Natural Language and Inference in a Computer Game

Malte Gabsdil and **Alexander Koller** and **Kristina Striegnitz**
Dept. of Computational Linguistics
Saarland University, Saarbrücken, Germany
{gabsdil|koller|kris}@coli.uni-sb.de

Abstract

We present an engine for text adventures – computer games with which the player interacts using natural language. The system employs current methods from computational linguistics and an efficient inference system for description logic to make the interaction more natural. The inference system is especially useful in the linguistic modules dealing with reference resolution and generation and we show how we use it to rank different readings in the case of referential and syntactic ambiguities. It turns out that the player's utterances are naturally restricted in the game scenario, which simplifies the language processing task.

1 Introduction

Text adventures are computer games with which the player interacts via a natural language dialogue. Texts describe the game world and how it evolves, and the player can manipulate objects in this game world by typing in commands; Fig. 1 shows a sample interaction. Text adventures were very popular and commercially successful in the eighties, but have gone out of fashion since then – mostly because the parsers were rather limited and forced the user into very restricted forms of interaction.

We describe an engine for text adventures that attempts to overcome these limitations by using current methods from computational linguistics for processing the natural language input and output, and a state-of-the-art inference system based on description logic (DL) to represent the dynamic state of the game world and what the player knows about it. The DL prover is used in all language-processing modules except for parsing and surface realization, and supports the inferences we need very well.

This shows in particular in the modules for the resolution and generation of referring expressions. By keeping track of the true state of the world and the player's knowledge in separate knowledge bases, we can evaluate definite descriptions with respect to what the player knows. In generation, such inferences allow us to produce smaller while still sufficiently informative references.

Another interesting aspect which we discuss in this paper is the treatment of syntactic and referential ambiguities that come up in understanding input sentences. Here, too, the player knowledge restricts the way in which the input should be interpreted and guides the resolution process. We use inferences about the player knowledge to rule out inconsistent analyses and pragmatic heuristics to possibly select the preferred one.

Players of a text adventure are effectively situated in a game world and have to accomplish a specific task, which severely restricts the utterances they will naturally produce. For example, they will typically only refer to objects they could "see" in the simulated world. This simplifies the language processing tasks tremendously. The scenario of the game can be freely specified and tailored to different applications and levels of complexity. Along with the modularity of our implementation, this makes our system an interesting testbed for evaluating more sophisticated NLP modules, which also have access to the inference infrastructure. The same reason makes it useful as a teaching tool; in fact, our implementation was created as an advanced project for students.

Plan of the paper. We will start with an overview over the architecture of our system in Section 2. Section 3 will give a brief introduction to description logics, and then describes the knowledge bases we use. We will discuss how they interact with the linguistic modules in Section 4, which focuses on the resolution and generation of referring expressions. Section 5 deals with the resolution of ambiguities in the player input. Finally, Section 6 concludes and points to further research.

2 Architecture

The general architecture of the game engine, shown in Fig. 2, consists of a number of language-processing modules (drawn as ellipses), which in-

```
Observation Lounge
This is where the station staff and visitors come to relax. There are a lot of tables
and chairs here, a large observation window, and a plush carpet. In the corner you can
see an AstroCola dispenser. A tube leads up to the station's main corridor.

> put my galakmid coin into the dispenser
Click.
The dispenser display now reads "Credit = 1.00".

> push diet astrocola button
You hear a rumbling noise in the dispenser, but nothing appears in the tray.

> kick dispenser
A can drops into the tray. Amazing! The oldest trick in the book, and it
actually worked.
```

Figure 1: An example interaction with a text adventure, taken from (Ledgard, 1999).

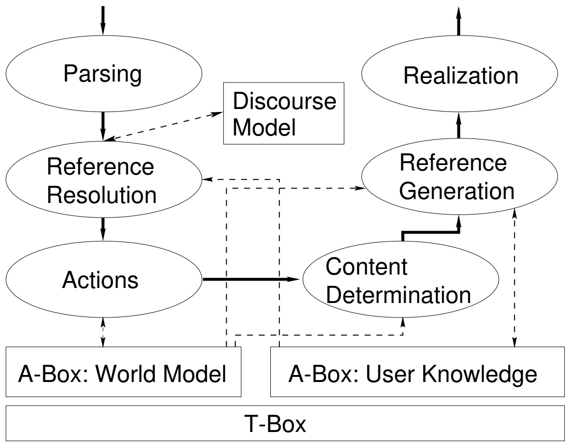

Figure 2: The architecture.

terface with knowledge bases and a discourse model (drawn as rectangles). There are two separate knowledge bases, which share a set of common definitions: One represents the true state of the world in a world model, the other keeps track of what the player knows about the world. Solid arrows indicate the general flow of information, dashed arrows indicate access to the knowledge bases.

The user's input is first parsed using an efficient parser for dependency grammar (Duchier and Debusmann, 2001). Next, referring expressions are resolved to individuals in the game world. The result is a ground term or a sequence of ground terms that indicates the action(s) the user wants to take. The Actions module looks up these actions in a database (where they are specified in a STRIPS-like format), checks whether the action's preconditions are met in the world, and, if yes, updates the world state with the effects of the action.

The action can also specify effects on the user's knowledge. This information is further enriched by the Content Determination module; for example, this module computes detailed descriptions of objects the player wants to look at. The Reference Generation module translates the internal names of individuals into descriptions that can be verbalized. In the last step, an efficient realization module (Koller and Striegnitz, 2002) builds the output sentences according to a TAG grammar. The player knowledge is updated after Reference Generation when the content of the game's response, including the new information carried e.g. by indefinite NPs, is fully established.

If an error occurs at any stage, e.g. because a precondition of the action fails, an error message specifying the reasons for the failure is generated by using the normal generation track (Content Determination, Reference Generation, Realization) of the game.

The system is implemented in the programming language Mozart (Mozart Consortium, 1999) and provides an interface to the DL reasoning system RACER (Haarslev and Möller, 2001), which is used for mainting and accessing the knowledge bases.

3 The World Model

Now we will look at the way that the state of the world is represented in the game, which will be important in the language processing modules described in Sections 4 and 5. We will first give a short overview of description logic (DL) and the theorem prover we use and then discuss some aspects of the world model in more detail.

3.1 Description Logic

Description logic (DL) is a family of logics in the tradition of knowledge representation formalisms such as KL-ONE (Woods and Schmolze, 1992). DL is a fragment of first-order logic which only allows unary and binary predicates (*concepts* and *roles*) and only very restricted quantification. A knowledge base consists of a *T-Box*, which contains axioms relating the concepts and roles, and one or more *A-Boxes*, which state that individuals belong to certain concepts, or are related by certain roles.

Theorem provers for description logics support a range of different reasoning tasks. Among the most common are *consistency* checking, *subsumption* checking, and *instance* and *relation* checking. Consistency checks decide whether a combination of T-Box and A-Box can be satisfied by some model, subsumption is to decide of two concepts whether all individuals that belong to one concept must necessarily belong to another, and instance and relation checking test whether an individual belongs to a certain concept and whether a certain relation holds between a pair of individuals, respectively. In addition to these basic reasoning tasks, description logic systems usually also provide some *retrieval functionality* which e.g. allows to compute all concepts that a given individual belongs to or all individuals that belong to a given concept.

There is a wide range of different description logics today which add different extensions to a common core. Of course, the more expressive these extensions become, the more complex the reasoning problems are. "Traditional" DL systems have concentrated on very weak logics with simple reasoning tasks. In the last few years, however, new systems such as FaCT (Horrocks et al., 1999) and RACER (Haarslev and Möller, 2001) have shown that it is possible to achieve surprisingly good average-case performance for very expressive (but still decidable) logics. In this paper, we employ the RACER system, mainly because it allows for A-Box inferences.

3.2 The World Model

The T-Box we use in the game specifies the concepts and roles in the world and defines some useful complex concepts, e.g. the concept of all objects the player can see. This T-Box is shared by two different A-Boxes representing the state of the world and what the player knows about it respectively.

The player A-Box will typically be a sub-part of the game A-Box because the player will not have explored the world completely and will therefore not have encountered all individuals or know about all of their properties. Sometimes, however, it may also be useful to deliberately hide effects of an action from the user, e.g. if pushing a button has an effect in a room that the player cannot see. In this case, the player A-Box can contain information that is inconsistent with the world A-Box.

A fragment of the A-Box describing the state of the world is shown in Fig. 3; Fig. 4 gives a graphical representation. The T-Box specifies that the world is partitioned into three parts: rooms, objects, and players. The individual 'myself' is the only instance that we ever define of the concept 'player'. Individuals are connected to their locations (i.e. rooms, container objects, or players) via the 'has-location' role; the A-Box also specifies what kind of object an individual is (e.g. 'apple') and what properties it has ('red'). The T-Box then contains axioms such as 'apple $\sqsubseteq$ object', 'red $\sqsubseteq$ colour', etc., which establish a taxonomy among concepts.

These definitions allow us to add axioms to the T-Box which define more complex concepts. One is the concept 'here', which contains the room in which the player currently is – that is, every individual which can be reached over a 'has-location' role from a player object.

$$\text{here} \doteq \exists \text{has-location}^{-1}.\text{player}$$

In this definition, 'has-location^{-1}' is the *inverse role* of the role 'has-location', i.e. it links a and b iff 'has-location' links b and a. Inverse roles are one of the constructions available in more expressive description logics. The quantification builds a more complex concept from a concept and a role: $\exists R.C$ is the concept containing all individuals which are linked via an R role to some individual in C. In the example in Fig. 3, 'here' denotes the singleton set $\{\text{kitchen}\}$.

Another useful concept is 'accessible', which contains all individuals which the player can manipulate.

$$\begin{aligned}\text{accessible} \;\doteq\; & \forall\text{has-location.here} \sqcup \\ & \forall\text{has-location.}(\text{accessible} \sqcap \text{open})\end{aligned}$$

All objects in the same room as the player are accessible; if such an object is an open container, its contents are also accessible. The T-Box contains axioms that express that some concepts (e.g. 'table', 'bowl', and 'player') contain only 'open'

room(kitchen)	player(myself)
table(t1)	apple(a1)
apple(a2)	worm(w1)
red(a1)	green(a2)
bowl(b1)	bowl(b2)
has-location(t1, kitchen)	has-location(b1, t1)
has-location(b2, kitchen)	has-location(a1, b2)
has-location(a2, kitchen)	has-detail(a2,w1)
has-location(myself, kitchen)	...

Figure 3: A fragment of a world A-Box.

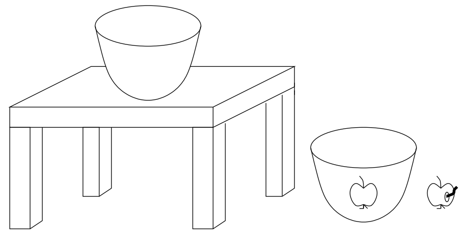

Figure 4: Example Scenario

objects. This permits access to the player's inventory. In the simple scenario above, 'accessible' denotes the set {myself, t1, a1, a2, b1, b2}. Finally, we can define the concept 'visible' in a similar way as 'accessible'. The definition is a bit more complex, including more individuals, and is intended to denote all individuals that the player can "see" from his position in the game world.[1]

4 Referring Expressions

The interaction between the game and the player revolves around performing actions on objects in the game world and the effects that these actions have on the objects. This means that the resolution and generation of referring expressions, which identify those objects to the user, are central tasks in our application.

Our implementation illustrates how useful the availability of an inference system as provided by RACER to access the world model is, once such an infrastructure is available. The inference engine is complemented by a simple discourse model, which keeps track of available referents.

4.1 The Discourse Model

Our discourse model (DM) is based on Strube's (1998) salience list approach, due to its simplicity. The DM is a data structure that stores an ordered list of the most salient discourse entities according to their "information status" and text position and provides methods for retrieving and inserting elements. Following Strube, *hearer-old* discourse entities (which include definites) are ranked higher in the DM (i.e. are more available for reference) than *hearer-new* discourse entities (including indefinites). Within these categories, elements are sorted according to their position in the currently processed sentence. For example, the ranking of discourse entities for the sentence *take a banana, the red apple, and the green apple* would look as follows:

$$[red\ apple \prec green\ apple]_{old} \prec [banana]_{new}$$

The DM is built incrementally and updated after each input sentence. Updating removes all discourse entities from the DM which are not realized in the current utterance. That is, there is an assumption that referents mentioned in the previous utterance are much more salient than older ones.

4.2 Resolving Referring Expressions

The task of the resolution module is to map definite and indefinite noun phrases and pronouns to individuals in the world. This task is simplified in the adventure setting by the fact that the communication is situated in a sense: Players will typically only refer to objects which they can "see" in the virtual environment, as modeled by the concept 'visible' above. Furthermore, they should not refer to objects they haven't seen yet. Hence, we perform all RACER queries in this section on the player knowledge A-Box, avoiding unintended ambiguities when the player's expression would e.g. not refer uniquely with respect to the true state of the world.

The resolution of a **definite** description means to find a unique entity which, according to the player's knowledge, is visible and matches the description. To compute such an entity, we construct a DL concept expression corresponding to the description and then send a query to RACER asking for all instances of this concept. In the case of *the apple*, for instance, we would retrieve all instances of the

[1] Remember that "seeing" in our application does not involve any graphical representations. The player acquires knowledges about the world only through the textual output generated by the game engine. This allows us to simplify the DL modeling of the world because we don't have to specify all (e.g. spatial) relations that would implicitly be present in a picture.

concept

$$\text{apple} \sqcap \text{visible}$$

from the player A-Box. The query concept for *the apple with the worm* would be

$$\text{apple} \sqcap (\exists\text{has-detail.worm}) \sqcap \text{visible}.$$

If this yields only one entity ($\{a2\}$ for *the apple with the worm* for the A-Box in Fig. 3), the reference has been unambiguous and we are done. It may, however, also be the case that more than one entity is returned; e.g. the query for *the apple* would return the set $\{a1,a2\}$. We will show in the next section how we deal with this kind of ambiguity. We reject input sentences with an error message indicating a failed reference if we cannot resolve an expression at all, i.e. when no object in the player knowledge matches the description.

We resolve **indefinite** NPs, such as *an apple*, by querying the player knowledge in the same way as described above for definites. Unlike in the definite case, however, we do not require unique reference. Instead, we assume that the player did not have a particular object in mind and arbitrarily choose one of the possible referents. The reply of the game will automatically inform the player which one was chosen, as a unique definite reference will be generated (see below).

Pronouns are simply resolved to the most salient entity in the DM that matches their agreement constraints. The restrictions our grammar imposes on the player input (no embeddings, no reflexive pronouns) allow us to analyze sentences including intra-sentential anaphora like *take the apple and eat it*. The incremental construction of the DM ensures that by the time we encounter the pronoun *it*, *the apple* has already been processed and can serve as a possible antecedent.

4.3 Generating Referring Expressions

The converse task occurs when we generate the feedback to show to the player: It is necessary to construct descriptions of individuals in the game world that enable the player to identify these.

This task is quite simple for objects which are new to the player. In this case, we generate an indefinite NP containing the type and (if it has one) color of the object, as in *the bowl contains a red apple*. We use RACER's retrieval functionality to extract this information from the knowledge base.

To refer to an object that the player already has encountered, we try to construct a definite description that, given the player knowledge, uniquely identifies this object. For this purpose we use a variant of Dale and Reiter's (1995) incremental algorithm, extended to deal with relations between objects (Dale and Haddock, 1991). The properties of the target referent are looked at in some predefined order (e.g. first its type, then its color, its location, parts it may have, ...). A property is added to the description if at least one other object (a distractor) is excluded from it because it doesn't share this property. This is done until the description uniquely identifies the target referent.

The algorithm uses RACER's reasoning and retrieval functionality to access the relevant information about the context, which included e.g. computing the properties of the target referent and finding the distracting instances. Assuming we want to refer to entity a1 in the A-Box in Fig. 3 e.g., we first have to retrieve all concepts and roles of a1 from the player A-Box. This gives us {apple(a1), red(a1), has-location(a1,b1)}. As we have to have at least one property specifying the type of a1, we use RACER's subsumption checks to extract all those properties that match this requirement; in this case, 'apple'. Then we retrieve all instances of the concept 'apple' to determine the set of distractors which is $\{a1, a2\}$. Hence, 'apple' alone is not enough to uniquely identify a1. So, we consider the apple's color. Again using subsumption checks, we filter the colors from the properties of a1 (i.e. 'red') and then retrieve all instances belonging to the concept $\text{apple} \sqcap \text{red}$ to check whether and how the set of distractors gets reduced by adding this property. This concept has only one member in the example, so we generate the expression *the red apple*.

5 Ambiguity Resolution

The other aspect of the game engine which we want to highlight here is how we deal with referential and syntactic ambiguity. We handle the former by a combination of inference and discourse information, and the latter by taking psycholinguistically motivated preferences into account.

5.1 Resolving Referential Ambiguities

When the techniques for reference resolution described in the previous section are not able to map a definite description to a single entity in the player knowledge, the resolution module returns a set of possible referents. We then try to narrow this set down in two steps.

First, we filter out individuals which are completely unsalient according to the discourse model. In our (simplified) model, these are all individuals that haven't been mentioned in the previous sentence. This heuristic permits the game to deal with the following dialogue, as the red but not the green apple is still accessible in the final turn, and is therefore chosen as the patient of the 'eat' action.

Game:	... red apple ... green apple.
Player:	Take the red apple.
Game:	You have the red apple.
Player:	Eat *the apple*.
Game:	You eat the red apple.

If this narrows down the possible referents to just one, we are done. Otherwise – i.e. if several or none of the referents were mentioned in the previous sentence –, we check whether the player's knowledge rules out some of them. The rationale is that an intelligent player would not try to perform an action on an object on which she knows it cannot be performed.

Assume, by way of example, that the player knows about the worm in the green apple. This violates a precondition of the 'eat' action for apples. Thus if both apples were equally salient, we would read *eat the apple* as *eat the red apple*. We can test if a combination of referents for the various referring expressions of a sentence violates preconditions by first instantiating the appropriate action with these referents. Then we independently add each instantiated precondition to fresh copies of the player knowledge A-Box and test them for consistency. If one of the A-Boxes becomes inconsistent, we conclude that the player knows this precondition would fail, and conclude that this is not the intended combination of referents.

If neither of these heuristics manages to pick out a unique entity, we consider the definite description to be truly ambiguous and return an error message to the user, indicating the ambiguity.

5.2 Resolving Syntactic Ambiguities

Another class of ambiguities which we consider are syntactic ambiguities, especially of PP attachment. We try to resolve them, too, by taking referential information into account.

In the simplest case, the referring expressions in some of the syntactic readings have no possible referent in the player A-Box at all. If this happens, we filter these readings out and only continue with the others (Schuler, 2001). For example, the sentence *unlock the toolbox with the key* is ambiguous. In a scenario where there is a toolbox and a key, but the key is not attached to the toolbox, resolution fails for one of the analyses and thereby resolves the syntactic ambiguity.

If more than one syntactic reading survives this first test, we perform the same computations as above to filter out possible referents which are either unsalient or violate the player's knowledge. Sometimes, only one syntactic reading will have a referent in this narrower sense; in this case, we are done.

Otherwise, i.e. if more than one syntactic reading has referents, we remove those readings which are referentially ambiguous. Consider once more the example scenario depicted in Fig. 4. The sentence *put the apple in the bowl on the table* has two different syntactic analyses: In the first, *the bowl on the table* is the target of the put action whereas in the second, *in the bowl* modifies *the apple*. Now, note that in the first reading, we will get two possible referents for *the apple*, whereas in the second reading *the apple in the bowl* is unique. In cases like this we pick out the reading which only includes unique references (reading 2 in the present example). This approach assumes that the players are cooperative and try to refer unambiguously. It is furthermore similar to what people seem to do. Psycholinguistic eye-tracking studies (Chambers et al., 2000) indicate that people prefer interpretations with unambiguous references: subjects who are faced with scenarios similar to Fig. 4 and hear the sentence *put the apple in the bowl on the table* do not look at the bowl on the table at all but only at the apple in the bowl (which is unique) and the table.

At this point, there can still be more than one syntactic reading left; if so, all of these will have unambiguous, unique referents. In such a case we cannot decide which syntactic reading the player meant, and ask the player to give the game a less ambiguous command.

6 Conclusion and Outlook

We have described an engine for text adventures which uses techniques from computational linguistics to make the interaction with the game more natural. The input is analyzed using a dependency parser and a simple reference resolution module, and the output is produced by a small generation system. Information about the world and about the player's knowledge is represented in description logic knowledge bases, and accessed through a state-of-the-art inference system. Most modules use the inference component; to illustrate its usefulness, we have looked more closely at the resolution and generation of referring expressions, and at the resolution of referential and syntactic ambiguities.

Preliminary experiments indicate that the performance of our game engine is good enough for fluent gameplay. The constraint based dependency parser we use for parsing and generation achieves very good average case runtimes on the grammars and inputs we use. More interestingly, the inference system also performs very well. With the current knowledge bases, reasoning on the world model and user knowledge takes 546ms per turn on average (with a mean of 39 queries per turn). How well this performance scales to bigger game worlds remains to be seen. One lesson we take from this is that the recent progress in optimizing inference engines for expressive description logics is beginning to make them useful for applications.

All the language-processing modules in our system are rather simplistic. We can get away with this because the utterances that players seem to want to produce in this setting are restricted, e.g. to objects in the same simulated "location" as the player. (The precise extent of this, of course, remains to be evaluated.) The result is a system which exceeds traditional text adventures by far in the flexibility offered to the user.

Unlike the input, the output that our game generates is far away from the quality of the commercial text adventures of the eighties, which produced canned texts, sometimes written by professional book authors. A possible solution could be to combine the full generation with a template based approach, to which the TAG-based generation approach we take lends itself well. Another problem is the generation of error messages asking the user to resolve an ambiguous input. The game should ideally generate and present the player with a choice of possible (unambiguous) readings. So, the generation strategy would have to be augmented with some kind of monitoring, such as the one proposed by Neumann and van Noord (1994). Finally, we want to come up with a way of synchronizing the grammars for parsing and generation, in order to ensure that expressions used by the game can always be used by the player as well.

The system is designed in a way that should make it reasonably easy to replace our simple modules by more sophisticated ones. We will shortly make our adventure engine available over the web, and want to invite colleagues and students to test their own language processing modules within our system. Generally, we believe that the prototype can serve as a starting point for an almost unlimited range of extensions.

References

C.G. Chambers, M.K. Tanenhaus, and J.S. Magnuson. 2000. Does real-world knowledge modulate referential effects on PP-attachment? Evidence from eye movements in spoken language comprehension. In *14th CUNY Conference on Human Sentence Processing*.

R. Dale and N. Haddock. 1991. Generating referring expressions involving relations. In *EACL '91*.

R. Dale and E. Reiter. 1995. Computational interpretations of the gricean maxims in the generation of referring expressions. *Cognitive Science*, 18.

D. Duchier and R. Debusmann. 2001. Topological dependency trees: A constraint-based account of linear precedence. In *ACL '01*.

V. Haarslev and R. Möller. 2001. RACER System Description. In *IJCAR '01*.

I. Horrocks, U. Sattler, and S. Tobies. 1999. Practical reasoning for expressive description logics. In H. Ganzinger, D. McAllester, and A. Voronkov, editors, *LPAR'99*.

A. Koller and K. Striegnitz. 2002. Generation as dependency parsing. In *ACL '02*.

D. Ledgard. 1999. Space Station. Text adventure, modelled after a sample transcript of Infocom's *Planetfall* game. `http://members.tripod.com/~infoscripts/planetfa.htm`.

Mozart Consortium. 1999. The Mozart Programming System web pages. `http://www.mozart-oz.org/`.

G. Neumann and G.-J. van Noord. 1994. Self-monitoring with reversible grammars. In T. Strzalkowski, editor, *Reversible Grammar in Natural Language Processing*.

W. Schuler. 2001. Computational properties of environment-based disambiguation. In *ACL '01*.

M. Strube. 1998. Never Look Back: An Alternative to Centering. In *COLING-ACL '98*.

W. Woods and J. Schmolze. 1992. The KL-ONE Family. *Computer and Mathematics with Applications*, 23(2–5).