the model unfortunately did not only cause improvements in the agents' behaviour. During these changes also a flaw in the rule update process was introduced that caused a contradicting update in the rules. This flaw was discovered after all experiments were performed already and there was no time to repair the model.

The contradicting rule update occurs in a very specific situation. When a game failed and the hearer did not consider the topic as a valid example because it used an exception for the topic, the update of the exception is wrong. According to the scheme in Figure 3.1 the hearer must in this case punish the sequence of rules used for the topic. In this specific case this sequence of rules ends with an exception and the conclusion is that the topic is not an example of the concept. However the implementation of the model assumes that only sequences of rules are updated that end with an ordinary rule and thus with a positive conclusion. In every other situation this is indeed the case, except for this specific situation.

Following the reasoning explained in Section 3.6.1 all exceptions about the concept contradict the result of a sequence of rules. So when the sequence is rewarded, the exception should be negatively updated and vice-versa. However, in this case the sequence of rules ends with the exception and therefore the exception does not contradict the outcome of the sequence, instead the exception determines the outcome of the sequence. The result is that the sequence of rules ending in an exception is punished because the game failed. During this update the exception gets a positive update (because it supposedly contradicted the outcome of the sequence), while it should have received a negative update.

In all other cases sequences of rules that are updated end with ordinary rules and therefore the counter-intuitive update of the exception only occurs in this very specific case. The contradicting update will cause exceptions to be updated positively somewhat more often than they should be. However, as the situation in which this flaw occurs is a very specific one, it presumably only has a minor influence on the behaviour of the population. During the experiments performed

Game 3.6: *Example of rule updating in a game that ended successfully.*

```
game: 2892 | speaker: agent_6 | hearer: agent_2 | topic: o1 | scene:
 o1: blackbird = [canFly, canSing, eatsAnimals, eatsPlants, hasBeak,
                  hasBlackColour, hasSpinalColumn, hasWings, laysEggs]
 o2: dog = [eatsAnimals, eatsPlants, hasMammaryGlands, hasSpinalColumn]
rules known by the speaker (sn) and rules known by the hearer (hn):
  s1. hasSpinalColumn -> Rearp                                    [0.194]
  s2. hasBeak -> Haidceam                                         [0.159]
  s3. eatsAnimals -> Froopsars                                    [0.229]
  s4. x(hasSpinalColumn laysEggs -> Rearp)                        [0.011]
  s5. isRuminant -> Haidceam                                      [0.001]
  s6. canSwim hasSpinalColumn -> Rearp                            [0.000]
  h1. eatsPlants -> Haidceam                                      [0.122]
  h2. hasSpinalColumn -> Froopsars                                [0.170]
  h3. canFly -> Haidceam                                          [0.040]
  h4. eatsAnimals hasSpinalColumn -> Froopsars                    [0.217]
  h5. hasSpinalColumn -> Rearp                                    [0.017]
  h6. x(hasGoldColor hasSpinalColumn -> Rearp)                    [0.000]
rules matching topic used by the speaker:                  [s1, s4, s3]
the speaker uses rule s3 and says 'Froopsars'
the hearer hears 'Froopsars'
rules about Froopsars matching o1 used by the hearer:              [h4]
rules about Froopsars matching o2 used by the hearer:              [h4]
objects the hearer believes are examples of Froopsars:        [o1, o2]
the hearer chooses o1 as example of Froopsars and points to o1
the hearer pointed to the topic
the following rules were updated positively:               [s4, s3, h4]
the following rules were updated negatively:                    [none]
the following rules were forgotten:                             [none]
```

with the model no observations have been made that suggested that exceptions had a suspicious advantage over ordinary rules. Also the high percentage of successful games during simulations ensure that the flaw does not occur often (because one of the prerequisites for this flaw to be triggered is that the game has failed).

3.6.3 Examples of updating rules

In the previous sections all theory about updating rules was covered. The next sections will show some examples of how sequences of rules are updated in different situations.

Updating the rules after a successful game

This is the simplest case. The only rules that have to be updated are those about the object the hearer pointed to. This is the only object the hearer and the speaker exchanged information about.

An example of this scenario is shown in Game 3.6. The hearer pointed to the blackbird which happened to be the topic of the game as well. To determine that the blackbird is an example of *Froopsars* the hearer used rule `h4`. Therefore the hearer updates rule `h4` positively because it caused success during this game.

Game 3.7: *Example of the updating of rules in a game in which the hearer shrugged.*

```
game: 7485 | speaker: agent_5 | hearer: agent_2 | topic: o1 | scene:
 o1: octopus = [canSwim, expelsInk, hasGills, inSeaWater, isVenomous,
                laysEggs]
 o2: blackbird = [canFly, canSing, eatsAnimals, eatsPlants, hasBeak,
                  hasBlackColour, hasSpinalColumn, hasWings, laysEggs]
rules known by the speaker (sn) and rules known by the hearer (hn):
  s1. hasSpinalColumn -> Zour                                    [0.541]
  s2. eatsAnimals -> Croop                                       [0.215]
  s3. canFly hasSpinalColumn -> Zour                             [0.040]
  s4. hasTrunk -> Croop                                          [0.000]
  h1. eatsAnimals -> Zour                                        [0.440]
  h2. hasSpinalColumn -> Croop                                   [0.798]
  h3. hasBeak -> Zour                                            [0.007]
  h4. x(eatsAnimals eatsPlants -> Zour)                          [0.001]
rules matching topic used by the speaker:                          [s5]
rule invented by the speaker to designate the topic:
  s5. expelsInk -> Dup
the speaker uses rule s5 and says 'Dup'
the hearer hears 'Dup'
rules about Dup matching o1 used by the hearer:                   [none]
rules about Dup matching o2 used by the hearer:                   [none]
objects the hearer believes are examples of Dup:                  [none]
the hearer shrugs and the speaker points to o1
the hearer decided to learn a new rule:
  h5. laysEggs -> Dup
the following rules were updated positively:                      [none]
the following rules were updated negatively:                        [s5]
the following rules were forgotten:                             [s4, h4]
```

The speaker used multiple rules to find the concept for the topic. At first rule `s1` was used, but this rule was defeated by `s4`, therefore `s1` is not considered used during this game and does not have to be updated. While it is unsure whether the concept *Rearp* would have been successful in this game, the exception that prevented its use is updated positively nevertheless. The reason is that this exception ensured that the concept *Froopsars* could be used and this concept was successful. The last rule used by the speaker, rule `s3` is also updated positively because it was used with success.

Updating the rules after the hearer shrugged

An example of the scenario in which the hearer could not select an example of the concept and shrugged is shown in Game 3.7. The speaker updates rule `s5` negatively because it was used without success. The hearer shrugged and did not use any rules for the topic. Therefore there are no rules to update for the hearer.

Updating the rules after verification of the example failed

When the speaker could not verify the example chosen by the hearer, the speaker will punish the sequence of rules for the topic because the concept used for the

Game 3.8: *Example of a game in which the speaker did not agree with the hearer when verifying the example selected by the hearer.*

```
game: 2963 | speaker: agent_0 | hearer: agent_5 | topic: o1 | scene:
 o1: dog = [eatsAnimals, eatsPlants, hasMammaryGlands, hasSpinalColumn]
 o2: mute swan = [canFly, canSwim, eatsPlants, hasBeak, hasSpinalColumn,
                  hasWebbedFeet, hasWhiteColour, hasWings, laysEggs]
rules known by the speaker (sn) and rules known by the hearer (hn):
  s1. hasSpinalColumn -> Krul                                    [0.712]
  s2. eatsAnimals -> Bilk                                        [0.338]
  s3. hasFourStomachs -> Bilk                                    [0.000]
  s4. expelsInk -> Vorpclirm                                     [0.000]
  h1. hasSpinalColumn -> Krul                                    [0.617]
  h2. hasWings -> Bilk                                           [0.070]
  h3. eatsAnimals -> Bilk                                        [0.121]
  h4. hasWings -> Krul                                           [0.000]
rules matching topic used by the speaker:                           [s2]
the speaker uses rule s2 and says 'Bilk'
the hearer hears 'Bilk'
rules about Bilk matching o1 used by the hearer:                    [h3]
rules about Bilk matching o2 used by the hearer:                    [h2]
objects the hearer believes are examples of Bilk:                [o1, o2]
the hearer chooses o2 as example of Bilk and points to o2
rules used by the speaker to verify o2:                           [none]
the speaker did not agree that o2 is an example of Bilk
the hearer decided to learn a new rule:
  h5. hasMammaryGlands -> Bilk
the following rules were updated positively:                        [h3]
the following rules were updated negatively:                     [s2, h2]
the following rules were forgotten:                                 [h4]
```

topic caused confusion in the game. In the example in Game 3.8 the speaker therefore updates rule `s2` negatively. The hearer punishes the sequence of rules for the example, in this case that is `h2`. Because the hearer also considered the topic as a valid example of the concept, it rewards the sequence of rules for the topic by updating rule `h3` positively.

3.6.4 Updating synonyms and homonyms

During initial testing with the model a lot of homonyms and synonyms were observed in populations of agents. There was no mechanism that prevented agents to come up with many different successful concepts for the same objects or to make a concept with many different meanings. To decrease the amount of homonyms and synonyms competition between known rules was introduced.

Synonyms in an agent's set of known rules are found by looking for ordinary rules about a different concept than the one used during the game, but with exactly the same antecedent as the rule used for the object the agents agreed upon. In other words: a different concept with the exact same meaning as a concept successfully used during the game. These synonyms are all updated negatively by the agent.

Homonyms are found by looking for ordinary rules about the same concept

Game 3.9: *Example showing updating synonyms and homonyms.*

```
game: 7499 | speaker: agent_1 | hearer: agent_4 | topic: o1 | scene:
 o1: chimpanzee = [eatsAnimals, eatsPlants, hasMammaryGlands,
                   hasSpinalColumn, livesInGroup]
 o2: barn owl = [canFly, eatsAnimals, hasBeak, hasSpinalColumn, hasWings,
                 huntsAtNight, laysEggs]
rules known by the speaker (sn) and rules known by the hearer (hn):
  s1. hasSpinalColumn -> Nudgyd                                    [0.183]
  s2. eatsPlants hasSpinalColumn -> Nudgyd                         [0.044]
  s3. eatsAnimals -> Meilkkreed                                    [0.311]
  s4. hasWings laysEggs -> Mairtjit                                [0.074]
  s5. hasSpinalColumn -> Mairtjit                                  [0.008]
  s6. inSeaWater -> Meilkkreed                                     [0.000]
  h1. eatsAnimals -> Meilkkreed                                    [0.018]
  h2. laysEggs -> Mairtjit                                         [0.079]
  h3. eatsAnimals laysEggs -> Meilkkreed                           [0.213]
  h4. eatsAnimals -> Nudgyd                                        [0.075]
  h5. hasSpinalColumn -> Mairtjit                                  [0.117]
  h6. hasMammaryGlands -> Nudgyd                                   [0.001]
rules matching topic used by the speaker:                             [s3]
the speaker uses rule s3 and says 'Meilkkreed'
the hearer hears 'Meilkkreed'
rules about Meilkkreed matching o1 used by the hearer:                [h1]
rules about Meilkkreed matching o2 used by the hearer:                [h3]
objects the hearer believes are examples of Meilkkreed:           [o1, o2]
the hearer chooses o1 as example of Meilkkreed and points to o1
the hearer pointed to the topic
the following rules were updated positively:                      [s3, h1]
the following rules were updated negatively:                  [s6, h4, h3]
the following rules were forgotten:                                [none]
```

as the one that was used during the game. All these ordinary rules that are not applicable to the topic of the game are considered as homonyms. All homonyms found are updated negatively by the agent.

An example of updating homonyms and synonyms is shown in game 3.9. In this game the concept *Meilkkreed* is used for the topic and the hearer pointed to the topic, causing a successful game. The speaker starts looking for synonyms for `s3` but it does not know other ordinary rules with the same antecedent. Therefore the speaker does not know any synonyms in this case and it does not have to perform updates for synonyms. Thereafter the speaker starts looking for homonyms for the concept *Meilkkreed*. It finds rule `s6` that is also about *Meilkkreed* but cannot be applied to the topic of the game. Rule `s6` is therefore considered a homonym and it is updated negatively. The hearer used rule `h1` for selecting the topic, it therefore looks for synonyms of rule `h1`. It finds rule `h4` that has the same antecedent as rule `h1` and therefore is a synonym. Therefore rule `h4` is updated negatively by the hearer. The hearer also knows a homonym for concept *Meilkkreed*. Rule `h3` about concept *Meilkkreed* cannot be applied to the topic and therefore is a homonym. This rule is updated negatively as well.

3.7 Rule and concept creation

New rules are created in two situations. First of all, if the speaker cannot find a rule to describe the topic, it must create a new rule otherwise the speaker is not able to select a concept for the topic. In this situation the speaker also invents a new concept. The second situation in which new rules can be created is when a game failed, in this case the hearer can create a new rule that would have caused success in the game if it would have been used.

3.7.1 A new concept is created by the speaker

When the speaker cannot find a rule to choose a concept for the topic, it will create a new concept. This is the only way in which new concepts enter the population, so new concepts are only created if a speaker does not know a concept for the topic. If the speaker encounters the object for which it created a new concept again in a future game, it must be able to recognize the object as an example of the generated concept. Therefore the speaker must create a rule that matches the topic to the new concept. This is done by randomly choosing one of the properties of the topic and using that property as antecedent in a new rule about the concept. Consider the following object is the topic:

```
platypus = [hasSpinalColumn, hasBeak, laysEggs, hasMammaryGlands]
```

If the speaker cannot find a concept for the platypus, it must create a new concept. Suppose the speaker randomly selects the property `laysEggs` and the new concept it invented is *Sulk*. The speaker will now create the following new rule:

$$\texttt{laysEggs} \rightarrow \texttt{Sulk}$$

Note that this rule cannot be defeated by an exception because there are no exceptions about the concept *Sulk* because it is a brand new concept (unless by chance the same concept was invented earlier during the experiment, this is possible but the chance this happens is very small). This rule is now added to the set of rules the speaker used in finding a concept for the topic.

The concepts themselves are formed by randomly creating a word with one syllable or two syllables. Each syllable is created by choosing letter combinations from three different lists. One list contains syllable parts used in the first part of a syllable, the second list contains syllable parts used in the middle and the third list contains syllable parts used in the end of a syllable. An example of a newly formed concept from these parts could be:

```
("pr" + "ea" + "k") + ("str" + "a" + "d") = Preakstrad
```

The total number of possible combinations is over 48 million and therefore it is unlikely that the same concept is created twice in one run of the experiment.

3.7.2 The hearer creates a new rule

After a game failed, the hearer might decide to create a new rule. This new rule should explain the decisions made by the speaker during the game. The

speaker makes at least one decision: the decision which concept should be used for the topic. In some cases a second decision is made by the speaker: the decision whether the selected example is a valid example of the concept. If a game failed the speaker will point to the topic, thereby communicating that the object the speaker points to is a valid example of the concept. In case the game failed while the hearer actually did point to an object, the result of the second decision - that the speaker disagrees about the example chosen by the hearer - is made clear to the hearer as well. So when a game fails the hearer knows one or two constraints for a new rule, depending on whether the hearer shrugged or not.

The hearer can choose between two types of rules to create. First it could make a new ordinary rule that is applicable to the topic. Second the hearer could make an exception that is applicable to the selected example and that defeats a rule that could be used for the example (that was invalid according to the speaker). Both options are discussed in the next sections.

Creating an ordinary rule for the topic

A new ordinary rule that explains why the topic of the game is a valid example of the concept can always be created when a game failed since the speaker pointed to the topic in that case. The hearer will use all information it has to create a new rule. A new rule can be a very simple rule that only has one property in the antecedent, or it can be based on an existing rule that is expanded with one extra property. If a new rule is created based on an existing rule, the property that is added must be a property that does not yet occur in the antecedent of the rule. Having two equal properties in the antecedent has no meaning and such rules are therefore not created.

The rules that the hearer will try to expand must be ordinary rules or exceptions that can be applied to the topic of the game and that are about the concept of the game. If an exception is chosen to be expanded, not only a property will be added, but the type is also changed from exception to ordinary rule. The new property that is added to the rule is chosen randomly from the topic. This way the new ordinary rule is guaranteed to be applicable to the topic since the rule on which the new rule was based already was applicable, adding a different property of the topic will not change the applicability of the rule to the topic.

The new ordinary rule must meet some other constraints as well. If the hearer already knows an exception that defeats the new rule when applied to the topic, the new rule is not of much use. Therefore the rules that will be defeated by an already existing exception are not considered by the hearer. The last constraint depends on the information that was available to the hearer during this game. In case the hearer pointed to an object that was considered an invalid example by the speaker, the hearer will incorporate this information as well when creating a new rule. The hearer therefore does not consider new rules that can be applied to the invalid example, unless the hearer knows an exception that defeats the new rule when applied to the example.

This way all information available to the hearer is used when creating a new ordinary rule. The new rule is guaranteed to be applicable to the topic of the game and is not defeated by any known exception. Therefore the hearer will be able to use the new rule with success in a future game for the object that was

the topic in the current game. In case the invalid example is encountered by the hearer in a future game, the new rule cannot be applied to this object or it is defeated by an exception.

Creating an exception for the invalid example

A new exception can also be created by the hearer in case the hearer did not shrug and thus pointed to an invalid example. In this scenario the hearer has all information about the selected example and the topic of the game that is available. It knows that the speaker thinks the topic is a valid example of the concept, but the object chosen by the hearer is not. Therefore a new exception should match this information exactly.

When an exception is made, this is always done by expanding existing rules. The hearer will expand the most specific ordinary rules about the concept that match both the topic and the invalid example. When an exception is made that defeats such a rule when it is used on the invalid example, the original general rule can still be used successfully for the topic but not for the invalid example. This way the hearer has an effective mechanism to use general rules and create exceptions for those objects that should not be considered as examples of the concept. Creating exceptions that defeat rules that only apply to the invalid example - and not to the topic - would not be very effective. In that case the agent would be better of just forgetting the ordinary rule, thereby saving memory space.

Besides being based on an ordinary rule about the concept that matches both the topic and the invalid example, the ordinary rule that forms the base of the new exception also must be the most specific rule that applies to the objects without being defeated. Expanding the most specific ordinary rule makes sure there is no other, more specific rule that is not covered by the new exception or that even defeats the new exception. It might be the case that multiple ordinary rules exist that meet the qualifications, in this case the new exception might not defeat all of these rules. However by creating an exception that is based on the most specific ordinary rule, the hearer makes sure that the new exception defeats as many rules as possible. It also makes sure that a very specific exception is created, one that does not defeat the existing ordinary rules for a large range of objects. Instead it can only be applied in some exceptional cases where a specific set of properties is observed in the object.

Expanding an existing ordinary rule is done by adding one property that occurs in the invalid example but not in the topic and changing the type of the rule to an exception. This way it is guaranteed that the new exception defeats the ordinary rule it was based on when used for the invalid example, but not when used for the topic.

Deciding to learn a new rule

When a game failed, often there are many possible rules that could be created. All possible rules that match the constraints outlined in the previous section are generated by the hearer. In some very specific situations it could be the case that no rules exist that could solve the situation (suppose a maximally specific exception exists for the topic that has all properties of the topic in its antecedent; in this case there is no way the agent can create a rule for the topic

that is not defeated by the exception). However in most situations there will be multiple rules the agent can choose from when learning a new rule.

The hearer does not always decide to learn a new rule though. The agent will decide based on its own experience whether it will learn a new rule. This decision to learn a new rule or not is an important one. It could be the case that the speaker was a badly informed agent who only knew rules the rest of the population would not agree with. There is no way for the hearer to know whether this is the case, but the hearer can have a look at its own rules. Suppose the hearer already knows a very strong rule for the topic, not necessarily about the same concept the speaker used, but about another concept. Such a rule would indicate that a successful concept already exists for the topic and therefore it might not be wise to spend more memory space to learn other concepts for the topic. On the other hand, perhaps the hearer already knows a very strong rule for the concept used during the game, a rule that not necessarily can be applied to the topic. Such a rule would indicate that the speaker used a concept with a well defined meaning in a strange context.

The hearer makes its decision to learn a new rule therefore based on its own beliefs. It looks for the strongest rule that can be applied to the topic to see whether it knows a successful concept for the topic already. It also looks for the strongest rule about the concept, to see whether it is likely that its beliefs about the concept need to be updated. The hearer takes a look at all rules that meet these qualifications and chooses the very strongest rule from this selection. The strength of this rule determines the chance to learn a new rule or not.

Suppose the strongest rule either about the concept or applicable to the topic, that was found by the hearer has strength s. The chance $P(X)$ that a new rule will be learnt is then defined as:

$$P(X) = 1 - s^2$$

A high value for s - meaning a strong rule for the concept or a strong rule for the topic is known by the hearer - results in a low chance that a new rule is learnt. When the value of s is lower, the chances of learning a new rule will increase quadratically. This mechanism makes sure new rules are unlikely to be learnt when the hearer already has a strong opinion about the topic or about the concept that was used. The formula to calculate the chance for learning a new rule is based on a number of initial tests with the model. The formula ensures that when the agent has strong beliefs about the topic or about the concept used by the speaker, there is a very low chance to learn a new rule. This is advantageous because if a new rule is learnt, this often means that an old rule has to be forgotten to make room in the memory of an agent. Each time a rule is forgotten, there is a chance that a strong rule will be forgotten, causing performance of the population to decrease.

Which new rule to learn

If the hearer decided to learn a new rule, it must choose one of the possible explaining ordinary rules or exceptions. In many cases it can be sufficient to learn an ordinary rule. A few different cases can occur when a hearer wants to learn a new rule. These cases are related to the properties of the objects in the game. Set theory shows that there are four different situations that can occur

	properties of topic	*properties of example*	*topic \ example*	*example \ topic*	
1	a	a	-	-	In these cases agents would always agree.
	b	b	-	-	
	ab	ab	-	-	
2	a	b	a	b	Agents used two different rules: $\texttt{a} \rightarrow \texttt{C}$ and $\texttt{b} \rightarrow \texttt{C}$.
	b	a	b	a	
3	a	ab	-	b	Speaker used an exception: $\times(\texttt{a} \wedge \texttt{b} \rightarrow \texttt{C})$.
	b	ab	-	a	
4	ab	a	b	-	Speaker used a more specific rule: $\texttt{a} \wedge \texttt{b} \rightarrow \texttt{C}$, or
	ab	b	a	-	agents know different rules: $\texttt{b} \rightarrow \texttt{C}$ and $\texttt{a} \rightarrow \texttt{C}$.

Table 3.1: *An overview of the differences in properties that can occur when a game failed because the hearer pointed to the wrong object. The differences in properties for the objects can be divided in four cases, each having its own explanation for the decision of the speaker not to agree with the hearer.*

when comparing the sets of properties of the topic and the example selected by the hearer:

1. The topic and the selected example are exactly the same.
2. The topic has properties not observed in the example and the example has properties not observed in the topic (and both objects might or might not share some properties as well).
3. The example has all properties of the topic and more. In other words; the properties of the example form a superset of the properties of the topic.
4. The topic has all properties of the example and more. In other words; the properties of the topic form a superset of the properties of the example.

These situations are also listed in Table 3.1. As can be seen, learning a new exception is only strictly necessary in case 3, where the set of properties of the selected example is a strict super-set of the properties of the topic. In case 3 any ordinary rule that would be applicable to the topic would also apply to the selected example because the selected example has all properties the topic also has. Such a situation can only be explained by the hearer if the speaker used an exception that defeated the rule $\texttt{a} \rightarrow \texttt{C}$ when verifying the example selected by the hearer. And thus the situation in case 3 can only be explained if the speaker knows an exception for the example. However if agents would only create an exception in case 3 exceptions would never be introduced in the population! For case 3 to occur in a game the speaker already must know an exception. Which would be impossible if it could only learn exceptions in case 3. Thus a

	properties of topic	*properties of example*	*topic* \ *example*	*example* \ *topic*	
2	ac	bc	a	b	Agents used two different rules: $\texttt{a} \rightarrow \texttt{C}$ and $\texttt{b} \rightarrow \texttt{C}$ or
	bc	ac	b	a	the speaker used an exception: $\times(\texttt{a} \wedge \texttt{c} \rightarrow \texttt{C})$.

Table 3.2: *Case 2 of table 3.1 expanded with a third property to show why an exception can sometimes explain the disagreement in this case.*

bootstrapping problem would be created if exceptions could only be introduced in case 3: to create an exception there already must exist an exception.

Before case 3 can occur in a game, exceptions must have been created in other cases before. There is another case from Table 3.1 that allows exceptions to be created that can explain the disagreement between the speaker and the hearer. In case 2 - where both objects have at least one property the other object does not have - an exception could be used to explain the failure. To show how an exception can be used in this case, case 2 is expanded with a third property in Table 3.2. Now an exception is a valid explanation for the disagreement in a situation where an exception was not necessarily the cause for the disagreement. This allows exceptions to be introduced into the population.

This example shows that to introduce exceptions into the population the agents cannot rely on only making exceptions in those cases where they are strictly necessary. The agents must have the ability to create exceptions in case 2 as well or exceptions would never be introduced into the system. The method of creating exceptions for the example that was described before, allows exceptions to be made for both case 2 and case 3. Therefore the agents in the population are able to introduce exceptions. Case 2 is also by far the most occurring situation because case 1 would require two objects that are exactly the same (which is not allowed in the model) and case 3 and 4 require the set of properties of one object to be a strict super-set of the set of properties of the other object (which is possible but does not occur often in the set of objects used in the experiments).

When choosing which new rule to learn, the hearer will consider all possible rules it can come up with using the restrictions that were outlined before. From all possible ordinary rules and exceptions one is randomly chosen. The chosen rule will be added to the agent's memory and will be initialized with a default strength (as explained in Section 3.4).

Example of learning a new rule

In Game 3.10 an example is shown of an agent learning a new rule. In this game the hearer pointed to the common buzzard after the speaker said *Sleel*. However, the speaker did not agree that the common buzzard was a *Sleel*. The hearer now considers new ordinary rules that only apply to the african elephant

Game 3.10: *Example showing the hearer learning a new rule after pointing to the wrong object.*

```
game: 7498 | speaker: agent_6 | hearer: agent_5 | topic: o1 | scene:
 o1: african elephant = [eatsPlants, hasMammaryGlands, hasSpinalColumn,
                         hasTrunk]
 o2: common buzzard = [canFly, eatsAnimals, hasBeak, hasSpinalColumn,
                       hasWings, laysEggs]
rules known by the speaker (sn) and rules known by the hearer (hn):
  s1. eatsAnimals -> Kroot                                        [0.735]
  s2. eatsPlants -> Sleel                                         [0.102]
  s3. eatsPlants hasSpinalColumn -> Sleel                         [0.561]
  s4. canSwim -> Sleel                                            [0.000]
  h1. hasSpinalColumn -> Sleel                                    [0.658]
  h2. laysEggs -> Kroot                                           [0.395]
  h3. eatsAnimals -> Kroot                                        [0.014]
  h4. eatsAnimals eatsPlants hasSpinalColumn -> Sleel             [0.001]
rules matching topic used by the speaker:                            [s3]
the speaker uses rule s3 and says 'Sleel'
the hearer hears 'Sleel'
rules about Sleel matching o1 used by the hearer:                    [h1]
rules about Sleel matching o2 used by the hearer:                    [h1]
objects the hearer believes are examples of Sleel:                [o1, o2]
the hearer chooses o2 as example of Sleel and points to o2
rules used by the speaker to verify o2:                            [none]
the speaker did not agree that o2 is an example of Sleel
the hearer decided to learn a new rule:
  h5. hasSpinalColumn hasTrunk -> Sleel
the following rules were updated positively:                         [h1]
the following rules were updated negatively:                      [s3, h1]
the following rules were forgotten:                                  [h4]
```

and exceptions that apply only to the common buzzard. The ordinary rules considered by the hearer are all expansions of rules it knows about *Sleel* already that can be applied to the african elephant. In this case rule `h1` is the only rule that meets this condition. There are many possible expansions of this rule, for example the rule `hasMammaryGlands` $\wedge$ `hasSpinalColumn` $\rightarrow$ `Sleel` would be a valid expansion of rule `h1` because it can be applied to the african elephant, but not to the common buzzard. The hearer also considers new rules with only one property in the antecedent (those can be considered expansions of an empty rule). A valid example of such a rule would be `hasTrunk` $\rightarrow$ `Sleel`. In this case there are no invalid expansions of rule `h1` because the property `hasSpinalColumn` is the only property shared by the african elephant and the common buzzard. So it is not possible to expand rule `h1` with a property of the african elephant so that the new rule would still apply to the common buzzard.

Another option for `agent_5` in Game 3.10 would be to create an exception that excludes the common buzzard but not the african elephant. The most specific ordinary rule `agent_5` knows that can be applied to both objects and is not defeated is rule `h1`. This rule could be expanded with a property of the selected example that does not occur in the topic. There are many properties that would be viable: `canFly`, `eatsAnimals`, `hasBeak`, `hasWings`,

`laysEggs`. All of them would cause rule `h1` to be defeated when used for the common buzzard, but not when used for the african elephant. An example of a valid exception that is considered to be learnt by `agent_5` is therefore $\times$(`hasSpinalColumn` $\wedge$ `hasWings` $\rightarrow$ `Sleel`. This exception would defeat rule `h1` when `agent_5` would try to use it for the common buzzard, but it would not influence the agents decision when it used `h1` for the african elephant.

All possible rules that solve the disagreement between the speaker and the hearer are generated by the agent (this part is not shown). In the given example `agent_5` decides to learn the ordinary rule `hasSpinalColumn` $\wedge$ `hasTrunk` $\rightarrow$ `Sleel`.

3.8 Rule pruning

When agents decide to learn a new rule it often occurs that the agent's memory is full already. When this happens, the agent must create room for a new rule. The memory size is a parameter of the model that can be configured to perform experiments with different memory sizes.

When an agent must remove a rule from memory, the chance of forgetting a rule depends on the inverse of its strength and on chance of forgetting other rules. The inverse strengths of all rules are taken and these values are used as a weight when determining the chance of each rule to be selected for removal. If there are three rules having strengths 0.01, 0.50 and 0.99 their weights would be $0.01^{-1} = 100$, $0.50^{-1} = 2$ and $0.99^{-1} = 1.01$. Their chances of being forgotten would be respectively 97.1%, 1.9% and 0.9%. This makes sure that strong rules have a very low chance of being forgotten when a weak rule is present as well.

When an ordinary rule is removed, all exceptions that defeat the ordinary rule, but do not defeat any other ordinary rules, are removed as well. Exceptions that do not defeat any other ordinary rules known by the agent will never be used and therefore they should be removed.

3.9 Summarizing implementation details

All important features that were mentioned in the previous chapters and at the beginning of the current chapter have been implemented in the model. Ideas from models that were shown to work have been implemented and slightly adapted such as the games from Steels' Talking Heads experiment (Steels, 1999) and the concept selection method similar to the action selection used by Grefenstette et al. (1990). Some important decisions about the exact implementation details were made. For example agents should not only learn exceptions if it is strictly necessary, but to introduce exceptions into the system they should also be considered even when ordinary rules could explain a difference of meanings between agents. Another important design choice is the choice to always use the most specific rules that apply to objects. As explained this allows more specific concepts to be created, which otherwise would have been very hard to accomplish.

The model was completely implemented in the programming language C++ and was used to perform a number of experiments. The details of these experiments are fully covered in the following chapter.

Chapter 4

Experiments

The goal in this work was to find out whether defeasible rules and exceptions could form the base of successful concepts that are grounded in the environment. In chapter 3 the model was presented that will be used to find an answer to this question. A lot of examples were shown that already uncovered some of the behaviour shown by the agents in the population. Also various implementation details were covered and it was explained how these are expected to influence the behaviour of the agents in the model. In this chapter the experiments will be presented that are used to test the performance of the agents in the population and investigate the dynamics of the model. With these experiments it is hoped to find an answer to the question whether defeasible rules and exceptions can form a solid base on which to build concepts grounded in the environment.

4.1 Measurements during experiments

The experiments are aimed at finding answers to questions about the performance of the population, but also provide data for a detailed discussion about the emergence of concepts and rules. Several measurements performed with different settings in the model will be compared. These measurements and the set-up of all experiments are explained in the following sections.

4.1.1 Performance

An important measure is the communicative success of the population while playing the games. Gaining insight into the communicative success allows gaining insight into the effectiveness of the emergent behaviour of the population. To be able to track the communicative success the outcomes of all games that are played will be recorded. The performance is measured as the result of 100 games played by a population.

The performance reached by a population at the end of a simulation is measured by taking the outcomes of the last 25 games of a run. The outcomes over all runs in an experiment are combined and together they form the performance reached by the populations in an experiment. Because each experiment is repeated 40 times, this provides 1000 game outcomes per experiment. By using the last 25 games from each experiment, a sufficient number of successful

Game 4.1: *Example of a game in which the speaker and the hearer do not agree about the topic, but the game succeeded nevertheless. This example is used to show the differences between performance and concept alignment.*

```
game: 7496 | speaker: agent_7 | hearer: agent_1 | topic: o1 | scene:
 o1: black swan = [canFly, canSwim, eatsPlants, hasBeak, hasBlackColour,
                   hasSpinalColumn, hasWebbedFeet, hasWings, laysEggs]
 o2: common buzzard = [canFly, eatsAnimals, hasBeak, hasSpinalColumn,
                       hasWings, laysEggs]
rules known by the speaker (sn) and rules known by the hearer (hn):
  s1. hasSpinalColumn -> Clen                                    [0.618]
  s2. eatsAnimals -> Fourmpreern                                 [0.114]
  s3. laysEggs -> Klun                                           [0.037]
  s4. hasSpinalColumn -> Fourmpreern                             [0.001]
  s5. eatsPlants -> Klun                                         [0.015]
  s6. hasFins -> Clen                                            [0.000]
  h1. eatsAnimals laysEggs -> Fourmpreern                        [0.026]
  h2. eatsAnimals -> Clen                                        [0.368]
  h3. hasBeak -> Klun                                            [0.015]
  h4. hasMammaryGlands -> Klun                                   [0.013]
  h5. eatsAnimals hasBeak -> Klun                                [0.003]
  h6. hasBeak huntsAtNight -> Klun                               [0.000]
rules matching topic used by the speaker:                          [s1]
the speaker uses rule s1 and says 'Clen'
the hearer hears 'Clen'
rules about Clen matching o1 used by the hearer:                  [none]
rules about Clen matching o2 used by the hearer:                    [h2]
objects the hearer believes are examples of Clen:                   [o2]
the hearer chooses o2 as example of Clen and points to o2
rules used by the speaker to verify o2:                             [s1]
the speaker agreed that o2 is an example of Clen
the following rules were updated positively:                    [s1, h2]
the following rules were updated negatively:                    [s4, s6]
the following rules were forgotten:                               [none]
```

games and a sufficient number of games that failed were recorded to be able to compare experiments statistically. This measurement will be compared with the performance reached by populations in other experiments to see whether there is a significant difference between experiments.

4.1.2 Alignment of concepts

Another measure about the performance is the alignment of concepts. This measure is used to calculate the agreement between agents in the population about which concepts can be used for objects. The alignment of concepts therefore is a measure for the alignment of beliefs of the agents. This measure is useful because agents can play a successful game while they do not fully agree about the objects that are used in a game.

The difference between performance and concept alignment is discussed using Game 4.1 as an example. This game ends successfully and therefore the performance of this single game would be 100%. However, the concept alignment of this game shows something quite different. First of all, the hearer does

not believe that the topic of the game is an example of the concept *Clen* and thus the alignment in the beliefs between the agents can never be 100%.

Lets discuss what the concept alignment between `agent_7` and `agent_1` is with respect to the objects shown in the game. The first object considered is the black swan. The concepts that could be used for the black swan according to `agent_7` are *Clen* (by using rule `s1`), *Klun* (by using rule `s3` or `s5`) and *Fourmpreern* (by using rule `s4`). `Agent_1` believes that the concept *Klun* (by using rule `h3`) is valid for the black swan but no other concepts can be used. To calculate the alignment between the two agents for the black swan, the number of shared concepts for the objects is divided by the total number of concepts either of the agents knowns for the object:

$$\begin{aligned} A_{\text{black swan}} &= \frac{|(\textit{Clen}, \textit{Klun}, \textit{Fourmpreern}) \bigcap (\textit{Klun})|}{|(\textit{Clen}, \textit{Klun}, \textit{Fourmpreern}) \bigcup (\textit{Klun})|} \\ &= \frac{|(\textit{Klun})|}{|(\textit{Clen}, \textit{Klun}, \textit{Fourmpreern})|} \\ &\approx 0.333 \end{aligned}$$

For the other object in the game, the common buzzard, `agent_7` believes that the concepts *Clen* (by using rule `s1`), *Klun* (by using rule `s3`) and *Fourmpreern* (by using rule `s2` or `s4`) are valid. `Agent_1` uses the same concepts for the common buzzard (*Clen* follows from `h2`, *Klun* follows from `h3` and `h5`, *Fourmpreern* follows from `h1`). The alignment between these agents over this object therefore is:

$$A_{\text{common buzzard}} = \frac{|(\textit{Clen}, \textit{Klun}, \textit{Fourmpreern})|}{|(\textit{Clen}, \textit{Klun}, \textit{Fourmpreern})|} = 1$$

The total alignment between these agents is the average alignment over all objects that occur in the set of objects used in the experiment. Suppose that the object set only included the black swan and the common buzzard then the alignment between `agent_7` and `agent_1` would be:

$$A_{\text{total}} = \frac{0.333 + 1}{2} \approx 0.666$$

The concept alignment over the entire population is calculated by taking the average of alignments over all possible combinations of two agents from the population. This measure provides a different way of looking at the success that is achieved in the population. Instead of measuring the performance achieved by the agents in the population, the alignment in the emerged concepts is measured and recorded at each time step of a run.

The last 25 concept alignments from each run (making 1000 concept alignments per experiment, because each experiment is repeated 40 times) are used to determine whether there is a statistical significant difference between different experiments.

4.1.3 Creation and lifetime of concepts and rules

For each agent in a population, all information about the creation of rules is recorded during a run. This information consists of the agent that created the

rule, the game in which the rule was created, the game in which the rule was forgotten, the strength of the rule when it was forgotten and the number of times the rule was updated and positively updated during the lifetime of the rule. These values allow tracking several interesting properties. The information will be used to discuss the creation of rules about certain concepts in detail. It will also be used for calculating the lifetime of concepts and gaining insight in which part of runs most concepts are created. Further the average amount of concepts known by a population at the end of a run will be calculated.

The creation time of a concept is determined by the game in which an agent invented the concept and used it for the first time. When all agents in the population forgot all rules they knew about the concept, the concept is considered being forgotten. The number of concepts known by the population at the end of a run is calculated by taking the number of different concepts for which at least one rule exists in the population (so in case there is a single agent who knows one rule about a concept, this concept contributes to the average number of concepts known by the population after a run finished).

The proportion of exceptions, simple ordinary rules and complex ordinary rules known by agents at the end of a run will be recorded as well. Combined with the average strengths, lifetimes and number of uses of these rules, the effectiveness of exceptions and complex rules can be discussed. A complex ordinary rule is considered as an ordinary rule with at least two properties in the antecedent of the rule.

4.1.4 Detailed information about games

Besides the performance, the concept alignment and the various values about rules, also the output of each game played by the agents will be recorded. Examples of the output of games were given already in the previous chapter and Game 4.1 in the current chapter shows an example of the output of a game. In the next chapter in which the results of the experiments will be discussed, a number of games will be discussed in detail to illustrate certain effects that occur during the experiments.

4.2 Set-up of the experiments

Various experiments will be performed with the model. Three of these experiments are used to study the effect of parameter settings of the model, while the fourth experiment is aimed at gaining insight in the development of the population in the long run. The parameters that are used during the experiments are listed in Table 4.1.

4.2.1 Objects used in the experiments

In all experiments the same set of objects was used that can be found in Appendix A. There are 16 objects in this set and they were chosen to include some objects that we consider as exceptions for various reasons (like the platypus because it shares properties of birds and mammals, the black swan because it has a different colour than the mute swan and occurs less often and the penguin because it is a bird that cannot fly). While testing other object sets would

parameter setting	default value
agents in a population	8
maximum rules per agent	8
objects in a scene	2
games played in a run	7500
different objects in the object set	16
runs per experiment	40

Table 4.1: *Settings used during all experiments unless explicitly mentioned otherwise.*

have been very interesting and almost certainly would have provided important insights about the model, no other object sets were tested because there was no time to focus on testing more different scenario's. The data that was gathered by performing the experiments with this single set of objects is not even fully explored in this work.

Important to notice is that the agents could be successful by learning only two rules and concepts when using this object set. The reason is that almost all objects in the set share the property `hasSpinalColumn`. A concept with the meaning `hasSpinalColumn` would therefore be very successful. The only object that does not have this property is the octopus for which a different concept would have to be created to cover all objects that are used.

4.2.2 The effect of memory size

To study the effect of memory size of agents, the number of rules each agent can remember is varied in a series of experiments. The number of rules each agent can remember in this series of experiments is 4, 6, 8, 10, 12, 16 or 20 rules. For this experiment all other parameter values use the default values listed in Table 4.1. In the rest of this work these experiments will be referred to as the *7.5k-n-rules* experiments.

4.2.3 The effect of population size

The effect the population size has on the emergence of rules is studied by performing several different experiments in which the population size varies. The population sizes used during these experiments are 4, 6, 8, 10, 12, 16 or 20 agents. These experiments are called the *7.5k-n-agents* experiments and the default settings from Table 4.1 are used.

4.2.4 The effect of scene size

The effect of the size of the scenes in which the agents play their games will be investigated as well by a series of experiments in which the scene size is varied. The scene sizes that are tested are 2, 3, 4 and 5 objects per scene. These experiments are referred to as the *7.5k-n-objects* experiments. All default parameter values from Table 4.1 are used in these experiments as well.

Besides using a fixed amount of objects in the scene, an experiment with a random number of objects per scene will be performed as well. In this case each game will have 2, 3, 4 or 5 objects in the scene and the number of objects is

randomly chosen for each game. This experiment also uses all default parameter values from Table 4.1 and the experiment is called the *7.5k-random-objects* experiment.

4.2.5 Long term effects

The last experiment is used to see what happens if the agents are allowed to play more than 7,500 games. Therefore during this experiment the agents will play 250,000 instead of 7,500 games. In this case a number of different memory sizes for agents are tested as well. The memory sizes that are tested in this experiment are 3, 4 and 6 rules. The experiment is referred to as the *250k-n-rules* experiment. And like all other experiments all other parameters use the default values listed in Table 4.1. Only small memory sizes are tested because larger memory sizes require more computing power and the experiment would take a lot longer when using large memory sizes for the agents[1].

These experiments should provide a lot of detailed information about the behaviour of the model and the emergence of concepts by using defeasible rules and exceptions. In the next chapter the results from all experiments will be discussed by looking at the performance measurements and the formation of concepts and rules. Also a number of runs will be looked at in more detail to see why certain interesting observations occurred during the experiments.

[1] All experiments were performed on a 2.4Ghz quad-core AMD machine with 4GB of memory. The model is implemented in such a way that it can utilize all processor cores of the cpu and the 40 runs of the experiment with 250,000 games took about 30 minutes to complete. The data that is recorded during the experiments is compressed to safe disk space. The total amount of data generated by the *250k-6-rules* experiment is almost 1GB (in compressed form). In comparison, the combined data generated by all experiments in which 7,500 games are played is slightly more than 800MB (in compressed form).

Chapter 5

Results

In the previous chapter the experiments were described that were performed to look at the effects of several parameter changes and to investigate the dynamics of the model. In the current chapter the results from these experiments will be discussed. The first part of the chapter will cover measurements taken from the experiments backed with graphs and statistical tests. In the second part of this chapter some runs will be examined more closely by looking at the concepts that are created and the rules that are used by the agents. Therefore the first part of the chapter will mainly be about the performance achieved by the population, while the second part of the chapter will cover the emergence of rules and concepts.

5.1 Average game outcomes

To measure the ability of the agents to invent a set of concepts that allow them to communicate successfully with other members of the population, the average outcomes of games is shown in Figures 5.1, 5.2 and 5.3. The average outcome of the games shows how often the agents were able to finish a game successfully and thus, how successful they were in communicating during the games.

Figure 5.1 shows the difference between memory sizes for agents. It seems that a larger memory size has a positive effect on the outcomes of the games. Larger memory sizes are performing slightly better. This can be explained because when agents can remember more rules, there will be a higher chance they are sharing some rules that allows them to successfully communicate in a game. In the experiment with 20 rules, the final performance (measured over the last 25 games) was significantly better than the performance in the experiments with 4 rules ($p = .007$), 8 rules ($p = .027$), 10 rules ($p = .017$) and 12 rules ($p = .041$). In the experiment with 16 rules the outcomes were significantly better than the outcomes in the experiments with 4 rules ($p = .004$), 8 rules ($p = .016$), 10 rules ($p = .010$) and 12 rules ($p = .025$). In the experiment with 6 rules agents performed significantly better than in the experiment with 4 rules ($p = .021$). All other comparisons between runs of different memory sizes were not significant. This suggest that a large difference in the number of rules can indeed result in higher performance.

In Figure 5.2 it can be seen that smaller populations seem to be perform-

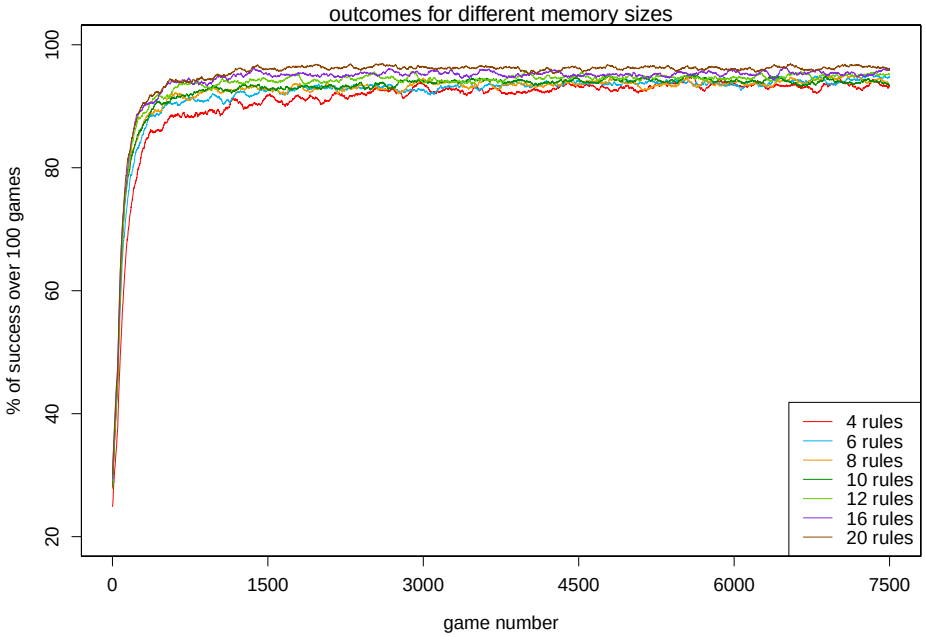

Figure 5.1: *The successrate of agents measured over 100 games for different memory sizes. The graph shows the average over 40 runs of the 7.5k-n-rules experiment.*

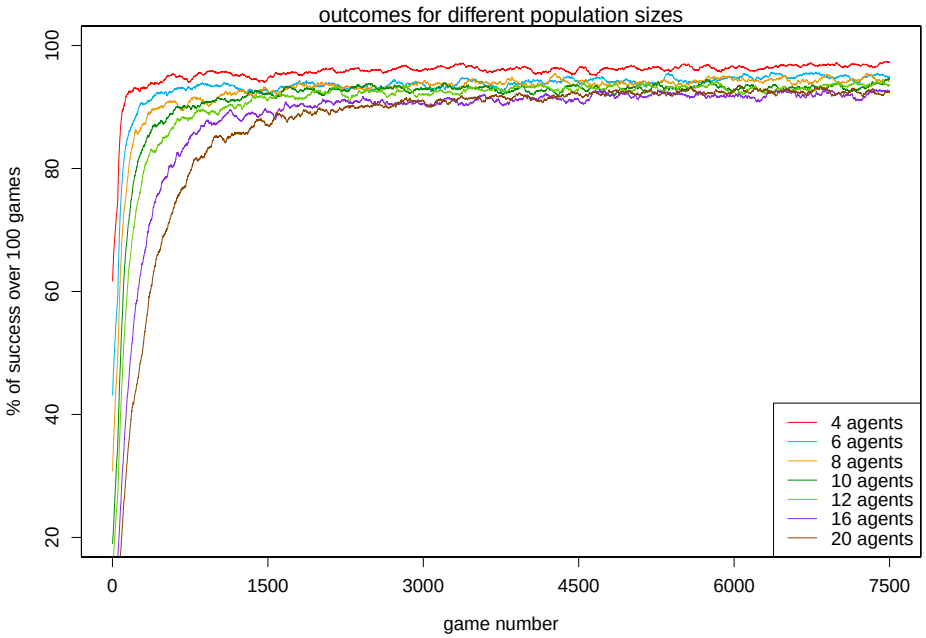

Figure 5.2: *The successrate of agents measured over 100 games for different population sizes. The graph shows the average over 40 runs of the 7.5k-n-agents experiments.*

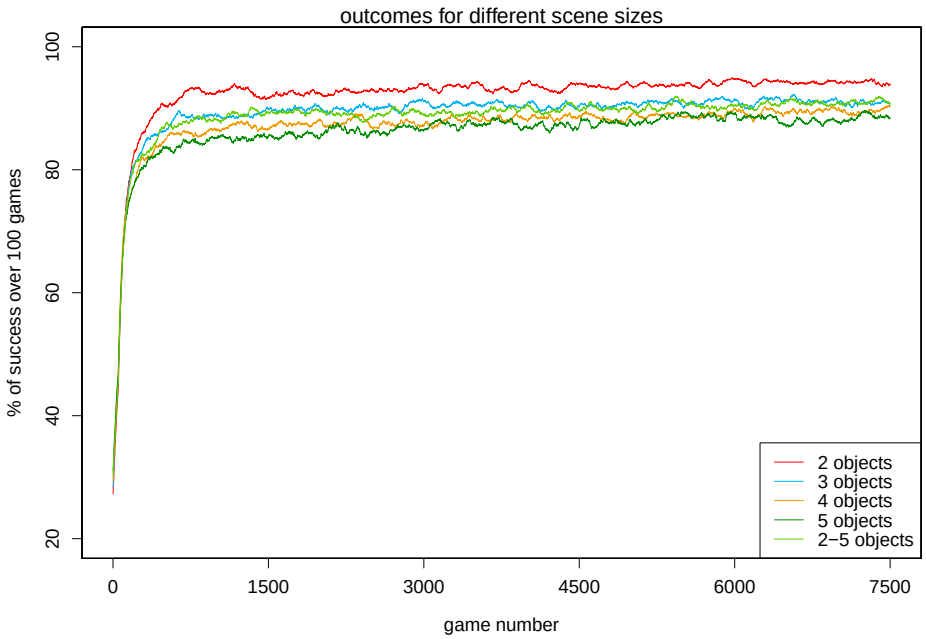

Figure 5.3: *The successrate of agents measured over 100 games for different scene sizes. The graph shows the average over 40 runs of the 7.5k-n-objects and the 7.5k-random-objects experiments.*

ing better than larger populations. The smaller populations also are able to reach the final performance levels earlier than the larger populations. This can be explained because it takes more time to spread knowledge through a larger population. In larger populations there are more agents who initially do not know any words for any objects, therefore the population will create more different concepts at the first stages of the simulation. Hereby more competition between concepts will occur, and thus it will take longer to reach a high rate of success. Larger populations do not reach high performance as fast and the performance reached is a bit lower, but they are still fairly successful playing the conceptualization games. The final performance (measured over the last 25 games) of the experiment with 4 agents is significantly better than the performance of the experiments with 20 agents ($p < .001$), 16 agents ($p < .001$), 12 agents ($p < .001$), 10 agents ($p < .001$), 8 agents ($p < .001$) and 6 agents ($p < .001$). The performance of the experiment with 10 agents was significantly better than the performance of the experiment with 20 agents ($p = .028$) and 16 agents ($p = .049$). None of the other comparisons were significant. This suggests that having 4 agents results in high performance but that less agents does not always mean a higher performance. Contrary to what was expected to see, larger populations were not able to exploit their greater ability to explore more rules. While having a larger population means that more different rules will be created, this does not necessarily mean that more effective rules are able to emerge in the population.

Figure 5.3 shows that a larger number of objects causes the performance to degrade. In the initial stages of the simulation the outcomes are virtually equal,

but the differences are showing up when the population has reached its final performance level. The reason for this observation is not obvious. One might think that there is a higher chance of finding a fitting example for a concept if the scene contains more objects and thus one would expect a higher success rate. On the other hand, if a scene contains only two objects, the chances are much higher that the hearer will pick the topic of the game, resulting in immediate success of the game. If a scene contains a lot of objects, the chances are higher the hearer picks an example that is not the topic. If this happens, the speaker must confirm the example and could thereby disagree, in which case the game fails. Thus, having lots of objects in the scene causes a higher need for verification of the chosen example, which in turn could lead to a lower success rate.

Another factor that might explain the results is that having more objects in a scene just puts less pressure on agents to repair faulty rules. If more objects are available to choose from, it is easier to find an object both agents agree about. Therefore the need to repair rules occurs less often and this in turn could lead to a lesser degree of increasing shared knowledge between agents.

The final performance (measured over the last 25 runs) of the experiment with 2 objects was significantly better than the performance of the experiments with 3 objects ($p < .001$), 4 objects ($p < .001$) and 5 objects ($p < .001$). The other comparisons between runs with different object sizes were not significant. This suggests that 2 objects in a scene was an optimal case that resulted in high performance.

5.2 Concept alignment measurements

The alignment between concepts for objects is measured throughout all runs of the simulation. In Figures 5.4, 5.5 and 5.6 can be seen how the different memory sizes, the difference in population size and the number of objects shown to the agents in a game influences the alignment of the concepts.

By looking at Figure 5.4 it seems that memory size influences the speed with which concept alignment increases. In populations with agents that can remember more rules, the concept alignment increases slower. The concept alignment reached at the end of the simulation is very similar for all memory sizes though. In Figure 5.5 it is observed that population size has a large impact on the speed with which the concept alignment increases, but also on the final concept alignment reached by the population. Figure 5.6 shows that the sizes of the scenes in which the agents play the games cause a difference in the final concept alignment. Larger scenes cause a lower concept alignment in the population. However for all scene sizes the concept alignment shows very similar increases in the beginnings of the runs.

The differences in concept alignment for different memory sizes seem small at the end of the runs. However, measured over the last 25 runs of the experiment, significant differences were found between the run with 4 rules and all other runs ($p < .001$ in all cases). Also differences were found in the comparisons between the runs with 8 rules and 10 rules ($p = .002$), 8 rules and 12 rules ($p = .003$), 8 rules and 16 rules ($p = .012$) and 8 and 20 rules ($p < .001$). A clear conclusion about this result cannot be drawn, but it seems that agents with 8 rules perform very well, while agents with only 4 rules perform significantly worse. Almost all

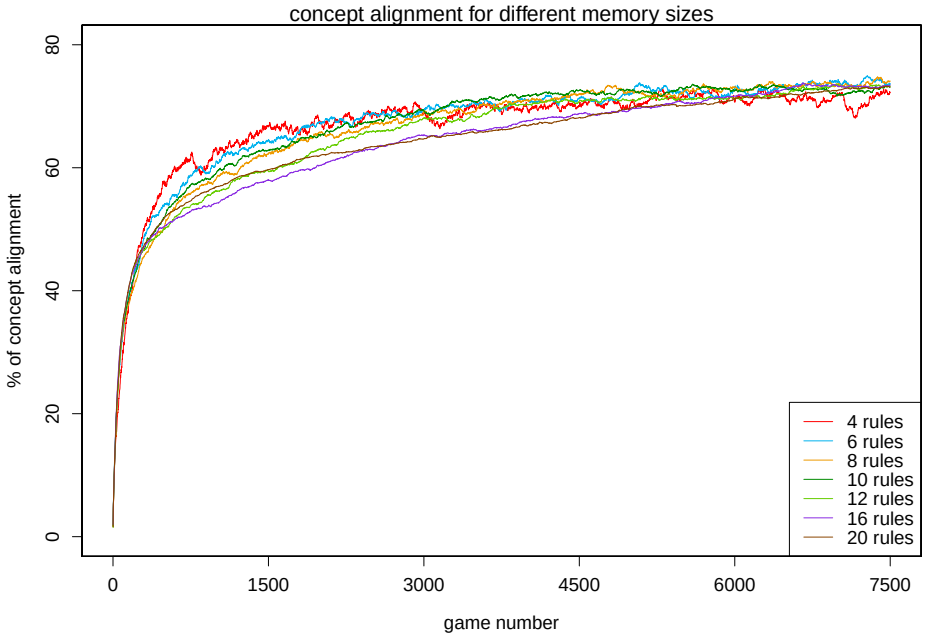

Figure 5.4: *The concept alignment in the population for different memory sizes of agents. The resuls are averaged over 40 runs of the 7.5k-n-rules experiments.*

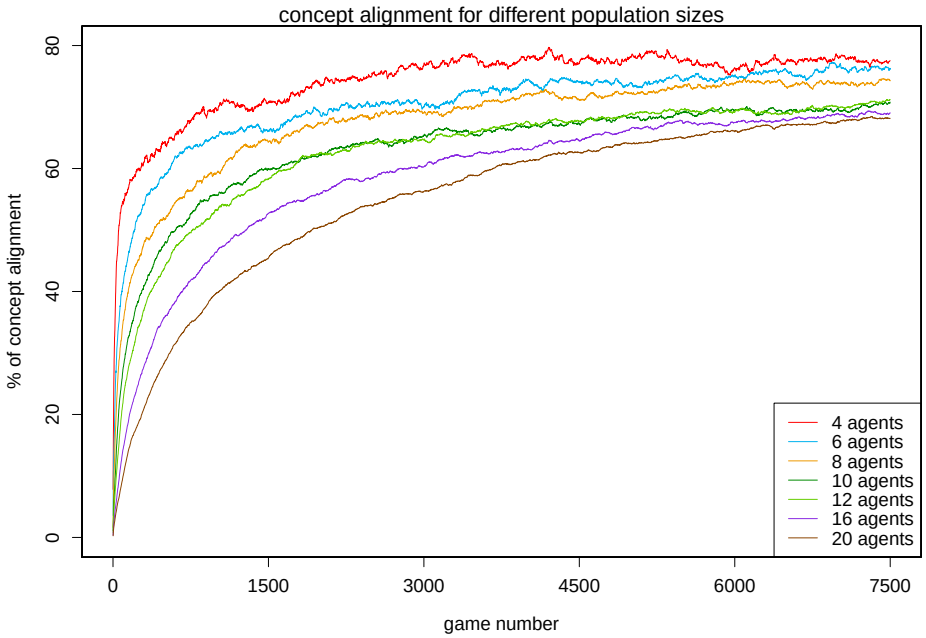

Figure 5.5: *The concept alignment in the population for different population sizes. The results are averaged over 40 runs of the 7.5k-n-agents experiments.*

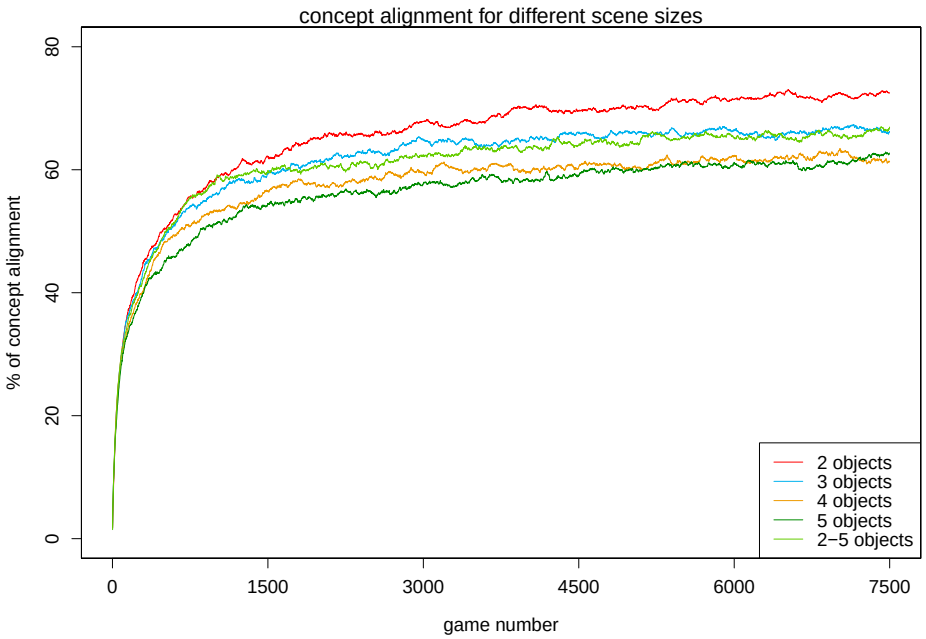

Figure 5.6: *The concept alignment in the population for different scene sizes. The results are averaged over 40 runs of the 7.5k-n-objects and the 7.5k-random-objects experiments.*

comparisons for the differences in the population size, measured over the last 25 concept alignments from each run, were significant. The population with 20 agents had a significantly lower concept alignment than the population with 16 agents ($p = .003$). The difference between the population with 12 agents and the population with 10 agents was not significant. All other comparisons showed that in populations with more agents the concept alignment was significantly lower ($p < .001$ in all cases). The concept alignment of the last 25 games of the runs showed no significant difference between the experiments with 5 objects and 4 objects in the scene. For all other comparisons between scene sizes the concept alignment for the larger scenes was significantly lower than the concept alignment for the smaller scenes ($p < .001$ in all cases).

The increase in time it takes to develop a high concept alignment in the case where agents are allowed to remember many rules, could be explained because more rules allow more concepts and also more complex concepts. A concept has to be used a certain amount of times to stabilize. This takes longer for complex concepts or for concepts that cannot be used that often because they only apply to objects not frequently observed. Thus having the ability to use more rules, and therefore also more concepts, will cause the concept alignment to increase slower. Agents that can remember more rules also causes rules that do not work well are not forgotten as quickly as opposed to the situation in which agents can remember less rules. The unsuccessful rules remain in the agents' memories and cause a lower concept alignment (while they are not used often and thus do not contribute much to the performance of the agents).

The lower alignment that results from a larger population might be consid-

#rules	#runs	100% alignment achieved in	average after 250k games
3	40	1 run(s)	84%
4	40	0 run(s)	86%
6	40	5 run(s)	88%

Table 5.1: *The number of runs in which the concept alignment reached 100% during the simulation and the average concept alignment reached at the point the simulation finished. These results are from the 250k-n-rules experiments.*

ered obvious: it can be explained because each member of the population has to agree with each other member of the population. If the population is larger, this is a harder goal to achieve. More agents means that the chances are lower that two particular agents meet each other, so they will not be able to increase their shared knowledge that often. This will cause slower growth of the alignment of concepts through the population. The lower alignment with increasing population size also clearly shows up in the average game outcomes in Figure 5.2. A lower success rate in games depends largely on the concept alignment.

Looking at Figure 5.6, the concept alignment resulting from a differing number of objects in the scenes is shown. This graph shows a similar effect as observed in Figure 5.3: more objects in the scene causes a lower performance. The same reasons as used with the success rate of games, can be used to try to explain the lower concept alignment. A higher number of objects in the scene means a lower chance that the hearer will actually choose the topic of the game. This means the speaker more often has to verify the chosen example and this could cause more failures and thereby a lower concept alignment as well.

When comparing the graphs of the concept alignment and the game outcomes, it can be seen that both follow very similar curves. Though performance achieved in games is higher than the average concept alignment. This can be explained because the concept alignment is a very strict measurement. For the concept alignment all objects and concepts are compared, also objects that are not often observed while games are being played. The performance reached by the agents when playing games is a measure that is not as strict because it is measured only over a small part of the data and agents do not have to agree about every object they observe. This was also explained in Section 4.1.2 and as expected the concept alignment is lower than the performance reached by the agents.

5.2.1 Long term effects for the concept alignment

In the *250k-3-rules*, *250k-4-rules* and the *250k-6-rules* experiments, the agents played 250,000 games to see whether the concept alignment in the population would converge to 100%. The results are shown in Figure 5.7 and Table 5.1. As can be seen there is still progress after the first 7,500 games. But progress is made a lot slower than in the first couple of thousand games. After 250,000 games there is still a notable difference between the concept alignment for the different number of rules. The agents that can remember more rules are performing slightly better. Table 5.1 shows that 100% alignment is not reached often but in some occasions the populations are able to achieve it! Populations with agents that can remember more rules seem to have a higher chance to reach 100% concept alignment.

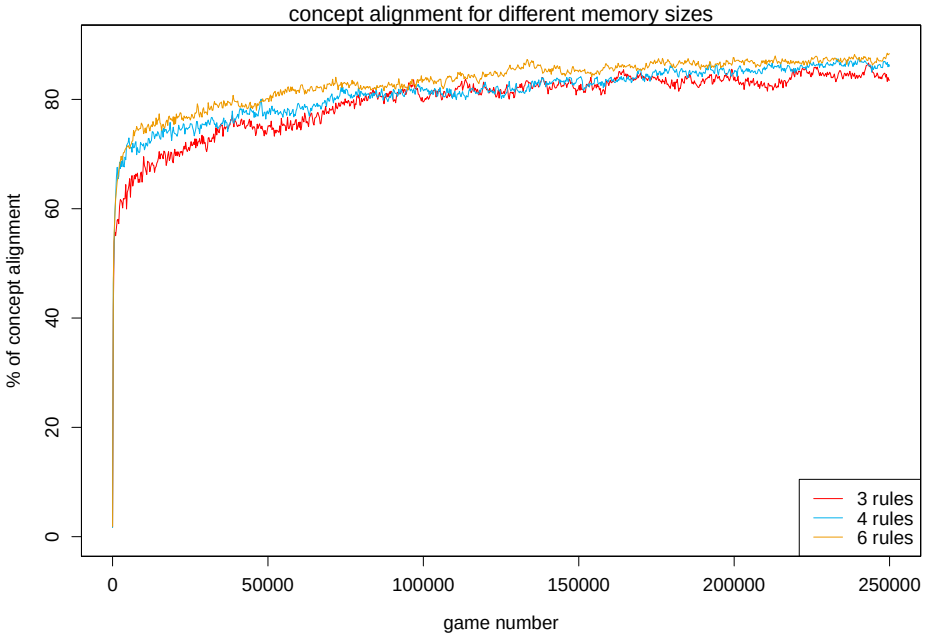

Figure 5.7: *The concept alignment in the population for different memory sizes of agents averaged over 40 runs in the 250k-n-rules experiments.*

5.3 When are concepts created?

In the beginning of an experiment, the agents do not know any rules about the objects in their environment. Therefore many new concepts are created in the first couple of games. Figure 5.8 shows the proportion of the total number of concepts invented at each point during a run. The first part of the graph is steep, within 60,000 games 50% of the concepts is invented already. This indicates that agents invent more concepts as long as a stable set of concepts is not yet created. After the agents have played many games, they will know a concept for most objects they encounter. Only in case an agent forgets a rule the it can occur that it does not know a concept for an object. Thus, as expected, most concepts are invented in the beginning of a run.

5.4 How many concepts are created?

What are the effects of the memory size of agents on the creation of concepts? Do agents remember more concepts if they are allowed to remember more rules? In Table 5.2 different measurements for concepts in the *7.5k-n-rules* experiments are shown. Striking is that more concepts are created during a run when the number of rules that can be remembered is lower. When agents can only remember a couple of rules, often the situation occurs in which the agent has no fitting rule for the scene. At that point the agent must invent a new concept. There will also be more concepts that are thrown away due to the heavy competition between rules. If an agent invents a new concept, the rule associated with it will be very weak at the beginning. The chances are very high this new

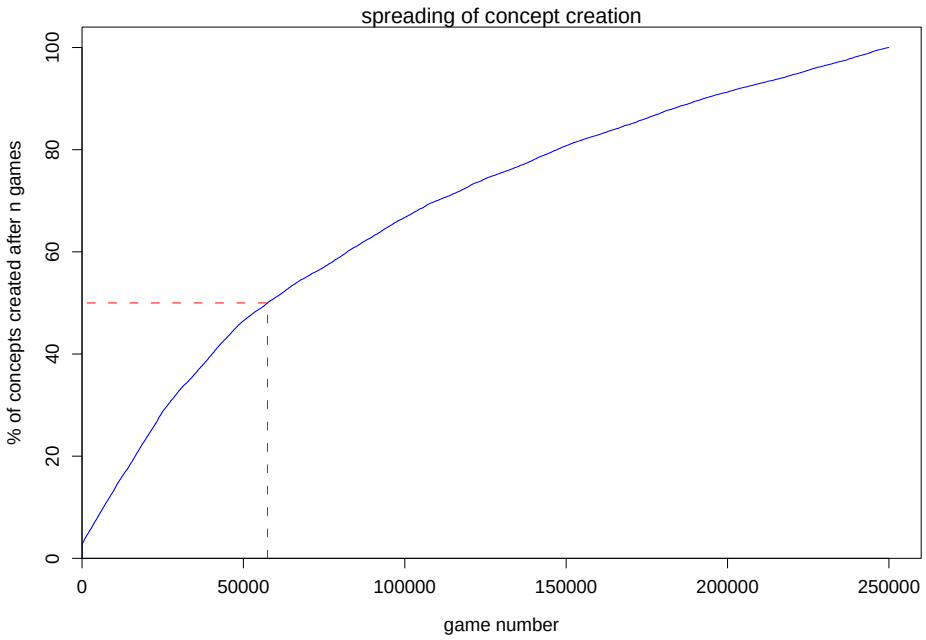

Figure 5.8: *Creation of concepts over time in the 250k-6-rules experiment. The graph shows the percentage of the total number of concepts created during all 250,000 games that is created after playing n games. As indicated, 50% of the concepts is created during the first 60,000 games. The results are averaged over 40 runs.*

rules	amount of concepts created	lifetime of concepts	known concepts after 7,500 games
4	73.9 (23.2)	450 (1308)	4.28 (1.04)
6	38.9 (16.2)	941 (1989)	4.33 (1.33)
8	27.5 (10.2)	1494 (2442)	4.40 (1.22)
10	19.1 (6.5)	2480 (2930)	5.33 (1.21)
12	15.4 (4.6)	3536 (3029)	5.78 (1.37)
16	12.2 (2.4)	5484 (2539)	7.00 (1.43)
20	10.6 (2.0)	6707 (1649)	8.28 (1.40)

Table 5.2: *Average number of concepts, average lifetime of concepts and average number of known concepts at the end of the 7.5k-n-rules experiments. The value between parenthesis is the standard deviation.*

rule and thus also the concept will disappear soon when another new rule is needed. The standard deviation of the lifetime of concepts that is higher than the average lifetime also indicates that the lifetime of concepts is highly skewed. There are many concepts that are forgotten very quickly, but those that survive have much longer lifetimes.

When an agent is allowed to remember many rules, almost no concepts are created. Apparently concepts have a good chance to survive in a population when agents can remember many rules. Probably this has to do with the diversity in the environment as well. The object-set used in these experiments do not need many rules to be conceptualized successfully (two concepts can describe

	single property in the antecedent	multiple properties in the antecedent	
	ordinary rules	ordinary rules	exceptions
rules of this type known	985 (51%)	732 (38%)	198 (10%)
average strength	0.63	0.51	0.37
average number of times used	8902	6446	393
average lifetime	135,155	117,331	77,035

Table 5.3: *Details about the known rules in the 250k-6-rules experiment after playing 250,000 games. The number of rules, the average strength of rules, the average number of times a rule is used and the average lifetime is shown for each type of rules. The ordinary rules are divided into simple rules with only one property in the antecedent and complex rules with multiple properties in the antecedent. Exceptions by definition have at least two properties in the antecedent.*

all objects in the object set) and therefore there is not much pressure on the rules and concepts created by the agents. Because there is not much pressure to make room for new rules when an agent is allowed to remember many rules, the average lifetime of a concept also increases greatly with larger memory sizes.

The number of known concepts at the end of a game increases as well when agents can remember more rules. This is obvious; if an agent can remember more rules, it can also remember more concepts. Note that the number of known concepts is the total number of concepts that occur in the population at the end of a run. Therefore the number of concepts at the end can be higher than the number of rules that fit in the memory of a single agent.

5.4.1 What type of rules is used?

So far has been shown that the agents in the population are able to create concepts and that they are able to achieve high performance in the games they play. But what rules do the agents use for the concepts? Do the agents use mainly simple and general rules, or do they also use more complex rules and exceptions? Table 5.3 gives an overview of the type of rules the agents know after playing 250,000 games in the *250k-6-rules* experiment. The table shows that most rules that are used are ordinary rules with only one property in the antecedent. There is also a quite large proportion of ordinary rules with two or more properties in the antecedent. And 10% of the rules the agents know are exceptions. The exceptions have a shorter average lifetime than the complex ordinary rules and the simple ordinary rules. This could partially explain their lower usage as well, on the other hand the lower usage could explain the short lifetime. If rules are not used often, they also cannot gain strength, meaning they have a higher chance of being forgotten.

The smaller amount of exceptions, their lower strength, lower usage and shorter lifetime cannot be explained because exceptions are more complex though. The values for the complex ordinary rules show that more complex rules can be successful. It also can be questioned whether the lower number of exceptions and their lower usage is a shortcoming of the model. After all, exceptions are meant for exceptional situations that do not occur often.

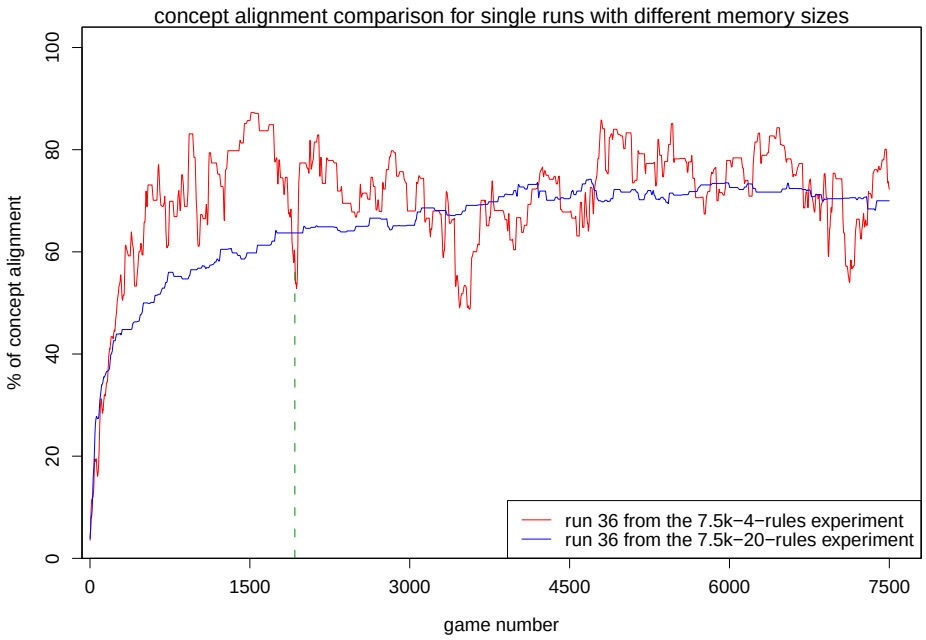

Figure 5.9: *Comparison between the concept alignment of single runs in the experiments with 4 rules (7.5k-4-rules experiment) and 20 rules (7.5-20-rules experiment) per agent. The dashed line marks game 1922 which is discussed in detail in game 5.1.*

5.4.2 Variation in concept alignment

In Figure 5.4 in which the concept alignment for different memory sizes is compared, can be seen that the concept alignment for the experiment with 4 rules shows much more variation than the concept alignment for experiments with more rules. Especially near games 1000, 3100 and 7300 some large deviations from the trend are observed. After inspecting each of the single games of the experiment with 4 rules, no games are found that clearly cause these valleys. The cause for these deviations has to be found somewhere else. Figure 5.9 shows a comparison between the concept alignment during a single run from the *7.5k-4-rules* experiment and the concept alignment during a single run from the *7.5k-20-rules* experiment. It is clearly visible that the alignment changes in the experiment with 20 rules are a lot smoother than in the experiment with only 4 rules.

There are two reasons the concept alignment is much smoother when agents can remember more rules. First of all, agents can remember more rules about a concept. This means an agent probably knows several rules about the concept with multiple meanings. Remembering more rules means that there is less competition between rules. Knowing more rules per concept means that it is probably less catastrophic when one rule is forgotten, the other rules about the concept (partially) take over. The second reason is that the agents can remember more concepts because less competition between rules also implies less competition between concepts. The concept alignment is an average over agents and concepts and by having more concepts, the average will also be

Game 5.1: *Game 1922 from run 36 of the 7.5k-4-rules experiment.*

```
game: 1922 | speaker: agent_0 | hearer: agent_6 | topic: o1 | scene:
 o1: salmon = [canSwim, eatsAnimals, hasFins, hasGills, hasSpinalColumn,
               inFreshWater, inSeaWater, laysEggs]
 o2: common buzzard = [canFly, eatsAnimals, hasBeak, hasSpinalColumn,
                       hasWings, laysEggs]
rules known by the speaker (sn) and rules known by the hearer (hn):
  s1. hasSpinalColumn -> Groofzouf                                  [0.405]
  s2. hasMammaryGlands -> Joonzyg                                   [0.031]
  s3. x(eatsAnimals hasSpinalColumn -> Groofzouf)                   [0.017]
  s4. hasWings -> Joonzyg                                           [0.000]
  h1. eatsAnimals -> Groofzouf                                      [0.452]
  h2. hasSpinalColumn -> Joonzyg                                    [0.485]
  h3. hasMammaryGlands -> Groofzouf                                 [0.001]
  h4. hasWhiteColour -> Groofzouf                                   [0.000]
rules matching topic used by the speaker:                    [s1, s3, s5]
rule invented by the speaker to designate the topic:
  s5. hasSpinalColumn -> Nut
the speaker uses rule s5 and says 'Nut'
the hearer hears 'Nut'
rules about Nut matching o1 used by the hearer:                    [none]
rules about Nut matching o2 used by the hearer:                    [none]
objects the hearer believes are examples of Nut:                   [none]
the hearer shrugs and the speaker points to o1
the hearer decided to learn a new rule:
  h5. canSwim -> Nut
the following rules were updated positively:                       [none]
the following rules were updated negatively:                     [s3, s5]
the following rules were forgotten:                              [s4, h4]
```

calculated over more values. If one of many values changes in this calculation, this will have a smaller impact than the case in which only a few concepts are known.

An example from the the experiment with 4 rules is given in Game 5.1 (see also Figure 5.9). This game causes the average alignment in the population to drop from 63% to 53%. A couple of things happen in this game that cause this big drop. First of all, `agent_0` does not know a concept for the topic and therefore invents a new concept called *Nut*. At this point, none of the other agents know about this concept and therefore all alignments with other agents for this concept will result in an alignment of 0. `agent_6` learns about concept *Nut*, increasing the alignment between `agent_0` and `agent_6` a bit, but not much because their rules about *Nut* differ. On the other hand, `agent_0` forgets rule `s4`, thereby it does not recognize objects with the property `hasWings` as examples of *Joonzyg* anymore. This also causes the alignment to drop, because other agents in the population believe that these objects are examples of *Joonzyg*. A similar thing happens with `agent_6`, who forgets rule `h4`, thereby decreasing the alignment with the rest of the population. As can be observed in this example, a single game can have a big influence on the concept alignment when agents can only remember a few rules.

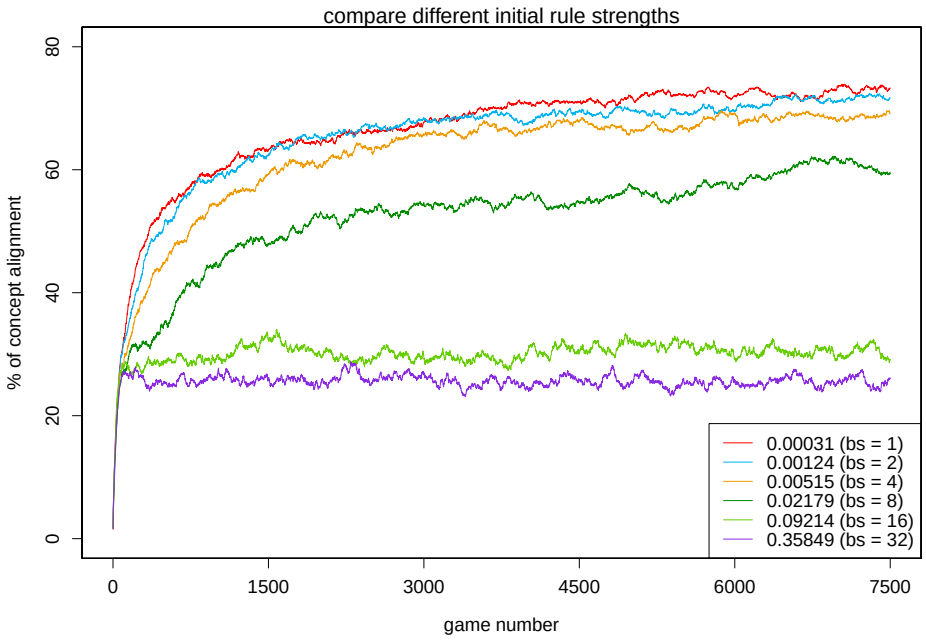

Figure 5.10: *Average concept alignment when using different intial rule strengths. To vary the initial rule strength, the parameter b_s was changed in the model. The experiment was repeated 40 times for each value of b_s and the average concept alignment over the 40 runs was taken.*

5.5 The effect of the initial strength of new rules

All experiments described above were done with very low initial strengths of new rules. The initial strength of rules is only 0.00031 (see Equation 3.1 in Section 3.4). Therefore there is a very high chance new rules will be forgotten if more room is required for other rules. The chances of selecting this rule for use are also very low unless it is the only rule that is applicable or other rules are comparably weak. However, all rules start out very weak, so compared to other new rules the strength is comparable. One might argue that stronger initial rules would be used more often, therefore they would be proved wrong or right sooner. As a result, if a rule was proved wrong often, its strength would decline and the rule would have a higher chance of being forgotten, giving other new rules a chance to develop. However, it seems there is a flaw in this reasoning. When giving new rules a high strength, they will be used more often. It is unknown at that point whether the new rule works well. However, each time a new rule is used that does not work well, it causes the game to fail. Then there is a chance the hearer will create a new rule and needs room for it in its memory. So the hearer has to select an old rule for removal. The old rules, that have already proved to be working well, will have a higher chance of being forgotten in case new rules have a higher initial strength (because the chance of forgetting a rule is the inverse of their strength). So, when using high strength initial rules there are two effects that cause more old good working rules to be forgotten:

- Games will fail more often, because new unproven rules will be used more often. When a game fails, there is a chance that room must be created for a new rule. This could cause a strong old rule to be forgotten.
- When a decision must be made which rule to forget, the inverse of the strength of the rules is used to determine the chance a rule is forgotten. If unproven rules have a high strength, the old rules will have a higher chance to be forgotten because the differences in strength between old rules and new rules is much smaller. This causes more old rules to be forgotten when new rules are created.

The effect of stronger initial rules can be seen in Figure 5.10. It is very clear in the graph that the concept alignment suffers from a high initial rule strength. When the initial rule strength is around 0.10 or higher, the concept alignment does not even reach 30% and looks fairly stable already after only hundreds of games. Further it seems there is a baseline in the performance around 25% because the difference between an initial strength of 0.09 and 0.36 is much smaller than would be expected otherwise.

5.6 Example of evolving a concept

The results discussed in the sections before were all based on measurements of performance, concept alignment and numerical values for concepts and rules. The rest of this chapter will cover a more detailed look at the actual games that were played by the agents and the concepts that evolved from these games. This should provide more insight into the dynamics that come into play with the discussed model. The current section will start with describing the details of the creation of a concept in the population. As example, the concept *Fil* is chosen. This concept was created in run 0 of the *250k-6-rules* experiment (more results from this run are listed in Tables 5.6 and 5.7, these results will be discussed later). In this simulation the concept *Fil* stands for objects with the property `laysEggs`. Table 5.4 shows the rules of all agents about *Fil* after 250,000 games.

As can be seen, all agents know the rule `laysEggs` → `Fil`, which exactly matches the set of objects that are examples of concept *Fil*. This is remarkable because in most cases agents use multiple rules to describe a concept and these rules are often different for each agent. The extra rules known by `agent_2` and `agent_6` do not change the meaning of the concept at all, because both rules are just more specific versions of the general `laysEggs` → `Fil` rule. Such a more specific rule could only change the meaning of *Fil* if there were exceptions for *Fil* as well.

How did this concept that all agents agree upon evolve in the population? And how is it possible the rules of all agents are so alike? Table 5.5 shows the history of the rule `laysEggs` → `Fil` in the population. As can be seen in this table, `agent_2` first came with the rule `laysEggs` → `Fil` in game 3. `Agent_2` learnt this rule from `agent_6` who invented the concept in game 2 with rule `eatsAnimals` → `Fil`. Not much later, in game 33, `agent_3` was the second agent to learn the exact same rule. `Agent_6` and `agent_5` followed in games 209 and 258 respectively. However, they both forgot the rule quickly, using it only in a very limited number of games.

agent	rule(s)
agent_0	`laysEggs` → `Fil`
agent_1	`laysEggs` → `Fil`
agent_2	`laysEggs` → `Fil`, `eatsAnimals` ∧ `laysEggs` → `Fil`
agent_3	`laysEggs` → `Fil`
agent_4	`laysEggs` → `Fil`
agent_5	`laysEggs` → `Fil`
agent_6	`laysEggs` → `Fil`, `hasSpinalColumn` ∧ `laysEggs` → `Fil`
agent_7	`laysEggs` → `Fil`

Table 5.4: *Rules about concept Fil agents know after 250,000 games in run 0 of the 250k-6-rules experiment.*

agent	learnt	forgotten	strength	times used	times successful
agent_0	1794	250000	0.98	22199	22106
agent_1	3436	3569	0	2	2
	5838	250000	0.99	20731	20682
agent_2	3	5766	0.49	257	234
	7096	7135	0	1	1
	7139	7163	0	0	0
	7878	7891	0	0	0
	9586	9636	0	1	1
	13249	13272	0	0	0
	24175	24207	0	1	1
	24224	33352	0.86	766	766
	38404	38514	0	2	2
	38803	38909	0.01	5	5
	46694	250000	0.96	3408	3406
agent_3	33	250000	0.98	25073	24962
agent_4	1583	250000	0.99	25149	25083
agent_5	258	304	0	1	0
	765	823	0	0	0
	2116	250000	0.99	22489	22439
agent_6	209	343	0	2	0
	612	637	0	0	0
	1952	250000	0.77	792	766
agent_7	1158	1239	0	0	0
	3116	5208	0.13	33	33
	10110	250000	0.99	20369	20353

Table 5.5: *A list of all occurances of the rule `laysEggs` → `Fil` during `run_0` of the 250k-6-rules experiment.*

When skipping ahead to game 1,000 it is observed that only two agents are aware of the rule `laysEggs` → `Fil` at that point, namely `agent_2` and `agent_3`. Between game 1,000 and 2,000 `agent_0`, `agent_4` and `agent_6` also learn the rule, while `agent_2` and `agent_3` did not forget the rule. So at game 2,000, from the 8 agents in the population, there are 5 who know the rule `laysEggs` → `Fil`. With more than half of the population knowing this exact rule, the chances for the concept to stabilize are looking good. Only `agent_1`, `agent_5` and `agent_7` do not know about this rule at this point. `Agent_5` and `agent_7` both knew the

Game 5.2: *Game 5766 from run 0 of the 250k-6-rules experiment.*

```
game: 5766 | speaker: agent_1 | hearer: agent_2 | topic: o1 | scene:
  o1: blackbird = [canFly, canSing, eatsAnimals, eatsPlants, hasBeak,
                   hasBlackColour, hasSpinalColumn, hasWings, laysEggs]
  o2: common buzzard = [canFly, eatsAnimals, hasBeak, hasSpinalColumn,
                        hasWings, laysEggs]
rules known by the speaker (sn) and rules known by the hearer (hn):
  s1. eatsAnimals -> Stegslim                                    [0.346]
  s2. hasWings -> Fil                                            [0.358]
  s3. isRuminant -> Porpfrom                                     [0.038]
  s4. eatsAnimals laysEggs -> Stegslim                           [0.023]
  s5. eatsPlants -> Porpfrom                                     [0.209]
  s6. eatsPlants isRuminant -> Porpfrom                          [0.000]
  h1. laysEggs -> Fil                                            [0.490]
  h2. eatsAnimals -> Stegslim                                    [0.561]
  h3. hasMammaryGlands -> Porpfrom                               [0.219]
  h4. hasFourStomachs hasMammaryGlands -> Porpfrom               [0.072]
  h5. eatsAnimals hasSpinalColumn -> Stegslim                    [0.107]
  h6. eatsAnimals hasBeak hasSpinalColumn -> Stegslim            [0.002]
rules matching topic used by the speaker:                          [s5]
the speaker uses rule s5 and says 'Porpfrom'
the hearer hears 'Porpfrom'
rules about Porpfrom matching o1 used by the hearer:             [none]
rules about Porpfrom matching o2 used by the hearer:             [none]
objects the hearer believes are examples of Porpfrom:            [none]
the hearer shrugs and the speaker points to o1
the hearer decided to learn a new rule:
  h7. hasWings -> Porpfrom
the following rules were updated positively:                     [none]
the following rules were updated negatively:                       [s5]
the following rules were forgotten:                                [h1]
```

rule at some point, but they forgot it without using it often. `Agent_1` does not know the rule until it learns it in game 3,436, however also in this case the rule is forgotten before it had a chance to prove itself.

In game 5,766 something interesting happens with `agent_2`: after using the rule 257 times, often with success, it forgets the rule. The rule had built up a reasonable strength, but it was not enough to memorize the rule. Looking at the rules the other agents know, this was actually a bad choice of `agent_2`. The details of game 5,766 are listed in Game 5.2. In game 5,766 `agent_2` decides to learn a new rule about the concept *Porpfrom*. It has no room left in its memory to memorize another rule and thus a rule is chosen to be removed. With the inverse rule strength used as weight that defines the chance whether a rule is removed or not, the rule `laysEggs` → `Fil` (rule `h1`) has a very low chance to be selected. Rule `h6` has a much higher chance to be selected because it has a very low strength. Also rule `h4`, `h5` and `h3` have strengths that are much lower than that of rule `h1`. Nevertheless `laysEggs` → `Fil` is accidentally chosen to be removed.

After this incident `agent_2` struggles to relearn the rule again. As can be seen in Table 5.5, `agent_2` reinvents the rule six times, but each time the rule

		platypus	*cow*	*dog*	*evening bat*	*chimpanzee*	*african elephant*	*salmon*	*goldfish*	*fin whale*	*octopus*	*blackbird*	*common buzzard*	*barn owl*	*mute swan*	*black swan*	*penguin*
run 0	*Porpfrom*		✓	✓		✓	✓		✓			✓			✓	✓	
	Fil	✓						✓	✓		✓	✓	✓	✓	✓	✓	✓
	Stegslim	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓			✓
run 19	*Mainhood*		✓	✓		✓	✓		✓			✓			✓	✓	
	Vren	✓						✓	✓		✓	✓	✓	✓	✓	✓	✓
	Brurnstraig	✓		✓	✓	✓		✓	✓	✓		✓	✓	✓			✓
	Craiglurm	✓		✓	✓	✓		✓	✓	✓		✓	✓	✓			✓
run 24	*Heirmfreeg*		✓	✓		✓	✓		✓			✓			✓	✓	
	Lamkyrn	✓						✓	✓		✓	✓	✓	✓	✓	✓	✓
	Mairswreem	✓		✓	✓	✓		✓	✓	✓		✓	✓	✓			✓
run 28	*Vryrt*		✓	✓		✓	✓										
	Sloof	✓						✓	✓		✓	✓	✓	✓	✓	✓	✓
	Groorp	✓		✓	✓	✓		✓	✓	✓		✓	✓	✓			✓
run 29	*Wrourmstreilk*		✓	✓		✓	✓		✓			✓			✓	✓	
	Steifblet	✓						✓	✓		✓	✓	✓	✓	✓	✓	✓
	Writklourn	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓

Table 5.6: *An overview of which concepts apply to which objects, from five runs with 8 agents that each can remember 6 rules, and that ended with 100% concept alignment within 250,000 games.*

is forgotten within 100 games while having used the rule at most one time. In game 24,224 `agent_2` invents the rule for the eighth time. This time the rule proves very successful. The rule is remembered by the agent over 9,000 games and is used 766 times, each time successfully! However, bad luck strikes again for `agent_2`. While `laysEggs` → `Fil` was the strongest rule known by `agent_2`, it was chosen to be removed (the actual chance that this rule was selected for removal was only 0.034%).

In the meantime, all other agents have stable versions of this rule for a long time already. `agent_7` was the last agent to develop a stable version of `laysEggs` → `Fil` and discovered this rule in game 10,110 already. After `agent_2` finally developed a stable version of `laysEggs` → `Fil` in game 46,694, no more rules about the concept *Fil* were created in the population. This can be explained because at that point all agents agreed about the concept and none of them forgot any rules about *Fil*. Therefore there was never the need to create new rules about concept *Fil*.

5.7 Meaning of created concepts

It is interesting to see which concepts are created in the population and how these differ between several runs. To compare the created concepts between several runs, the results from all runs of the *250k-6-rules* experiment in which the concept alignment reached 100% are used. These results can be seen in Table

run	concepts	possible rule(s) that describe the concept
0	*Fil*	
19	*Vren*	
24	*Lamkyrn*	`laysEggs` → `C`
28	*Sloof*	
29	*Steifblet*	
0	*Porpfrom*	
19	*Mainhood*	
24	*Heirmfreeg*	`eatsPlants` → `C`
29	*Wrourmstreilk*	
19	*Brurnstraig*, *Craiglurm*	
24	*Mairswreem*	`eatsAnimals` → `C`
28	*Groorp*	
29	*Writklourn*	`hasSpinalColumn` → `C`
28	*Vryrt*	`eatsPlants` ∧ `hasMammaryGlands` ∧ `hasSpinalColumn` → `C`
0	*Stegslim*	`hasSpinalColumn` → `C` ×(`canSwim` ∧ `hasSpinalColumn` ∧ `eatsPlants` ∧ `hasBeak` → `C`)

Table 5.7: *An overview of possible rules that describe the behaviour of the population. These rules are the simplest possible rules that comply with the concepts used in table 5.6. The rules do not reflect the actual rules known by the agents (which could be completely different, but the result of using them on this data-set would be the same).*

5.6 and Table 5.7. In this experiment, five of the forty runs reached a concept alignment of 100%. The concepts with the accompanying rule(s) that make up the concept are listed in Table 5.7. Because the actual rules used by each agent can differ, the listed rule (or combination of rules) is the simplest rule that can explain the observed behaviour. For example some of the agents in run 0 actually use the rules `eatsAnimals` → `Stegslim`, `hasSpinalColumn` → `Stegslim` and ×(`hasBeak` ∧ `hasSpinalColumn` → `Stegslim`) for the concept *Stegslim*. This combination of rules has the same meaning as the rules listed in Table 5.7 with respect to the set of objects used during these experiments.

A few things can be noticed quickly from these tables. First of all, in all five runs a concept was created for animals with the property `laysEggs`. Apparently this is a concept that has some advantageous properties in this specific situation. The same could be said for the concepts that were created for animals with the property `eatsPlants` or `eatsAnimals` that were used in respectively four of five and three of five runs. The concept that is created in run 29 for animals with the property `hasSpinalColumn` only occurs one time, which is remarkable because the property `hasSpinalColumn` is the most occurring property in the set of objects that was used during the simulations. In run 0 and run 28, some more complex concepts are created. The concept in run 28 requires multiple properties to be observed while the one in run 0 requires an exception to be explained.

Another remarkable observation is the synonym that occurs in run 19. There are two concepts that cover exactly the same set of objects. This is unexpected because in the model synonyms are disadvantageous. Table 5.8 lists the rules

agent	rules for *Brurnstraig* and *Craiglurm*
agent_0	eatsAnimals ∧ hasSpinalColumn → Brurnstraig eatsAnimals ∧ hasSpinalColumn ∧ laysEggs → Brurnstraig eatsAnimals → Craiglurm
agent_1	eatsAnimals → Brurnstraig hasSpinalColumn → Brurnstraig ×(eatsPlants ∧ hasSpinalColumn → Brurnstraig) eatsAnimals ∧ hasSpinalColumn → Craiglurm
agent_2	eatsAnimals → Brurnstraig eatsAnimals ∧ hasSpinalColumn → Craiglurm
agent_3	eatsAnimals ∧ hasSpinalColumn → Brurnstraig eatsAnimals → Craiglurm
agent_4	eatsAnimals ∧ hasSpinalColumn → Brurnstraig eatsAnimals ∧ hasSpinalColumn ∧ laysEggs → Brurnstraig eatsAnimals → Craiglurm
agent_5	eatsAnimals ∧ hasSpinalColumn → Brurnstraig eatsAnimals → Craiglurm
agent_6	eatsAnimals → Brurnstraig eatsAnimals ∧ hasSpinalColumn → Craiglurm hasBeak → Craiglurm ×(canSwim ∧ hasBeak → Craiglurm)
agent_7	eatsAnimals → Brurnstraig hasSpinalColumn → Brurnstraig ×(eatsPlants ∧ hasSpinalColumn → Brurnstraig) eatsAnimals ∧ hasSpinalColumn → Craiglurm

Table 5.8: *Rules created for the synonyms Brurnstraig and Craiglurm in run 19 of the 250k-6-rules experiment.*

used for the concepts *Brurnstraig* and *Craiglurm* in run 19. When taking a closer look at these rules, it can be observed that none of the agents actually uses equal rules for both concepts. Instead the rules for both concepts differ from each other for all agents and therefore there is no competition between these rules in the simulation (as explained in Section 3.6.4, rules are considered synonyms if their concepts differ, but they use exactly the same antecedent).

The main difference between the concepts *Brurnstraig* and *Craiglurm* is that one of them is used in a more specific way. This differs for each agent. For agent_2 the concept *Craiglurm* is more specific than the concept *Brurnstraig* because it requires the animal to have the property hasSpinalColumn as well besides the property eatsAnimals. For agent_3 it is exactly the other way around. The set of objects used in the simulation does not include animals that have the property eatsAnimals but do not have the property hasSpinalColumn. Therefore, with respect to the objects used in the simulation, there is no difference between the concepts.

The rules used for the concepts *Brurnstraig* and *Craiglurm* also nicely show the differences in which agents come up with fully aligned concepts. While the main difference between both concepts is that one of the rules has the antecedent eatsAnimals ∧ hasSpinalColumn while the other rule has the shorter antecedent eatsAnimals, these antecedents are not used by all agents. For example, agent_7 uses the rule hasSpinalColumn → Brurnstraig that causes all animals with the property hasSpinalColumn to be examples of *Brurnstraig*. The

Game 5.3: *Game 152,022 from run 19 of the 250k-6-rules experiment.*

```
game: 152022 | speaker: agent_0 | hearer: agent_6 | topic: o1 | scene:
  o1: barn owl = [canFly, eatsAnimals, hasBeak, hasSpinalColumn,
                  hasWings, huntsAtNight, laysEggs]
  o2: mute swan = [canFly, canSwim, eatsPlants, hasBeak, hasSpinalColumn,
                   hasWebbedFeet, hasWhiteColour, hasWings, laysEggs]
rules known by the speaker (sn) and rules known by the hearer (hn):
  s1. eatsAnimals hasSpinalColumn -> Brurnstraig                  [0.761]
  s2. eatsPlants hasSpinalColumn -> Mainhood                      [0.978]
  s3. eatsAnimals -> Craiglurm                                    [0.942]
  s4. eatsAnimals hasSpinalColumn laysEggs -> Brurnstraig         [0.003]
  s5. laysEggs -> Vren                                            [0.875]
  s6. canFly hasWebbedFeet -> Vren                                [0.000]
  h1. eatsAnimals -> Brurnstraig                                  [0.707]
  h2. eatsPlants hasSpinalColumn -> Mainhood                      [0.966]
  h3. laysEggs -> Vren                                            [0.507]
  h4. hasBeak -> Craiglurm                                        [0.022]
  h5. eatsAnimals hasSpinalColumn -> Craiglurm                    [0.159]
  h6. eatsAnimals hasGills hasSpinalColumn -> Craiglurm           [0.002]
rules matching topic used by the speaker:                           [s3]
the speaker uses rule s3 and says 'Craiglurm'
the hearer hears 'Craiglurm'
rules about Craiglurm matching o1 used by the hearer:               [h5]
rules about Craiglurm matching o2 used by the hearer:               [h4]
objects the hearer believes are examples of Craiglurm:          [o1, o2]
the hearer choses o2 as example of Craiglurm and points to o2
rules used by the speaker to verify o2:                           [none]
the speaker did not agree that o2 is an example of Craiglurm
the hearer decided to learn a new rule:
  h7. x(canSwim hasBeak -> Craiglurm)
the following rules were updated positively:                        [h5]
the following rules were updated negatively:                    [s3, h4]
the following rules were forgotten:                                 [h6]
```

animals that have the property `eatsAnimals` happen to form a subset of this set of objects. However, not all animals in the simulation that have the property `hasSpinalColumn` also have the property `eatsAnimals` and therefore an exception is used to exclude those animals that have the property `eatsPlants`. Further it can be seen that the rules `hasSpinalColumn` → `Brurnstraig` and ×(`eatsPlants` ∧ `hasSpinalColumn` → `Brurnstraig`), used by `agent_7`, are superfluous because they match the same set of objects as rule `eatsAnimals` → `Brurnstraig` which is also known by `agent_7`.

A different set of rules was used by `agent_6` that used the rule `hasBeak` → `Craiglurm`. From all objects in the simulation that have the property `hasBeak` only the mute swan and the black swan do not have the property `eatsAnimals`. This is corrected by adding the exception ×(`canSwim` ∧ `hasBeak` → `Craiglurm`). This exception works because the mute swan and the black swan both have the property `canSwim` and thus the exception prevents `agent_6` to use the concept *Craiglurm* for the mute swan and the black swan.

Just like for `agent_7`, for `agent_6` it is also true that the agent would have

known exactly the same concepts for objects if it would forget the rule for the property `hasBeak` and the exception for that rule. So why did this agent learn these rules? In game 152,022 the exception was introduced, the game is shown in Game 5.3. In this game the speaker uses the concept *Craiglurm* for the topic, the barn owl. The hearer uses the rule `hasBeak` → `Craiglurm` to conclude that the mute swan is an example of *Craiglurm* and points to the mute swan. As can be seen, the rule that was used for the mute swan is quite weak, its strength is only 0.022, but because it is the only rule about *Craiglurm* that applies to the mute swan, it is used nevertheless. According to the speaker a *Craiglurm* must at least have the property `eatsAnimals` and thus the speaker disagrees and points to the barn owl to make clear this is the actual topic of the game. The hearer now has several options: just update the current rules, create a new ordinary rule for the topic or create a new exception for the selected example. In this case the hearer decided to create a new rule, but because its memory is full, one rule must be forgotten to make place for the new rule. Based on its low strength, rule `h6` is forgotten. With a strength of only 0.002 this rule is by far the weakest rule, even weaker than the rule about *Craiglurm* used for the mute swan. There is now room for a new rule and by chance an exception for rule `h4` was chosen to be added to the hearers memory.

Now have a look at the reason why rules `h4` and the newly introduced rule `h7` are not forgotten later in the simulation. Suppose `agent_6` is the speaker in a future game and the mute swan is the topic. What rules could `agent_6` choose from? It knows three ordinary rules that are applicable to the mute swan, those are rules `h2`, `h3` and `h4`. Rules `h2` and `h3` are far stronger than rule `h4` and therefore rule `h4` has a very small chance of being chosen. Even if `h4` would be chosen, there would be an exception that excludes the concept *Craiglurm* to be concluded and therefore `h4` still would not be used. All other animals in the simulation that have property `hasBeak` can be described by the concepts following from one of the stronger rules (rules `h2` and `h3`). So rule `h4` will not be used anymore by `agent_6` when it has the role of speaker. For rule `h7` there is a small chance of being used, but this only happens if rule `h4` was chosen initially and as explained, the chance this happens is very small. Rules `h2` and `h3` are much stronger and therefore more likely to be chosen.

What about using rule `h4` when `agent_6` is the hearer? Rule `h4` could only be used if the speaker has used the concept *Craiglurm*, because in the games a hearer is only interested in rules about the concept that was uttered by the speaker. So if the speaker says *Craiglurm* the hearer will check all objects in the scene to see if it is an example of *Craiglurm*. Suppose one of the objects in the scene is the mute swan. What will happen is that `agent_6`, by using `h4` and rule `h7`, will conclude that the mute swan is not an example of *Craiglurm*. Therefore `agent_6` will choose one of the other objects in the scene, or it will shrug. Assume `agent_6` selects an object that is not the mute swan as example of *Craiglurm*. If the speaker agrees with this choice, `agent_6` cannot conclude anything about the mute swan. The hearer and the speaker have not communicated about the mute swan, so the hearer cannot know whether the speaker thinks the mute swan is a *Craiglurm*. Thus the hearer cannot update the rules used to decide the mute swan is not an example of *Craiglurm*. In this case rules `h4` and `h7` are not considered being used and thus they are not updated.

There is a situation though in which rule `h4` is updated. This situation occurs when another rule about *Craiglurm* is successfully used by `agent_6` on an object

rule h4 does not apply to. In this case rule h4 is seen as a homonym by agent_6 because it cannot be applied and thus rule h4 has a different meaning than the rule that is currently used successfully. Rule h4 is now updated negatively because it is considered to be a homonym. An example would be using rule h5 successfully on the object dog. The rule h5 can be applied to a dog and it is about *Craiglurm* just like h4. But h4 cannot be applied to a dog because a dog does not have the property hasBeak. Therefore h4 is considered as a homonym of h5 and thus h4 will be updated negatively. Each time such a situation occurs, the chances of selecting rule h4 will get smaller because the strength of the rule is getting weaker.

Rule h4 has now reached a state in which it only gets weaker and the chances of being selected are getting smaller and smaller. For this reason, the exception h7 also does not have to be used because its only use is to prevent reaching a conclusion from rule h4 which is virtually never used. Rules h4 and h7 are useless for agent_6, because their strengths are so low. The chances they are forgotten when a new rule is needed is very large. However in this simulation agent_6 does not create any new rules after rule h7 was learnt. The reason for this is that agent_6 almost always achieves success in the games it plays after inventing rule h7. In case there were any games in which agent_6 had the role of hearer (and thus being able to create a new rule on failure) and the game was unsuccessful, it decided not to learn a new rule. This way rules h4 and h7 could be remembered by agent_6. While the rules were not necessary, the memory space they occupied was not necessary either. Neither did rules h4 and h7 cause any games to fail.

5.8 Concepts with exceptions

In Table 5.7 the concept *Stegslim* was shown that most easily could be described by using an exception. Looking back at Table 5.6 the concept *Stegslim* is shown to cover all animals except the octopus, the mute swan and the black swan. Are these animals missing a property that is observed in all other animals? This is not the case. The only property all other animals have in common is the property hasSpinalColumn. However, the mute swan and the black swan both also have this property. There is no other property the other animals have in common, so the concept cannot be explained by an ordinary rule that is a combination of properties either. This means the only way to explain the concept is by a combination of ordinary rules or using an exception that excludes the mute swan and the black swan from the concept.

A combination of ordinary rules that covers all animals that are part of the concept *Stegslim* could be the rules hasMammaryGlands $\rightarrow$ Stegslim and eatsAnimals $\rightarrow$ Stegslim. This would cause one problem though; these rules would compete with each other as they are considered as homonyms for all animals that have only one of these properties. Another possibility is given in Table 5.7. In this case the ordinary rule hasSpinalColumn $\rightarrow$ Stegslim is combined with the exception $\times$(canSwim $\wedge$ hasSpinalColumn $\wedge$ eatsPlants $\wedge$ hasBeak $\rightarrow$ Stegslim). However, this is quite a complex exception as there are four properties in its antecedent and thus it is hard to learn for the agents.

Table 5.9 shows the actual rules used for the concept for each agent in the population. All agents know the rule hasSpinalColumn $\rightarrow$ Stegslim. However

agent	rules for *Stegslim*
agent_0	eatsAnimals → Stegslim hasSpinalColumn → Stegslim hasMammaryGlands ∧ hasSpinalColumn → Stegslim ×(hasBeak ∧ hasSpinalColumn → Stegslim)
agent_1	eatsAnimals → Stegslim hasMammaryGlands → Stegslim hasSpinalColumn → Stegslim ×(hasSpinalColumn ∧ hasWebbedFeet → Stegslim)
agent_2	eatsAnimals → Stegslim hasSpinalColumn → Stegslim ×(hasBeak ∧ hasSpinalColumn → Stegslim)
agent_3	eatsAnimals ∧ hasSpinalColumn → Stegslim hasSpinalColumn → Stegslim ×(canSwim ∧ hasSpinalColumn → Stegslim)
agent_4	eatsAnimals ∧ hasSpinalColumn → Stegslim hasMammaryGlands ∧ hasSpinalColumn → Stegslim hasSpinalColumn → Stegslim ×(canSwim ∧ hasSpinalColumn → Stegslim)
agent_5	eatsAnimals ∧ hasSpinalColumn → Stegslim hasMammaryGlands ∧ hasSpinalColumn → Stegslim hasSpinalColumn → Stegslim ×(hasSpinalColumn ∧ hasWings → Stegslim)
agent_6	eatsAnimals → Stegslim hasSpinalColumn → Stegslim ×(hasBeak ∧ hasSpinalColumn → Stegslim)
agent_7	eatsAnimals → Stegslim hasSpinalColumn → Stegslim ×(hasBeak ∧ hasSpinalColumn → Stegslim) ×(hasSpinalColumn ∧ isVenomous → Stegslim)

Table 5.9: *Rules about concept Stegslim known by the agents at the end of run 0 of the 250k-6-rules experiment. This table shows that each agent knows at least one exception to describe the concept.*

none of them uses the complex exception that would only exclude both swans. Instead all agents know a simpler exception that not only excludes the swans, but also some other animals. In addition the agents use a second or even third ordinary rule that makes sure that the concept *Stegslim* can still be used for those animals that are not swans but were excluded by the exception.

Agent_2 and agent_6 both know exactly the same rules. The rule hasSpinalColumn → Stegslim covers all animals except the octopus. Using the exception ×(hasBeak ∧ hasSpinalColumn → Stegslim) the mute swan, the black swan, the penguin, the common buzzard, the barn own, the blackbird and the platypus are excluded. However except both swans, all these animals have the property eatsAnimals and thus they are covered with the rule eatsAnimals → Stegslim. Therefore the combination of rules known by agent_2 and agent_6 exactly covers the concept *Stegslim*.

Similar approaches are used by the other agents to describe *Stegslim*. All agents use multiple ordinary rules for *Stegslim*, this causes also some homonyms to occur in the population. However apparently the competition between the

homonyms is not strong enough to let one of the rules win. A few agents use the combination of ordinary rules `hasSpinalColumn` → `Stegslim` and `eatsAnimals` ∧ `hasSpinalColumn` → `Stegslim`. This causes only the more complex rule to be seen as a homonym in case the general rule was used for an animal without the property `eatsAnimals`. The reason is that the more complex rule cannot be applied in case an object does not have the property `eatsAnimals` and thus the rule is considered to have a different meaning.

This shows how the concept can be described, but how did this concept actually emerge? The games in which the exceptions are created all have one thing in common: the speaker either knows the rule `eatsAnimals` → `Stegslim` or the rules `hasSpinalColumn` → `Stegslim` and an exception about *Stegslim* while the hearer uses the more general `hasSpinalColumn` → `Stegslim`. In all games the hearer pointed to an object that did not have the property `eatsAnimals` and therefore the game failed. The exception thus emerges from the confusion about the concept and by chance the agents chose to repair the situation by adding an exception (they could have chosen to add another ordinary rule instead). The concept that needs an exception therefore seems to be created by melting together two possible meanings of the concept: the meaning `eatsAnimals` → `Stegslim` and the meaning `hasSpinalColumn` → `Stegslim`. For the meaning `eatsAnimals` → `Stegslim` all objects are kept, while for the meaning `hasSpinalColumn` → `Stegslim` some objects (the mute swan and the black swan) are lost.

5.9 Why is there no concept for vertebrates?

The most occurring property in the simulation is `hasSpinalColumn`. However, in the runs discussed earlier, only once a concept for `hasSpinalColumn` survived in the population (run 29 in Table 5.7). Why is this concept not used more often? A possible reason might be found in run 19 and run 24. The property `hasSpinalColumn` initially was used in many cases in these runs. When looking at run 19 and run 24 the concepts that survived until the end of the simulation, at some point all had rules with the property `hasSpinalColumn`. At that point these rules formed possible hypotheses for the meanings of the concepts that were formed in the population. These hypotheses did not survive in run 19 and run 24 though. Instead other hypotheses won the competition for the meanings of the concepts and thus no concepts were formed with the meaning `hasSpinalColumn`.

How is it possible that the hypotheses with property `hasSpinalColumn` lost the competition for the meanings of concepts in these runs? Instead other meanings like `laysEggs`, `eatsAnimals` and `eatsPlants` survived and formed the final concepts in these runs. In the object set used during the experiments, all these meanings cover a subset of the objects that have the property `hasSpinalColumn`. In other words; each object that has the property `eatsAnimals` also has the property `hasSpinalColumn`. The same is valid for the properties `eatsPlants` and `laysEggs` (the only exception is the very infrequently occurring octopus which has the property `laysEggs` but not the property `hasSpinalColumn`).

It seems that meanings that cover a subset of objects were favoured over meanings that cover a larger set of objects. How can this be explained? This preference is likely to be a property of the described model. Suppose two

agents are playing a game and one of the agents knows the more general rule `hasSpinalColumn` → `Concept` while the other knows the more specific rule `eatsAnimals` → `Concept`. The specificity difference between these rules does not follow from the the properties in their antecedents, it follows from the object set in which each animal that has the property `eatsAnimals` also has the property `hasSpinalColumn`, but not the other way around.

If these two agents are playing a game in which both objects have the property `hasSpinalColumn` but only one object has the property `eatsAnimals`, the agent with the more general rule will have a disadvantage. In case the rule `hasSpinalColumn` → `Concept` is used to describe the topic, it does not matter which object the agent with the rule `eatsAnimals` → `Concept` chooses as an example. Either object will be considered as a valid example by the agent with the rule about the property `hasSpinalColumn`. On the other hand, if the agent with the rule `eatsAnimals` → `Concept` uses its rule to describe the topic, only the topic will be considered as a valid example (as the other object did not have the property `eatsAnimals`). If the agent with the rule `hasSpinalColumn` → `Concept` points to the object without the property `eatsAnimals`, the game fails. At this point there is a chance that this agent will invent a new rule about the concept, and thus the rule `hasSpinalColumn` → `Concept` will suffer from more competition.

To summarize, this property of the model causes agents that use the more general meaning of the concept to create more rules for the concept. The agent with the more specific meaning of the concept will never be corrected though and therefore will not create new rules about the concept as often. This effect could form the reason for concepts for vertebrates do not seem to occur often.

5.10 Summarizing the results

In this chapter several properties of the behaviour of the model were discussed. First the performance of the agents in the population was investigated and it was shown that in all experiments the agents were able to achieve a high rate of success during the games. The size of the agents' memories, the size of the population and the size of the scene's in which the games are played all had an influence on the performance. Larger memory sizes, smaller populations and smaller scene's caused higher performance in the populations. The concept alignment was discussed as well and showed similar effects as the performance measures, but was interesting nevertheless because it is a stricter measurement and allows to gain a better insight in the extent of agreement between agents.

The number of different rule types was discussed to see how often more complex ordinary rules and exceptions were used by agents. Both, the complex ordinary rules and the exceptions, were shown to be used by agents. However complex ordinary rules occurred more often and were stronger than exceptions, showing that agents were not able to use exceptions as efficiently as ordinary rules.

In the *250k-6-rules* experiment it was shown that agents continue to improve their agreement about concepts. A high degree of concept alignment was achieved and in some runs the agents even reached 100% concept alignment. Several effects observed in this experiment were discussed in detail, like the evolution of a concept, the creation of multiple concepts that covered the same set

of objects, the forming of a concept with exceptions and finally why the most occurring property is not used more often to form a concept. Many insights were gained about the emergent properties of the model with these in depth discussions.

In the next chapter some of these emergent properties are discussed again with respect to related work. Some comparisons are made and possible solutions for unexpected or undesired behaviour will be discussed.

Chapter 6

Discussion

So what could be learnt from developing the model and running the experiments? First of all, using defeasible rules can lead to successful conceptualization of the environment. The agents are clearly achieving high communicative success and thus the emergent behaviour does work to some extent. However there are also a few remarks, some features that did not work out as was hoped for. These will be discussed in the following sections and possible adaptations will be considered that could improve the workings of the model.

6.1 Complex rules and exceptions

Concepts that need exceptions or complex ordinary rules about combinations of properties are not used often and they do not seem to be of key importance in the model. One of the problems with exceptions is that there is no need for the agents to deny that some object is an example of a concept. For example, when looking at the platypus; there is no reason why the agents would not believe that the platypus is a mammal and a bird. In our world we use a strict distinction between these two concepts: an animal that is a mammal can not be a bird and vice versa. In the world of the agents who play the conceptualization game, there is no such distinction. What possible mechanisms could introduce such a distinction? The answer probably has to be found in the goals the agents have during communication, or in the pressure the environment provides. The proposal for a possible mechanism for the evolution of inference, as suggested by Skyrms (2004) might be pointing in the right direction. There must be a reason to have a distinction between the two concepts to cause the distinction to enter the rules of the agents. Skyrms suggest that this distinction could be the type of threat different animals pose. Each type of threat requires a different defence response. Therefore the mechanism suggested by Skyrms introduces a reason grounded in the environment that could be exploited to emerge mutual exclusion of concepts.

Examples of properties besides the threat posed by animals, could be the edibility of an animal, or the edibility of the eggs they lay. What would be necessary is a property that could be linked to a concept. A property that is important for a population. This would allow rules like `Bird` → `eggsEdible`. Now if an animal occurs for which it is uncertain whether it is a bird or a

mammal, the property linked to the concept might help deciding. Suppose the eggs of the platypus would be poisonous and thus not edible, than a rule like $\times$(`laysEggs` $\wedge$ `hasMammaryGlands` $\rightarrow$ `Bird`) would become useful because it quickly discards the platypus as a bird. Therefore the agent can quickly decide that it should not eat the eggs of the platypus.

Perhaps the platypus is not the best example that could be used to explain the desire to disallow the occurrence of multiple concepts. Why did mankind agree that a platypus is not a bird but a mammal? The reason is probably found in the desire of scientists to come up with a strict taxonomy for animals that did not allow any confusions about animals. That property of a taxonomy is useful in science because it allows scientists to be precise in their wordings. It allows them to talk about mammals and birds without having to specify whether animals like the platypus are considered or not. The grounding for classifying the platypus as a mammal and not as a bird is probably because the platypus and mammals have a more recent common ancestor than the platypus and birds. Such a complex property is unknown to the simple agents in the simulation that was discussed, and they also did not have any reason to make the distinction between birds and mammals.

There could also be a role for the autotelic principle as discussed by Steels (2004) and Steels and Wellens (2007) that was used to scaffold the difficulty of tasks that have to be learnt by agents (see also Section 2.1.2). Steels suggests increasing the challenge for agents when they master a certain task, thereby the skill level of agents would have to increase as well. This method might turn out to be useful in the current model. A possible use could be to increase the difficulty of the game discussed in the current work by not allowing the hearer to point to any example of the concept, but only consider the topic itself as correct. This would lead to more specialized concepts and thus to more complex rules. Other ways of incorporating the autotelic principle could be allowing more complex rules as soon as the agents master using simple rules.

6.2 The poverty of the stimulus

A different problem with the discussed model is that rules for concepts must be used successfully many times before they gain enough strength to become stable. This is contradicting the poverty of the stimulus, which suggests that only a few examples are needed to learn. However, as shown in the experiments, the agents are not able to reach communicative success when the initial strength of new rules is too high (which would allow new rules to become stable much faster). In case the newly created rules were too strong, they forced older stable rules to be forgotten, resulting in bad performance. A different update mechanism might be needed to ensure that this does not happen anymore. For example it might be possible only to forget rules that contributed to a game that failed (instead of a random rule, based on the inverse of its strength). This could allow stronger new rules, which in turn would allow more new rules to be used soon after they were learnt. Using more new rules would improve the exploration of possible rules, instead of relying mainly on older stable rules (which is what happens when new rules have very low strengths).

Another approach might be starting out with more specific rules that perfectly fit the first example seen by the agent (and thus incorporates multiple

properties). As the agent sees more examples of the concept, it can discard some of the properties in the rule, thereby generalizing the rule. Such an approach would have much in common with the approach taken by Wellens et al. (2008) that uses weighted connections between properties and meanings to determine the certainty that the property belongs to the meaning. However, an approach that starts very specific and becomes more general by seeing more examples is a lot stricter than the approach taken by Wellens et al. (2008). Whether starting out with specialized rules instead of general rules would prove to be a good solution to generate better rules has to be seen though. In this case a speaker would not often use such a specialized rule to find the concept for the topic. In all cases the very specific rule would not apply, the agent would generate a new concept. Or a hearer would fail to recognize the concept used by the speaker. This could pollute the language with many different concepts, causing learning to be difficult. However, introducing an operator that allows an agent to remove one of the properties from a rule could still be a good idea. Currently only increasingly complex rules could be created. Taking a step back instead to a more general rule might prove to be a good solution in case an agent relied on the wrong property in the rule. The agent would have a mechanism to remove that property from the rule, while maintaining the correct part.

Another mechanism that could help solving the problem of new rules that are too weak to be chosen, could be making a distinction between new rules that are educated guesses, or new rules that are completely random. For example, a speaker that has to come up with a new concept because it does not know any concepts for the topic should not generate a rule that is initially strong; the agent knows he is just inventing something new, so it cannot expect the new rule to be a trustworthy one. On the other hand, suppose an agent with the role of hearer considers two objects in the scene as examples of the used concept and for both objects it uses the same rule. After the hearer selected one of these objects, the speaker disagreed and pointed to the other object. In this case apparently the rule worked very well for one object, but not for the other. Therefore the hearer has good reasons to believe that the used rule was correct, even though the speaker did not agree with the selected example. The used rule was still correct for the topic and thus the hearer has good grounds to believe that there must be an exception that applies to the selected example. This could be considered as an educated guess for making this exception. There are good reasons to believe that such a new exception is correct. Therefore the agent could choose to give this exception a higher initial strength than a new rule that was completely random when it was invented.

6.3 Problems with conjunctive rules

One of the problems that conjunctive rules have, is that they are less general than rules with only one property in the antecedent. Therefore these rules can be used in fewer situations. Wellens et al. (2008) argue that this will cause conjunctive rules not to develop in simulations that use competition between rules about the same concept. However in the current work, a partial solution to that problem is presented by letting agents always use the most specific rule that applies to an object. This ensures that the more specific version of a rule is used as often as possible. Thus, in case the more specific version of the rule

could be applied to half of the objects to which the more general rule applies, both rules would have an equal chance of surviving. However, there is one problem in the current implementation with the method of updating homonyms that counteracts this mechanism. The presented model looks for homonyms by looking at all ordinary rules about the same concept that can not be applied to the object about which was successfully communicated. This means that when a general rule was used successfully for an object, a more specific rule about the same concept that can not be applied to the object, would be seen as a homonym. The general rule would not have this problem the other way around, because by definition it can also be applied to an object to which the more specific rule could be applied. This problem could easily be circumvented by not seeing more specific versions of a rule as homonyms. That way, there would be no disadvantages anymore for more specific rules in the model.

From this example, it can be seen that the updating of rules requires a very precise approach to make sure the emergent behaviour of the model is steered in the right direction. Any skewed pressures that force the behaviour into a certain direction, will probably be seen back in the results. The updating of homonyms in the presented model is an example of an unintended pressure that probably was a factor that caused difficulties for more complex rules. Nevertheless, it was shown that more complex rules are used quite often, but concepts that required complex rules were not observed very often.

6.4 Why using exceptions is profitable

To be able to capture the phenomena in the environment efficiently, some form of defeasibility or negation is necessary. Suppose an agent would like to remember rules for mammals and for birds. The agent could do that with the rules `laysEggs` $\rightarrow$ `Bird` and `hasMammaryGlands` $\rightarrow$ `Mammal`. But if the agent would like to remember that a platypus is not a bird, something else is needed. An exception to the concept *Bird* is a possibility: $\times$(`hasMammaryGlands` $\wedge$ `laysEggs` $\rightarrow$ `Bird`). However, there is another option if negations would be allowed in the rules. In that case an agent could adapt the rule for the birds to `laysEggs` $\wedge$ $\neg$`hasMammaryGlands` $\rightarrow$ `Bird`. So, why were negations not used in this work? After all, it would allow one rule less to capture the same information. The reason is that there are a few problems with rules that include negations. First of all, they could grow into very complex rules that have to include a lot of special cases. Suppose after seeing the platypus, a snake is observed as well. Snakes lay eggs as well, so the rule would have to be adapted to: `laysEggs` $\wedge$ $\neg$`hasMammaryGlands` $\wedge$ $\neg$`isCrawler` $\rightarrow$ `Bird`. Thereafter a turtle is observed, again the rule would have to be adapted to: `laysEggs` $\wedge$ $\neg$`hasMammaryGlands` $\wedge$ $\neg$`isCrawler` $\wedge$ $\neg$`hasShield` $\rightarrow$ `Bird`. Such a rule would be hard to learn because it is so complex.

A second problem is that in case a negation would be added that is not correct after all, the rule would be hard to correct. For example, suppose a penguin is observed and one of the agents does not recognize a penguin as a bird. The following rule would now be created: `laysEggs` $\wedge$ $\neg$`hasMammaryGlands` $\wedge$ $\neg$`isCrawler` $\wedge$ $\neg$`hasShield` $\wedge$ $\neg$`cannotFly` $\rightarrow$ `Bird`. This is clearly wrong, but it would be very hard to correct this rule. Much easier would it be if the agent knew the following rules: `laysEggs` $\rightarrow$ `Bird`, $\times$(`laysEggs` $\wedge$ `hasMammaryGlands`

→ Bird), ×(laysEggs ∧ isCrawler → Bird), ×(laysEggs ∧ hasShield → Bird), and ×(laysEggs ∧ cannotFly → Bird). In this case all rules would be correct except the last one. The situation is easily corrected by removing the last rule. Further, this would still allow the possibility to extend the original rule to laysEggs ∧ hasWings → Bird. This would make all other rules useless and is thus a very efficient solution to solve the problem with all exceptions for birds. However, such an efficient solution would be very hard to obtain if a rule that included a long list of negations would already exist. Therefore it can be concluded that using defeasible rules in combination with the exceptions such as proposed by (Verheij, 2003) is profitable for learning purposes. It could even be argued that such rules would be easier to use for agents because only positive properties would have to be considered. That is: when a blackbird is observed, the only rule that becomes active would be the rule laysEggs → Bird. All exceptions are not active because they contain properties that do not occur in the blackbird. If an agent would see a turtle, the rules laysEggs → Bird and ×(laysEggs ∧ hasShield → Bird) would become active. This quickly allows the agent to determine that the turtle is not a *Bird*. In other words, agents only have to check on positive properties in rules. If rules that included negations would be used, some kind of mechanism would be necessary to check negative properties. Using only positive rules with exceptions therefore allow a simpler mechanism to select rules.

Chapter 7

Conclusion

In this work a model was presented that tried to find an answer to the question whether defeasible rules and exceptions could provide a solid base for conceptualizing the environment in a population of agents. The model showed that this approach works. A high rate of success was achieved by the agents while they were playing the conceptualization games. In some cases they even achieved 100% agreement about the concepts. It was also shown that multiple concepts were created during the simulations, sometimes concepts that needed more complex rules or exceptions to be explained.

The agents used a quite high proportion of rules with multiple properties in the antecedent of the rule. Thereby it was shown that by using a carefully chosen mechanism for using and updating rules does not have to favour general rules. Always using the most specific rule that applies to a situation, allows more complex rules to be competitive in the competition between different hypotheses for the meaning of concepts.

As discussed there were also some features of the model that did not work as was hoped for. A lot was learnt though from creating and testing this model. Many difficulties that occur when creating a method for updating a set of defeasible rules with exceptions were encountered and some of these were solved, while for others new insights were developed that would allow the creation of an improved model.

When starting the work on this model, the intention was to keep it as simple as possible. Occam's razor should apply; the simpler the model, the more likely that it was the right way of approaching the problem. Unfortunately in the iterative development of the model, it also grew more and more complex. A next step should therefore be to look for more simplicity in the model. Many problems that have arisen during development and testing probably have better solutions than those that were applied in this work. With all the knowledge collected with this model, an improved model could be designed with more elegant properties. Such a model would probably ease tracking down possible new problems as well, because it would be simpler and thus easier to understand.

Future research should also aim at identifying proper ways to find a balance between creating general concepts while also being able to be specific. To fully exploit the emergence of exceptions, the environment in which the agents operate must be shaped in such a way that there will be reasons to use exceptions. Only using exceptions as a repair strategy, when agents encounter conflicting

situations is not enough to develop a set of stable exceptions in the population.

The iterative nature of learning rules that was described in this work lends itself perfectly for scaling up the challenges given to the agents. It would therefore be interesting to look at ways to learn simple general rules at first and making them increasingly complex by scaling up the challenge in the games the agents play. One of the more complex challenges could be introducing the restriction that in some occasions concepts must be mutually exclusive. This would force the use of exceptions. Which reasons give rise to this restriction and when the restriction applies should be well considered though. Otherwise initial knowledge might creep into the model.

Other possible ways of increasing the complexity of the task the agents face is introducing more objects. The set of objects used in the current work was limited and there was no time to investigate other, more complex object sets. The set of objects that is used probably has a big influence as well on the concepts that are created by agents. It would therefore be interesting to see how the model behaves when other object sets are used.

With this work the first steps into research on the emergence of defeasible rules and exceptions grounded in the environment are taken. An existing approach to use defeasible rules with exceptions in an multi-agent model was not found. The novelty in this work therefore is that it was shown that defeasible rules and exceptions can successfully emerge from interactions between agents about the environment. It was also shown that the presented mechanism allows complex rules to emerge. This paves the way for the emergence of complex, more specific concepts for which complex rules would be needed. If proper reasons to use exceptions or complex rules could be found, and these reasons can be translated to pressures in the conceptualization game, it is likely that the model would be able to fully exploit the richness of the defeasible rules with exceptions.

References

Brighton, H., & Kirby, S. (2001). The survival of the smallest: stability conditions for the cultural evolution of compositional language. In J. Kelemen & P. Sosik (Eds.), *Advances in artificial life* (p. 592-601). Springer-Verlag.

Governatori, G., & Stranieri, A. (2001). Towards the application of association rules for defeasible rules discovery. In B. Verheij, A. R. Lodder, R. P. Loui, & A. J. Muntjewerff (Eds.), *Legal knowledge and information systems* (p. 63-75). Amsterdam: IOS Press.

Grefenstette, J. J. (1992). The evolution of strategies for multi-agent environments. *Adaptive Behavior*, *1*, 65-89.

Grefenstette, J. J., Ramsey, C. L., & Schultz, A. C. (1990). Learning sequential decision rules using simulation models and competition. *Machine Learning*, *5*, 355-381.

Johnston, B., & Governatori, G. (2003). Induction of defeasible logic theories in the legal domain. In *Icail '03: Proceedings of the 9th international conference on artificial intelligence and law* (pp. 204–13). New York, NY, USA: ACM.

Kirby, S. (2001). Spontaneous evolution of linguistic structure - an iterated learning model of the emergence of regularity and irregularity. *IEEE Journal of Evolutionary Computation*, *5*(2), 102-110.

Kirby, S. (2002). Natural language from artificial life. *Artificial Life*, *8*(2), 185-212.

Puglisi, A., Baronchelli, A., & Loreto, V. (2008). Cultural route to the emergence of linguistic categories. *Proceedings of the National Academy of Sciences*, *105*(23), 7936-7940.

Skyrms, B. (2004). Evolution of inference. In *The stag hunt and the evolution of social structure* (p. 49-63). Cambridge University Press.

Steels, L. (1998). The origins of syntax in visually grounded robotic agents. *Artificial Intelligence*, *103*, 133-156.

Steels, L. (1999). *The talking heads experiment, volume i. words and meanings (pre-edition).*

Steels, L. (2004). The autotelic principle. In F. Iida, R. Pfeifer, L. Steels, & Y. Kuniyoshi (Eds.), *Embodied artificial intelligence* (Vol. 3139, p. 231-242). Springer-Verlag.

Steels, L., & Hanappe, P. (2006). Interoperability through emergent semantics, a semiotic dynamics approach. *Journal on Data Semantics VI*, *4090*, 143-167.

Steels, L., & Kaplan, F. (1999). Collective learning and semiotic dynamics. In D. Floreano, J. Nicoud, & F. Mondada (Eds.), *Advances in artificial life, proceedings* (Vol. 1674, p. 679-688). Springer-Verlag.

Steels, L., Trijp, R. van, & Wellens, P. (2007). Multi-level selection in the emergence of langauge systematicity. In F. Costa, L. Rocha, E. Costa, I. Harvey, & A. Coutinho (Eds.), *Advances in artificial life, proceedings* (Vol. 4648, p. 425-434). Springer-Verlag.

Steels, L., & Wellens, P. (2007). Scaffolding language emergence using the autotelic principle. In *Ieee symposium on artificial life* (p. 325-332).

Verheij, B. (2003). Deflog: On the logical interpretation of prima facie justified assumptions. *Journal of Logic and Computation*, *13*(3), 319-346.

Wellens, P., Loetzsch, M., & Steels, L. (2008). Flexible word meaning in embodied agents. *Connection Science*, *20*(2-3), 1-18.

Appendix A

List of objects

The list of objects is given in two formats. In section A.1 the objects are listed in the same format as they are shown in the output of games. In section A.2 the same objects are listed as they are shown in the XML input file.

A.1 Objects as shown in games

```
platypus         = [eatsAnimals, hasBeak, hasMammaryGlands,
                    hasSpinalColumn, hasWebbedFeet, isVenomous, laysEggs]

cow              = [eatsPlants, hasFourStomachs, hasMammaryGlands,
                    hasSpinalColumn, isRuminant]

dog              = [eatsAnimals, eatsPlants, hasMammaryGlands,
                    hasSpinalColumn]

evening bat      = [canFly, doesHibernate, eatsAnimals, hasMammaryGlands,
                    hasSpinalColumn, hasWings]

chimpanzee       = [eatsAnimals, eatsPlants, hasMammaryGlands,
                    hasSpinalColumn, livesInGroup]

african elephant = [eatsPlants, hasMammaryGlands, hasSpinalColumn,
                    hasTrunk]

salmon           = [canSwim, eatsAnimals, hasFins, hasGills,
                    hasSpinalColumn, inFreshWater, inSeaWater, laysEggs]

goldfish         = [canSwim, eatsAnimals, eatsPlants, hasFins, hasGills,
                    hasGoldColor, hasSpinalColumn, inFreshWater, laysEggs]

fin whale        = [canSing, canSwim, eatsAnimals, hasBaleens, hasFins,
                    hasMammaryGlands, hasSpinalColumn, inSeaWater]

octopus          = [canSwim, expelsInk, hasGills, inSeaWater, isVenomous,
                    laysEggs]

blackbird        = [canFly, canSing, eatsAnimals, eatsPlants, hasBeak,
```

```
                    hasBlackColour, hasSpinalColumn, hasWings, laysEggs]

common buzzard   = [canFly, eatsAnimals, hasBeak, hasSpinalColumn,
                    hasWings, laysEggs]

barn owl         = [canFly, eatsAnimals, hasBeak, hasSpinalColumn,
                    hasWings, huntsAtNight, laysEggs]

mute swan        = [canFly, canSwim, eatsPlants, hasBeak, hasSpinalColumn,
                    hasWebbedFeet, hasWhiteColour, hasWings, laysEggs]

black swan       = [canFly, canSwim, eatsPlants, hasBeak, hasBlackColour,
                    hasSpinalColumn, hasWebbedFeet, hasWings, laysEggs]

penguin          = [canSwim, eatsAnimals, hasBeak, hasSpinalColumn,
                    hasWings, laysEggs]
```

A.2 Objects as specified in XML file

```
<object frequency="1.0" identifier="platypus">
    <property>hasMammaryGlands</property>
    <property>hasSpinalColumn</property>
    <property>hasBeak</property>
    <property>laysEggs</property>
    <property>isVenomous</property>
    <property>hasWebbedFeet</property>
    <property>eatsAnimals</property>
</object>

<object frequency="10.0" identifier="cow">
    <property>hasMammaryGlands</property>
    <property>hasSpinalColumn</property>
    <property>hasFourStomachs</property>
    <property>isRuminant</property>
    <property>eatsPlants</property>
</object>

<object frequency="10.0" identifier="dog">
    <property>hasMammaryGlands</property>
    <property>hasSpinalColumn</property>
    <property>eatsPlants</property>
    <property>eatsAnimals</property>
</object>

<object frequency="2.5" identifier="evening bat">
    <property>hasMammaryGlands</property>
    <property>hasSpinalColumn</property>
    <property>eatsAnimals</property>
    <property>hasWings</property>
    <property>canFly</property>
    <property>doesHibernate</property>
</object>

<object frequency="2.5" identifier="chimpanzee">
    <property>hasMammaryGlands</property>
    <property>hasSpinalColumn</property>
    <property>livesInGroup</property>
    <property>eatsPlants</property>
    <property>eatsAnimals</property>
</object>

<object frequency="2.5" identifier="african elephant">
    <property>hasMammaryGlands</property>
    <property>hasSpinalColumn</property>
    <property>hasTrunk</property>
    <property>eatsPlants</property>
</object>
```

```
<object frequency="10.0" identifier="salmon">
    <property>canSwim</property>
    <property>hasFins</property>
    <property>laysEggs</property>
    <property>hasGills</property>
    <property>hasSpinalColumn</property>
    <property>eatsAnimals</property>
    <property>inFreshWater</property>
    <property>inSeaWater</property>
</object>

<object frequency="5.0" identifier="goldfish">
    <property>canSwim</property>
    <property>hasFins</property>
    <property>laysEggs</property>
    <property>hasGills</property>
    <property>hasSpinalColumn</property>
    <property>inFreshWater</property>
    <property>eatsAnimals</property>
    <property>eatsPlants</property>
    <property>hasGoldColor</property>
</object>

<object frequency="1.0" identifier="fin whale">
    <property>canSwim</property>
    <property>hasFins</property>
    <property>hasSpinalColumn</property>
    <property>hasMammaryGlands</property>
    <property>hasBaleens</property>
    <property>eatsAnimals</property>
    <property>inSeaWater</property>
    <property>canSing</property>
</object>

<object frequency="1.0" identifier="octopus">
    <property>canSwim</property>
    <property>expelsInk</property>
    <property>isVenomous</property>
    <property>hasGills</property>
    <property>laysEggs</property>
    <property>inSeaWater</property>
</object>

<object frequency="10.0" identifier="blackbird">
    <property>canFly</property>
    <property>hasWings</property>
    <property>laysEggs</property>
    <property>hasBeak</property>
    <property>hasSpinalColumn</property>
    <property>hasBlackColour</property>
    <property>eatsPlants</property>
    <property>eatsAnimals</property>
    <property>canSing</property>
</object>
```

```
<object frequency="10.0" identifier="common buzzard">
    <property>canFly</property>
    <property>hasWings</property>
    <property>laysEggs</property>
    <property>hasBeak</property>
    <property>hasSpinalColumn</property>
    <property>eatsAnimals</property>
</object>

<object frequency="5.0" identifier="barn owl">
    <property>canFly</property>
    <property>hasWings</property>
    <property>laysEggs</property>
    <property>hasBeak</property>
    <property>hasSpinalColumn</property>
    <property>eatsAnimals</property>
    <property>huntsAtNight</property>
</object>

<object frequency="10.0" identifier="mute swan">
    <property>canFly</property>
    <property>hasWings</property>
    <property>laysEggs</property>
    <property>hasBeak</property>
    <property>hasSpinalColumn</property>
    <property>eatsPlants</property>
    <property>hasWebbedFeet</property>
    <property>hasWhiteColour</property>
    <property>canSwim</property>
</object>

<object frequency="1.0" identifier="black swan">
    <property>canFly</property>
    <property>hasWings</property>
    <property>laysEggs</property>
    <property>hasBeak</property>
    <property>hasSpinalColumn</property>
    <property>eatsPlants</property>
    <property>hasWebbedFeet</property>
    <property>hasBlackColour</property>
    <property>canSwim</property>
</object>

<object frequency="2.5" identifier="penguin">
    <property>hasWings</property>
    <property>laysEggs</property>
    <property>hasBeak</property>
    <property>hasSpinalColumn</property>
    <property>eatsAnimals</property>
    <property>canSwim</property>
</object>
```