Theorem Provers
Outline

• Theorem Proving for Text Adventures
  – Introduction
  – Knowledge Base
  – System Architecture
  – Conclusions

• Theorem Provers
  – Racer
  – Prover9 / Mace4
  – Demos
Introduction

• Koller et al. (2004):
  – engine for playing **text adventures**:
    • type of computer game, popular in the 80s
    • player interacts with the game world by typing NL commands
    • system gives NL answers
  – theorem proving combined with parsing and NL generation
Text Adventure: Example

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.
Introduction

• classical text adventures:
  – high quality output (hard-coded)
  – poor parsing of user input

• “identification problem”:

  Cupboard
  When you aren’t lying on the bed, you usually stay in here, snug and safe with your friends atop the warm pile of clothes. Your warm winter jacket is here, which may be just as well, it’s a little chilly.

>take the warm winter jacket
You can’t see any such thing.

>take the winter jacket
You can’t see any such thing.
Knowledge Base

- theorem prover Racer
- knowledge base (description logic):
  - one **T-Box**:
    - specifies the concepts and roles in the world
  - two **A-Boxes**:
    - specify locations, types and properties of individuals
    - one to represent the state of the world
    - one to represent the player’s knowledge of the world
Knowledge Base

• T-Box:

  – specifies that the world is partitioned in three parts: rooms, objects, player

  – example axioms:

    \[
    \begin{align*}
    \text{apple} & \subseteq \text{object} \\
    \text{red} & \subseteq \text{colour}
    \end{align*}
    \]
    \[
    \rightarrow \text{taxonomy}
    \]

    \[
    \text{here} \defines \exists \text{has-location}^{-1} \cdot \text{player}
    \]

    \[
    \text{accessible} \defines \forall \text{has-location} . \text{here} \sqcup \forall \text{has-location} . (\text{accessible} \sqcap \text{open})
    \]
Knowledge Base

• A-Boxes:
  – usually: player A-Box subpart of world A-Box
  – sometimes: effects of an action are hidden from player → player A-Box inconsistent with world A-Box
  – example axioms:
    
    \[
    \begin{align*}
    \text{room(kitchen)} & \quad \text{player(myself)} \\
    \text{table(t1)} & \quad \text{apple(a1)} \\
    \text{apple(a2)} & \quad \text{worm(w1)} \\
    \text{red(a1)} & \quad \text{green(a2)} \\
    \text{bowl(b1)} & \quad \text{bowl(b2)} \\
    \text{has-location(t1, kitchen)} & \quad \text{has-location(b1, t1)} \\
    \text{has-location(b2, kitchen)} & \quad \text{has-location(a1, b2)} \\
    \text{has-location(a2, kitchen)} & \quad \text{has-detail(a2,w1)} \\
    \text{has-location(myself, kitchen)} & \quad \ldots
    \end{align*}
    \]
Architecture
Parsing Module

- parser for Topological Dependency Grammar (TDG)
- user input
  
  \[ \downarrow \]

- syntactic dependency tree

  
  \[ \downarrow \]

- semantic dependency tree
Parsing Module

• semantic construction:
  – go top-down through syntax tree
  – map syntactic to semantic roles
  – record information for NPs
    • agreement
    • linear position within the sentence
    • definiteness / indefiniteness

```
eat(patient:[apple(agr:[unit(gender:[neut]
                               number:[sing]
                               spec:[def]]]
                      nmod:[’big-sized’ red]
                      pos:[5]]))
```
Reference Resolution

• definite NPs:
  – construct DL concept expression corresponding to description
    e.g. \textit{the apple}: \texttt{apple} \sqcap \texttt{visible}
    \textit{the apple with the worm}:
    \texttt{apple} \sqcap (\exists \texttt{has-detail.worm}) \sqcap \texttt{visible}
  – send query to Racer asking for all instances of this concept in
    the player knowledge base
  – if query yields only one entity: entity = referent
  – if query yields more than one entity: filter out all entities which
    are unsalient according to \texttt{discourse model}
      • if one entity left: entity = referent
      • otherwise: assume description was not unique and return error
        message
Reference Resolution

• indefinite NPs:
  – e.g. *an apple*
  – query player knowledge base in the same way as for definite NPs
  – but: choose an arbitrary possible referent
Discourse Model

- data structure that stores the most salient discourse entities
  - “hearer-old” discourse entities (e.g. definite NPs) ranked higher in discourse model than “hearer-new” discourse entities (e.g. indefinite NPs)
  - within these categories, elements sorted according to their position in the sentence
  - e.g. *take a banana, the red apple, and the green apple:*
    \[ \text{[red apple} \prec \text{green apple]} \text{old} \prec \text{[banana]} \text{new} \]
  - build DM incrementally
  - update DM after each input sentence: remove all entities from DM that are not realized in the sentence
Actions

- output of reference resolution module:
  - list of lists of action descriptions,
    e.g. *Take the apple and eat it.*
    
    \[
    \text{[[take(patient:a2), eat(patient:a2)]]}
    \]
  - one entry for each reading of an ambiguous input sentence
  - database with action representations

<table>
<thead>
<tr>
<th>take(patient:X)</th>
</tr>
</thead>
<tbody>
<tr>
<td>preconditions</td>
</tr>
<tr>
<td>accessible(X), takeable(X),</td>
</tr>
<tr>
<td>not(inventory-object(X))</td>
</tr>
<tr>
<td>effects</td>
</tr>
<tr>
<td>add: related(X myself has-location)</td>
</tr>
<tr>
<td>delete: related(X indiv-filler(X has-location) has-location)</td>
</tr>
<tr>
<td>user</td>
</tr>
<tr>
<td>add: related(X myself has-location)</td>
</tr>
<tr>
<td>knowledge</td>
</tr>
<tr>
<td>delete: related(X indiv-filler(X has-location) has-location)</td>
</tr>
</tbody>
</table>
Actions

- for ambiguous input sentences:
  - create identical copy of world A-Box for each reading
  - perform sequence of actions in each A-Box copy in parallel
  - if an action fails, discard the reading
  - if there is only one successful action sequence in the end: choose this reading
  - if there are several successful action sequences: report a true ambiguity
  - if there is no successful action sequence: report an error
Text Generation

Content Determination

Reference Generation

Realization
Content Determination

• input: instantiated user knowledge slot of last performed action

• task: verbalize “add” branch
  – lists like [has-location(a2 myself)] can be passed to the Reference Generation component without change
  – lists like [describe(a2)] are more complicated:
    • query Racer for all most specific concepts that a2 belongs to and all of its role assertions in the world A-Box
    • replace describe by a list of properties of a2
Reference Generation

• input: individuals with names like a2
• task: generate NL NPs that refer to these individuals
• objects that are new to the player:
  – retrieve information about type (and color) of object from world A-Box
  – generate indefinite NP containing this information
Reference Generation

• objects that player already encountered:
  – incremental algorithm:
    • look at object’s properties in predefined order (type, color, location, parts, …)
    • add property to description if at least one other object is excluded by not sharing this property
    • continue until description uniquely defines object
  – Racer queries for each step
  – example (if player knows about both apples, but not about the worm): a2

[content(goal:13
  sem:[13#has-detail(a2,w1), indef(w1), worm(w1),
       def(a2), apple(a2), green(a2)]])
Realization

- transform information into a NL text
  - sentence by sentence
  - Tree Adjoining Grammer (TAG)
Conclusions

- all language-processing components (except parser and NL realization module) use inference system
- most queries are A-Box queries
  - challenge for theorem prover
  - efficient A-Box reasoning relatively new
- Racer’s performance good enough for fluent gameplay on knowledge bases with several hundred individuals
- identification problem avoided (so far)
Racer

- **Renamed A-Box and Concept Expression Reasoner**
- tableau calculus
  - proof procedure for FOL
- multiple T-Boxes and A-Boxes
- reasoning about:
  - OWL Lite
  - OWL DL with approximations for nominals
  - algebraic reasoning beyond the scope of OWL
- unique name assumption can be switched off
- datatypes: integer, string, real
- new query language nRQL (pronounced “nercle”)

Prover9 / Mace4

• Prover9:
  – theorem prover for FOL and equational logic
  – successor of Otter
  – resolution refutation proof

• Mace4:
  – model builder for FOL and equational logic
  – complement to Prover9

• Prover9 and Mace4 can be run in parallel, Prover9 searching for a proof and Mace4 searching for a counterexample
References

  http://www.ps.uni-sb.de/Papers/abstracts/jolli04.pdf

  http://www.informatik.uni-ulm.de/ki/Liebig/papers/TR-U-Ulm-2006-04.pdf

• Racer: http://www.racer-systems.com/

• Prover9 / Mace4: http://www.cs.unm.edu/~mccune/mace4/