A Computer Game Based on Description Logic and Natural Language Processing

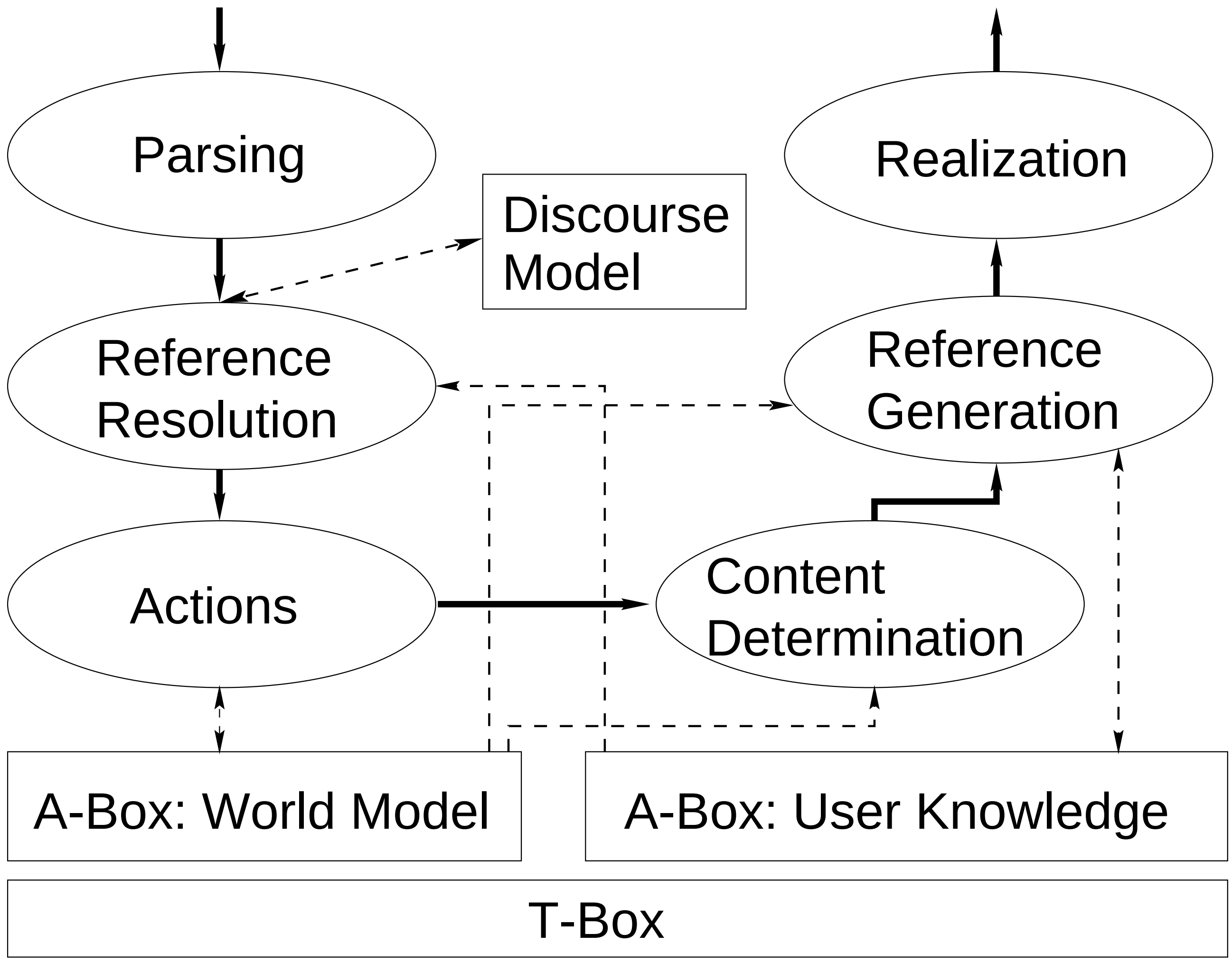
Malte Gabsdil, Alexander Koller, Kristina Striegnitz

Computational Linguistics

Saarland University, Germany

{gabsdil,koller,kris}@coli.uni-sb.de

Architecture of the Game Engine



Description Logic (DL) Crash Course

T(erminological)-Box:

$rabbit \sqsubseteq animal$

everything that's a rabbit is also an animal

$rabbit \sqsubseteq \exists has.tail$

everything that's a rabbit is related to a tail via the relation "has"

$rabbit \sqsubseteq \exists{=}1\ has.tail$

every rabbit is related to exactly one tail via the "has" relation

$rabbit \sqsubseteq animal \sqcap fluffy$

rabbis are things which are animals and fluffy

$\exists has.(ear \sqcap long) \sqsubseteq hare \sqcup rabbit$

everything that is related (via "has") to something which is long and an ear is either a rabbit or a hare

$rabbit \sqsubseteq \neg zebra$

if something is a rabbit, then it is not a zebra

A(ssertional)-Box:

$rabbit(bugs)$

bugs is a rabbit

$brother\_of(bugs, bunny)$

bugs and bunny are related to each other via the relation "brother_of"

DL Theorem Provers

DL theorem provers provide a range of different *inference services*:

- Does concept C_1 subsume C_2?
- Give me all (direct) ancestors/descendants of concept C.
- Is individual a an instance of concept C?
- Give me all instances of concept C.
- Give me all (most specific) concepts that instance a belongs to.
- Give me all individuals that a is related to via the relation R.
- $\ldots$

Modelling the Game World (1)

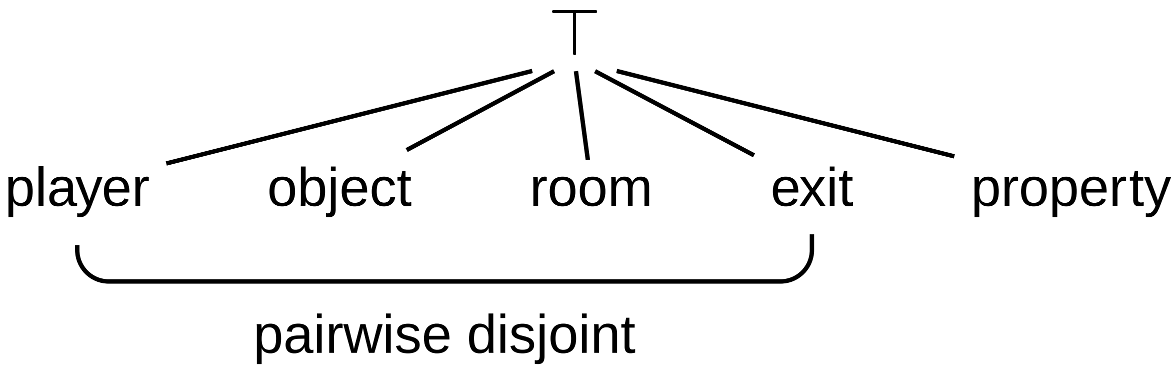
The T-Box defines a hierarchy of the concepts that we want to use in the game.

animal $\sqsubseteq$ *object*

frog $\sqsubseteq$ *animal*

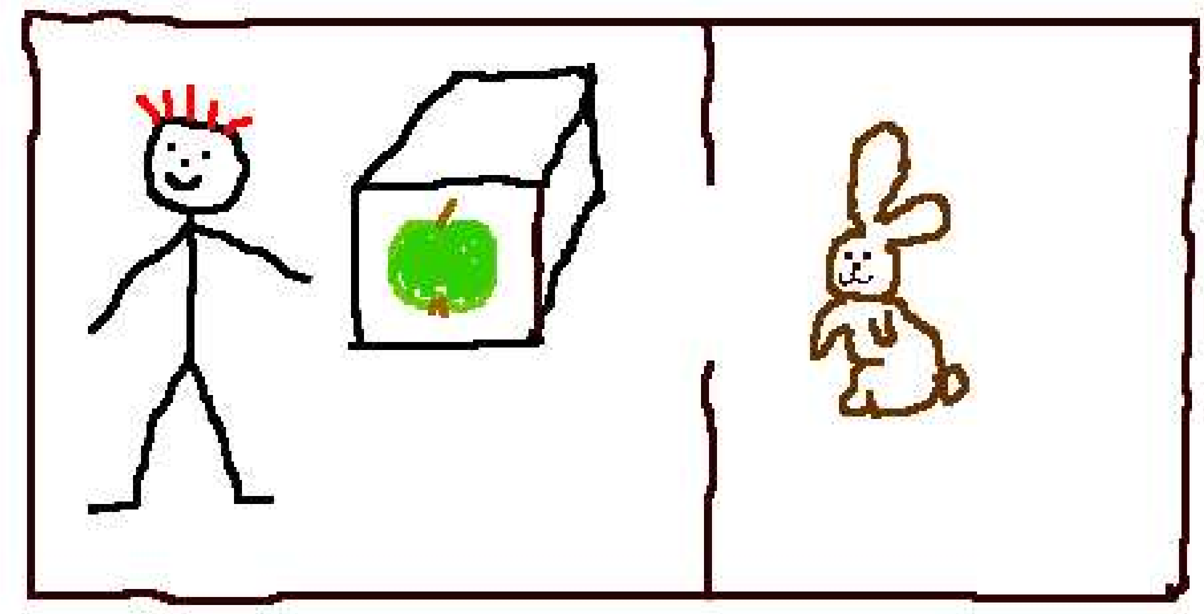
alive $\sqsubseteq$ *property*

...



Modelling the Game World (2)

The A-Box defines which objects/individuals exist in the game world and also specifies their properties.



rabbit(rabbit1)	*player(myself)*	*has-location(rabbit1,room2)*
box(box1)	*room(room1)*	*has-location(myself,room1)*
apple(apple1)	*room(room2)*	*has-location(box1,room1)*
green(apple1)		*has-location(apple1,box1)*

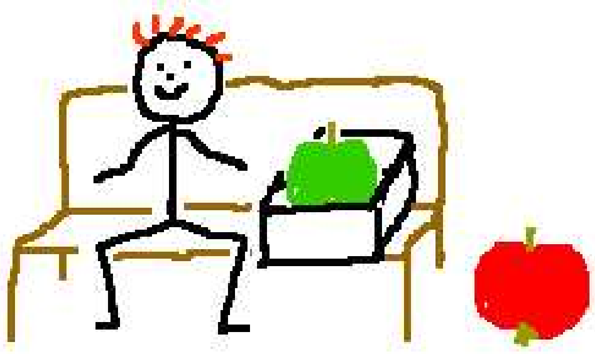
Modelling the Game World (3)

The T-Box furthermore defines some concepts that are important for restricting the actions of the player.

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

$$\textit{accessible} \doteq \textit{here} \sqcup \exists \textit{has-location}.\textit{here} \sqcup \exists \textit{has-location}.(\textit{accessible} \sqcap \textit{open})$$

$$\begin{aligned}\textit{visible} \doteq\ & \textit{here} \sqcup \exists \textit{has-location}.\textit{here} \sqcup \\ & \exists \textit{has-location}.(\textit{visible} \sqcap \textit{open}) \sqcup \\ & \exists \textit{has-location}^{-1}.(\textit{open} \sqcap \exists \textit{has-location}^{-1}.\textit{player}) \sqcup \\ & \exists \textit{has-location}.\exists \textit{has-location}^{-1}.(\textit{open} \sqcap \exists \textit{has-location}^{-1}.\textit{player})\end{aligned}$$



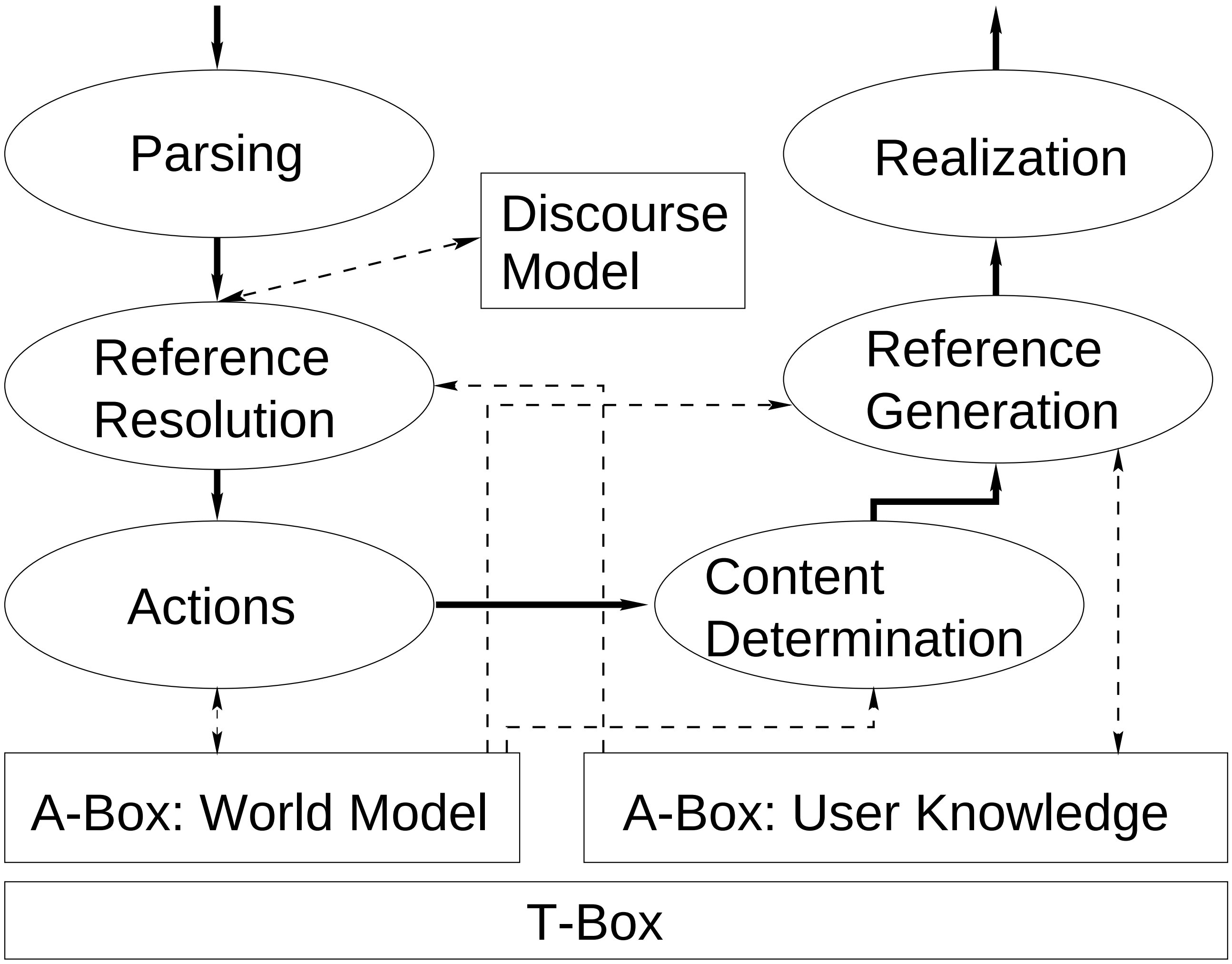
Modelling the User Knowledge

- Shares the general knowledge (T-Box) with the system.
- The A-Box reflects what the player knows about the game world.

Initial A-Box:

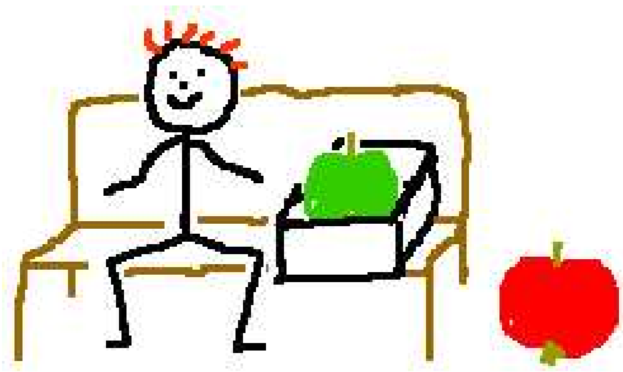
player(myself)

Architecture of the Game Engine



Acting in the Game World

“take the green apple”



apple(a)

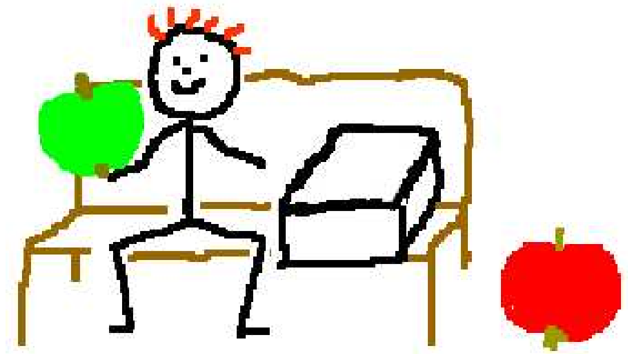
green(a)

box(b)

has-location(a,b)

. . .

→



apple(a)

green(a)

box(b)

has-location(a,myself)

. . .

Action Schemata

take(patient:X)	
preconditions	*accessible(X), takeable(X),* *not(inventory-object(X))*
effects	add: *related(X myself has-location)* delete: *related(X indiv-filler(X has-location) has-location)*

Action Schemata

take(patient:X)	
preconditions	*accessible(X), takeable(X),* *not(inventory-object(X))*
effects	add: *related(X myself has-location)* delete: *related(X indiv-filler(X has-location) has-location)*
user knowledge	add: *related(X myself has-location)* delete: *related(X indiv-filler(X has-location) has-location)*

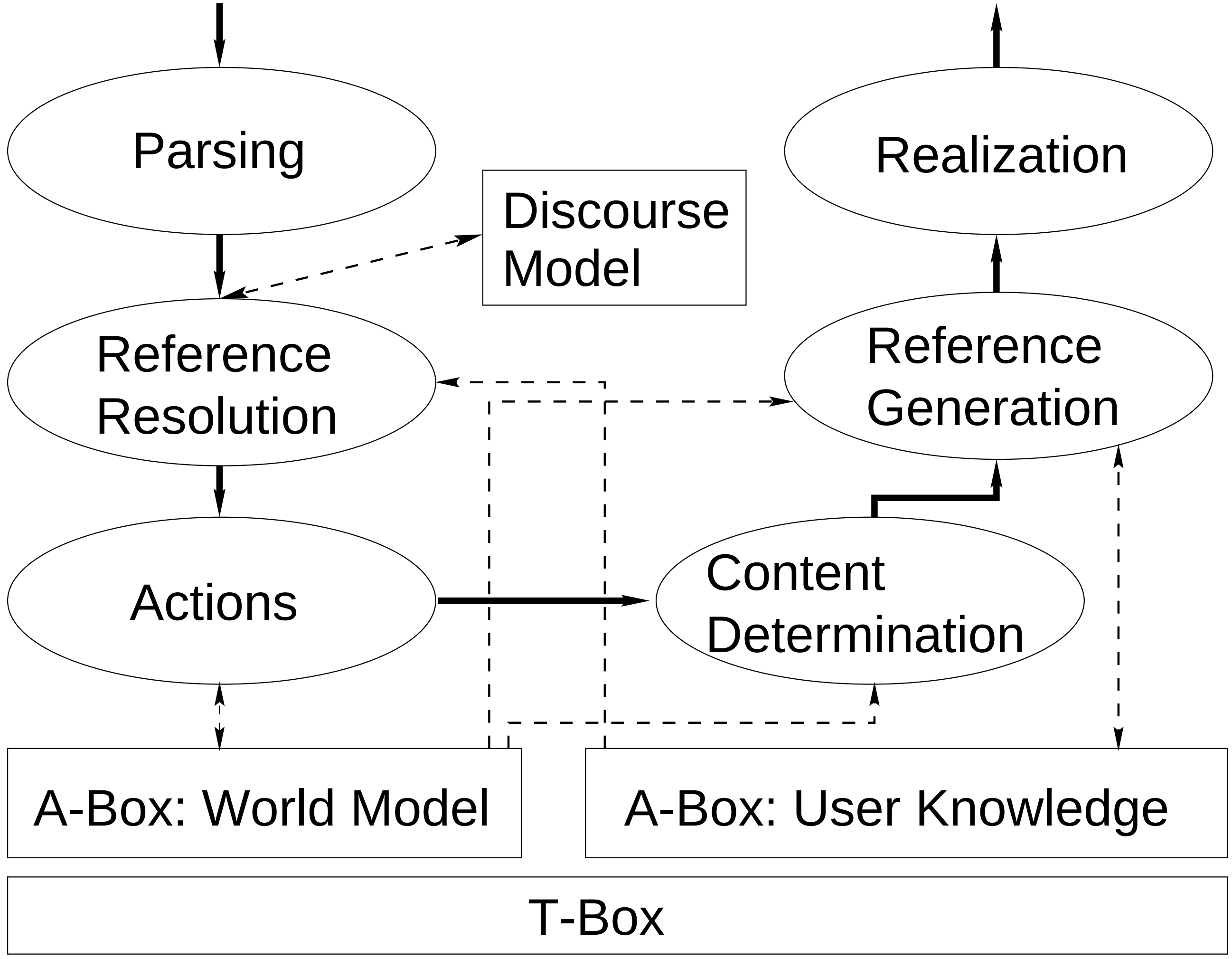
Action Schemata

output of the analysis of the user input

take(patient:X)		
preconditions	*accessible(X), takeable(X),* *not(inventory-object(X))*	
effects	add: *related(X myself has-location)* delete: *related(X indiv-filler(X has-location) has-location)*	
user knowledge	add: *related(X myself has-location)* delete: *related(X indiv-filler(X has-location) has-location)*	

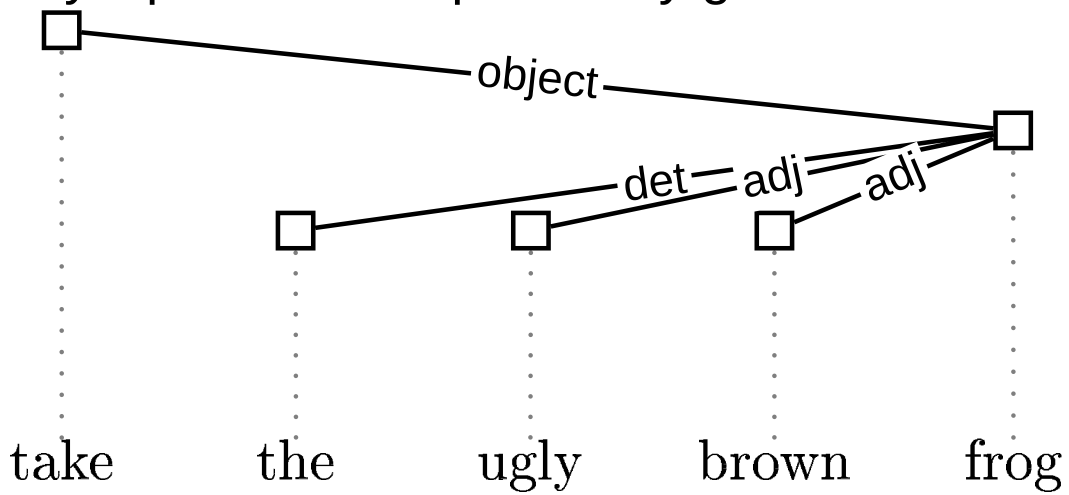
input for the generation of the responses

Architecture of the Game Engine



Parsing

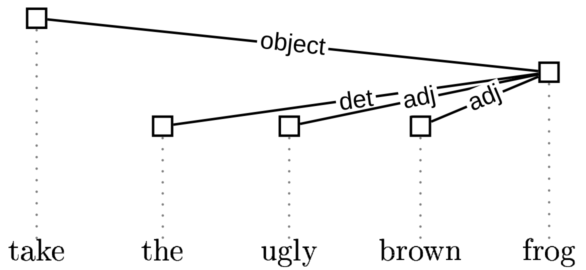
Ralph and Denys' parser for dependency grammar



$$take = \begin{bmatrix} \textsf{in} & : & \{\} \\ \textsf{out} & : & \{(\textsf{subj}, ?), (\textsf{obj}, !)\} \\ \textsf{sem} & : & \text{`take'} \\ \textsf{patient} & : & \{\textsf{object}\} \end{bmatrix}$$

$$frog = \begin{bmatrix} \textsf{in} & : & \{\textsf{subj}, \textsf{obj}\} \\ \textsf{out} & : & \{(\textsf{det}, ?), (\textsf{adj}, *)\} \\ \textsf{sem} & : & \text{`frog'} \\ \textsf{nmod} & : & \{\textsf{adj}, \textsf{pp}\} \end{bmatrix}$$

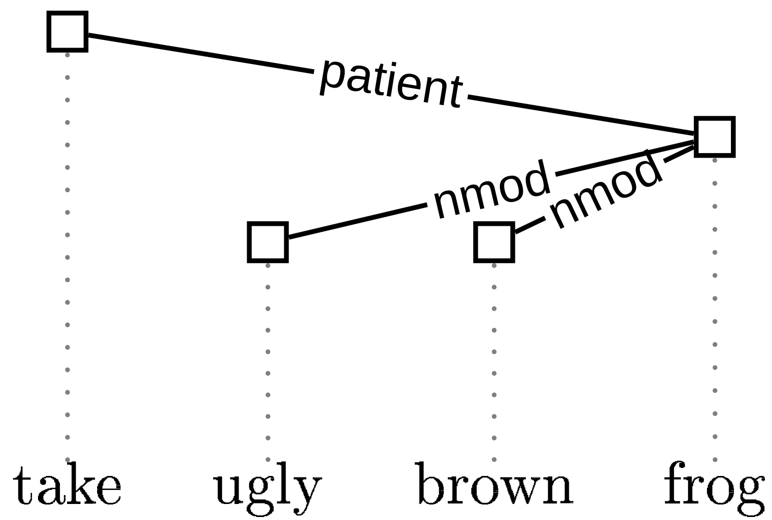
Semantic Construction



$$take = \begin{bmatrix} \textsf{in} & : & \{\} \\ \textsf{out} & : & \{(\textsf{subj}, ?), (\textsf{obj}, !)\} \\ \textsf{sem} & : & \text{`take'} \\ \textsf{patient} & : & \{\textsf{object}\} \end{bmatrix}$$

$$frog = \begin{bmatrix} \textsf{in} & : & \{\textsf{subj}, \textsf{obj}\} \\ \textsf{out} & : & \{(\textsf{det}, ?), (\textsf{adj}, *)\} \\ \textsf{sem} & : & \text{`frog'} \\ \textsf{nmod} & : & \{\textsf{adj}, \textsf{pp}\} \end{bmatrix}$$

$\Rightarrow$



$$frog = \begin{bmatrix} \textsf{agr} & : & \begin{bmatrix} \textsf{gender} & : & neut \\ \textsf{number} & : & sing \\ \textsf{spec} & : & def \end{bmatrix} \\ \textsf{pos} & : & 5 \end{bmatrix}$$

Reference Resolution

Replace object descriptions in the output of the parser by the internal name of the object that this description refers to.

```
take(patient: [frog(nmod:[ugly brown])] )
```

↓

```
take(patient: frog1 )
```

Construct a DL concept from the object description and retrieve all instances of this concept that are *visible*.

"the ugly brown frog": $visible \sqcap frog \sqcap ugly \sqcap brown$

"the frog with the crown": $visible \sqcap frog \sqcap \exists has\text{-}detail.crown$

Resolving Pronouns

- follows Strube's "Never Look Back", in Coling-ACL, 1998.
- discourse model (a list of entities) to keep track of salient entities
  - entities of current sentence are more salient than entities of previous sentence
  - entities introduces by a definite NP are more salient than entities introduced by an indefinite NP
  - entities earlier in the sentence are more salient than those metioned later

"take a banana, the red apple, and the green apple"

$$\Rightarrow red\ apple \prec green\ apple \prec banana$$

Content Determination

Input:

add: *related(X myself has-location)*
delete: *related(X indiv-fi ller(X has-location) has-location)*

Assumption: Verbalizing the "positive" effects is enough. The player can then infer the "negative" ones.

- standard case: pass on
- special treatment for some special keywords ($describe$, $disgusting$) and some actions (eat, $inventory$, $open$)

Detailed Descriptions of Objects

describe triggers detailed descriptions of individuals.

Two schemata for describing

1. an object
   - retrieve all most specific concepts
   - retrieve all role assertions
2. the location of the player:
   - retrieve all objects that are in the same location
   - retrieve all exits
   - if the location is an open container (e.g. a couch), do the same for room/location this container is in

Also does some aggregation.

Reference Generation

Input: $\left[contains\left(couch1, \left[frog1, frog2, apple1\right]\right)\right]$

Internal names of objects have to be described for the player.

Output: $\left[contains\left(couch1, \left[frog1, frog2, apple1\right]\right)\right.$

couch(chouch1), frog(frog1), frog(frog2), apple(apple1)

def(couch1), indef(frog1), indef(frog2), indef(appl1)]

- if the object is not an instance in the player A-Box, then use an indefinite NP
- if the object is an instance in the player A-Box, then use a definite NP

Indefinite NPs

- retrieve the object’s type (the most specific concept subsumed by the concept *object*) from the world model
- retrieve the object’s color (if it has one) from the world model

Definite NPs

- follows Dale & Reiter 1995 and Dale & Haddock 1991
- find a description that uniquely identifies the object to the player
- build a concept adding properties until the concept only has one instance, which is the target object

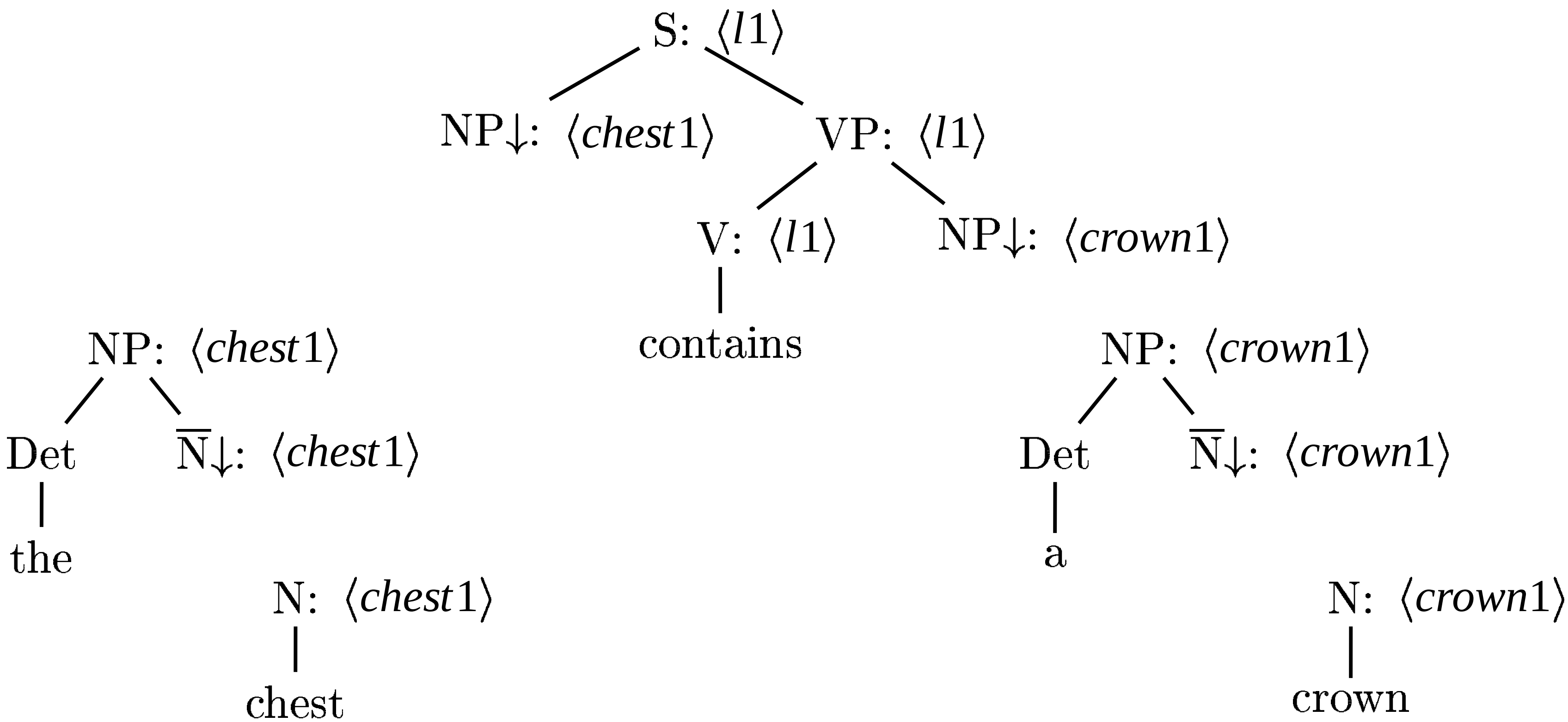
player A-Box:

apple(apple1),
frog(frog1), brown(frog1),
frog(frog2), green(frog1)

target: *apple1* $\Rightarrow$ *apple*

target: *frog1* $\Rightarrow$ $\mathit{frog} \sqcap \mathit{brown}$

Surface Realization



[*def*(*chest1*), *chest*(*chest1*), l1:*contains*(*chest1*,[*crown1*]), *indef*(*crown1*), *crown*(*crown1*)]

- Tree Adjoining Grammar where every elementary tree in the lexicon has exactly one piece of semantic information.
- Use Ralph and Denys' parser for selecting one elementary tree for each piece of semantic information and assembling the elementary trees into a sentence.

Architecture of the Game Engine