But is human-like dialogue ability really needed?

- Turing and Loebner Contests
  - Limited to text input/output
  - Awarded to most human-like computer
  - Turing Contest: Instantiation of the Turing Test
    - Became known as the Turing Test
    - We cannot tell it is a computer, we have achieved artificial intelligence
    - If we can build a computer system that can carry on a conversation and
    - Turing equated dialogue ability with full artificial intelligence

Overview

- Terminal Substitution
- What is Semantics?
- Interpretation
- Example Grammars
- Regular Grammars
- Shortcomings of FSM
- Slightly Beyond FSMs
- FSMs
- Introduction
Tradeoffs

- How much flexibility should we give the user?
  - The less we give the user, the simpler the system is.
  - But, might be difficult/daunting to user.

In the course,
- Start with simple systems (and components)
- Understand what they work
- Understand their advantages and disadvantages
- Start thinking about whether we can build more natural systems

Practical Dialogues

Practical Dialogue Hypothesis (Allen et al., 2001, AI Magazine)
The conversational competence required for practical dialogues, although still complex, is significantly simpler to achieve than general human conversational competence.

- Practical dialogues
  - conversations about some task
  - successful is user accomplished the goal in a reasonable amount of time
  - don't need to build a system
  - that can converse about poetry
  - that can fully understand all of the nuances of what you say

Emphasis is on the technologies and how to put them together.
Components of a Spoken Dialogue Systems

- Speech recognition
  - Convert audio signal into a sequence of words
- Understanding
  - Map system's response into a language that system understands
- Reasoning
  - Decide what to display in response to user's speech
- Generate response
  - Map word sequence into sequence of words
- Speech synthesis
  - Convert sequence of words into audio signal

But none of these problems are solved.

Varying Complexity of Spoken Language Systems

- How complex the system's actions are
- How complex user utterances are
- How indirect the actions are

From DeMori 2003 HTL/NAACL dialogue workshop
Course Info

Assignments 50%
Presentation 15%
Final Exam 35%

http://www.csee.ogi.edu/class/csc550

Syllabus

Finite-State Dialogue Management
- Separating ASR from Parsing and Semantics

Form-based Dialogue Management
- Speech acts, Information State Update

Rule-based Dialogue Management
- Separating ASR from Parsing and Semantics

Finite-State Dialogue Management
- Speech acts, Information State Update

Special Topics
- Grounding, Turn-taking, Adjacency, Parts of Initiative, Discourse
- Simulated users, Objective Function, Reinforcement Learning
- Machine Learning of Dialogue Policies
- Observe State, Information Structure
Overview

• Introduction
  ⇒ FSMs
• Slightly beyond FSMs
• Shortcomings of FSM
• Regular Grammars
• Example Grammars
• Terminal Substitution
  • What is Semantics?
  • Interpretation
• Example Semantics

Academic Integrity

• It is unacceptable to discuss the general concepts and principles behind an assignment with other students.
• It is not proper to arrive at collective solutions.
• Each student is expected to develop, write up, and hand in an individual solution and, in doing so, develop a sufficient understanding of the problem and solution as to be able to explain it adequately to the instructor.
• It is unacceptable to discuss the general concepts and principles behind an assignment with other students.
• Each student is expected to develop, write up, and hand in an individual solution and, in doing so, develop a sufficient understanding of the problem and solution as to be able to explain it adequately to the instructor.
Simple Dialogue System

- Canned dialogues
  - Where system follows a set script
  - User just fills in some blanks here and there
  + Answers yes or no, or from a very limited set of responses
  - User fills in blanks here and there
  - Where system follows a set script

How to specify the script (or dialogue behavior)?

- Use a flowchart?

Dialogue Flow

- Misunderstandings might need to be resolved
- Why?
  - User says something
  - System says something
  - User says something
  - System says something
  - User says something
  - System says something
  - User says something
- To accomplish some goal, will take a number of speaking turns

- Limitation of speech recognition technology
- Limitation of what is said
User responses

- User responses allow a small number of allowed responses
- Context-dependent speech recognition
- Dialogues driven by the system
- User's actions are very restricted
- User must give information one piece at a time
- User's actions are very restricted
- Each response has simple mapping to meaning
- Recognizer constrained to allowed responses
- Responses usually just a single word
- Small number of allowed responses
- Each response has simple mapping to meaning
- Recognizer constrained to allowed responses
- Responses usually just a single word

Typically only allow a small number of allowed user responses

Transitions: allowed responses

Master-slave dialogue, with system as master and user as slave

FSM for Dialogue

What happens in a dialogue specified by a finite state model?
- States can have a system output
- Set of transitions between states
- Set of states

What is a finite state model?
- Specifies state actions
- Specifies transitions
- Specifies output

Set of states
- Includes initial state
- Includes final state

Thank you

largelargemediummediumcheesecheesecheese veggiveggieveggie
smallveggievalargesmall mediumcheesecheesecheese veggiveggieveggie
What size pizza do you want? What kind? What kind? What kind?
Advantages of FSM:

• Advantages of FSM:
  - Speech recognizer uses dialogue state
  - Just has to distinguish between a few alternatives
  - Speech synthesis can use 'canned phrases'
  - Easy to understand 'flow-chart' of what can happen
  - Most appropriate for system-controlled dialogues
  - Where system can ask questions, and user provide short answers

• Disadvantages of FSM:
  - Forces user to segment what they want to do into the states of the
  structured dialogue
  - User may have several questions, and need to provide
  different answers
  - Speech recognizer uses dialogue state

Best Practices:

• Keep number of possible user responses as small as possible
  - System prompts should elicit a few alternatives as possible
  - If special commands such as 'cancel', should be in all states, and user
  - Style of prompts and responses always the same
  - Allows user to figure out how system will work and adapt to it

• Keep things consistent, transparent, and easy-to-use
  - If special commands such as 'cancel', should be in all states, and user
  - Style of prompts and responses always the same
  - If special commands such as 'cancel', should be in all states, and user
  - Style of prompts and responses always the same
  - If special commands such as 'cancel', should be in all states, and user
  - Style of prompts and responses always the same

• Can do trial runs to test our system prompts
  - Ask "Are you male or female?"
  - User could say "man", "I am male", or "masculine"
  - System prompts should elicit a few alternatives as possible
  - If asking question of user: two choices
  - Keep number of possible user responses as small as possible

CSE560 Class 01: 16
© P. Heeman, 2003
FSMs with Variables and Code

Speech Synthesis
- Can use template-based generation.
- Can use language-based generation.
- Template-based generation.
- Template-based generation.
- Template-based generation.
- Template-based generation.
- Template-based generation.

FSMs with Variables and Code

Speech Synthesis
- Can use language-based generation.
- Can use template-based generation.
- Can use language-based generation.
- Can use template-based generation.
- Can use template-based generation.
- Can use template-based generation.
- Can use template-based generation.
- Can use template-based generation.
- Can use template-based generation.

Overview

- Terminal Substitution
- What is Semantics?
- Interpretation
- Example Grammars
- Regular Grammars
- Shorthand Notation of FSM
- Slightly beyond FSM
- Introduction
Recursive Transition Networks

- Equivalent to Context-Free Grammar
- Can have node that is itself an RTN

With RTN's:
- Would have to have this 'inline'
- Redo it every time or copy it in
- With Finite Transition Networks
- could reuse them in any application
- For street address, city and zip code, or amounts
- Might want common routines

Example of Variables: Banking Application

- User can transfer, withdraw
- type of transaction?
- account? account? from?
- amount? balance is ...
- password?
- Exit
- userid?

- balance in $s9(recog) is ... and $s10(recog) is ...

- For now, only allow amounts of 20, 40, 60, 80
- Difficult to do the amounts
- balance, money and find out account
- User can transfer, withdraw

- 20 40 60 or 80
Even though powerful...
What is Needed?

- Table lookup would also require 9999 items!
- Interpret phrases
  - (In fact more, since different ways of saying same number)
  - For numbers between 1 and 9999, there would be 9999 items!
  - Infeasible to list all possibilities and read them all as individual words
- Recognize phrases

What is Needed?

Banking Application

- User can transfer, withdraw balance
  withdrawal
  type of transaction? account? account? from? to? amount? balance is ...

- Password?
- Exit

- Difficult to do amounts
  - Could add extra states, how many one's?
  - How many one's?
  - How many hundreds?
  - How many thousands do you want?

- Balance
  - User can transfer, withdraw money and find out account balance
Overview

Introduction

FSMs

Slightly beyond FSMs

⇒ Regular Grammars

Example Grammars

⇒ Interpretation

What is Semantics?

⇒ Terminal Substitution

⇒ Allow User to Over-answer

Would also be good to allow user
balance withdraw transfer
type of transaction?
account? account? from?
to? amount? balance is ...

⇒ Toward Mixed-initiative dialogues

skip to appropriate state
- If user specifies extra parameters,
- e.g. balance of checking account
- e.g. transfer from checking to savings
- allow user to also say other parameters
- When asked transaction type,
-User can take initiative to add extra
information that was not requested
- User can take initiative to add extra
information that was not requested
- User can take initiative to add extra

Would also be good to allow user
to specify more things at once
Right-Linear Regular Grammar

- Start state
- Non-terminals (For clarity: tokens that start with $)
- Terminals (For clarity: tokens that start with letter)
- Non-terminals (For clarity: tokens that start with $)

Rules of form:

- Right-linear: Non-terminal = 0 or one non-terminals followed by 0 or more terminals
- Similarly, can define Left-linear regular grammars

Right-linear regular grammars (same for Left-linear regular grammars)

- Start state
- Non-terminals
- Terminals
- Non-terminals

All have the same power:

- Different formats for writing grammars
- Same expressive power as finite state machines and regular expressions
- Recognizer in CSLU toolkit allows regular grammars
- Grammar helps it know things do make sense
- "It the who who" doesn't make sense and probably was not said
- Shouldn't take into account when things make sense to the naked ear
- Need way to concisely represent what recognizer can accept

Speech Recognition of Phrases
Delimiters

- Repeat zero or more times: \{ \}
- Repeat one or more times: < >
  - For optional:[ ]
  - For grouping: ( )
- Right hand side can include delimiters
  - Right Linear Regular Grammars don't have this restriction
- Only one rule can be used to define non-terminal

CSLU Toolkit Notation

- Non-terminals start with '$'
- Non-terminal on left hand side
  - A grammar rule
  - '=' used to separate left-hand side from right-hand side
  - Non-terminals start with '$'
  - RAD Tutorial 1.5
Grammar For Amounts

- Recognize digits one to nine
  \[
  \text{OneToNine} = \text{one} \mid \text{two} \mid \text{three} \mid \text{four} \mid \ldots \mid \text{nine}
  \]
- Recognize numbers from ten to nineteen
  \[
  \text{Teens} = \text{ten} \mid \text{eleven} \mid \ldots \mid \text{nineteen}
  \]
- Recognize numbers from twenty to 99
  \[
  \text{Tys} = \text{twenty} \mid \ldots \mid \text{ninety} \mid \text{OneToNine}
  \]
- Recognize numbers from one to 99
  \[
  \text{Tens} = \text{Tys} \mid \text{Teens} \mid \text{OneToNine}
  \]
- Recognize numbers from 100 to 999
  \[
  \text{Hundreds} = \text{OneToNine} \ \text{hundred} \ \text{Tens}
  \]
- Recognize numbers from 1000 to 9999
  \[
  \text{Thousands} = \text{OneToNine} \ \text{thousand} \ \text{Hundreds}
  \]
- Recognize from 1 to 9999 dollars
  \[
  \text{Amount} = (\text{Thousands} \mid \text{Hundreds} \mid \text{Tens}) \ \text{dollars}
  \]

Ensuring Regularness

- Rules are ordered
- Non-terminals in right-hand side must already be defined

Ensuring Regularness - Useful if there are efficient algorithms for right-linear regular grammars + CSTL formalism can be mapped into a right-linear regular grammar - But equivalent in power - Easier to write rules in this notation than a right-linear regular grammar - P=q \Rightarrow q \neq q
- So following is not allowed
- Rules are ordered
Finite State Machine for Numbers

- From Automata, FSM is equivalent to regular grammars.
- Can convert the regular expression for $Typs to an FSM.
- Example of FSM machine for $Tys:
  - one
  - two
  - three
  - four
  - five
  - six
  - seven
  - eight
  - nine
  - twenty
  - thirty
  - forty
  - fifty
  - sixty
  - seventy
  - eighty
  - ninety

- So, can use FSMs for dialogue states and word sequences.

7 Rules to capture 9999 different numbers:

1. Thousands = One to Nine thousand (All digits)
2. Tens = One to Nineteen (All tens)
3. $Tys = Twenty to Ninety (All tens)
4. $OneToNine = One to Nine
5. $Teens = Ten to Nineteen
6. $Hundreds = $OneToNine hundred
7. $Thousands = $OneToNine thousand

Full Grammar:

- Not only concise, but easy to debug, and maintain.
User must still learn all allowed syntax
- Speech recognition harder now!
- One node that allows user to ask any question

FSM for dialogue structure could be pretty simple

| Turn up music in dining room |
| Turn off music in living room |
| Turn on light in kitchen |

Commands to system

Home Automation Example
Aside: How Many Variations are needed?

- Measure speech recognition performance
- Collect a bunch of samples from lots of people
- Do test runs to find out if coverage is adequate
  - Slower it will become
  - Greater chance it will misrecognize
  - Larger search space recognizer is searching over
  - Do test runs to find out if coverage is adequate
  - So many variations...

Banking Example

withdraw 20 twenty dollars from my savings account
balance of my savings account
transfer ten dollars from checking account to the line of credit
transfer forty dollars to my savings from checking
transfer forty dollars from savings to checking
transfer twenty dollars from savings to checking
transfer twenty dollars from checking to line of credit

Example: Home Automation

- Speech recognizer returns output like "turn on light in kitchen"
  - What do we do with this?
  - Need to convert it into an instruction to give the back-end
  - What does back-end for home automation want?
    - Need to interpret it
      - Need to convert it into an instruction to give the back-end
      - What do we do with this?

(set room [join [lrange $reco 4 end] ""]
  set device [lindex $reco 2]
  set state [lindex $reco 1]
)
Example: Amounts in Banking

- Speech recognizer returns strings of form:
  - "nine hundred seventy nineteen"
  - "two thousand three hundred forty five"

- Code to build the amount will be VERY cumbersome to write:
  - "two thousand three hundred forty five"

- Code to build the amount will be VERY cumbersome to write:
  - "two thousand three hundred forty five"

- We need a principled way of doing this!

Example: Student Database

- Did student 555 take CSE550?
- What was the grade of student 555 in CSE550?
- How many students took CSE550 in Summer 2002?
- What was the highest mark in CSE550 in Summer 2002?
- When was CSE550 last taught?
- Did student 555 make CSE550?

- Code to build SQL query will become very cumbersome.
Example: Amounts in Banking

• Need it in a numerical format, i.e., 3225

• What does "three thousand two hundred twenty-five" mean?

• For our banking application, it is a monetary amount that we will transfer or withdraw.

• We still need to map words to meaning.

---

Overview

• Introduction
• FSMs
• Slightly beyond FSMs
• Regular Grammars
• Example Grammars
• Terminal Substitution
• What is Semantics?
• Interpretation
• Regular Grammars
• Shortcomings of FSMs
• Slightly beyond FSMs
• Introduction
Semantics is the internal representation that the computer converts the human language into, so it can easily reason with it.

- Unlike human language, non-ambiguous
- Expressive enough to capture what computer needs to reason about
- But collapses differences that are too subtle for computer

Is there a principled way of building the appropriate semantic representation from the words?

"Can you turn on the light?" versus "Turn on the light"

We will refer to the formalism for building the semantic representation as the Semantic Building Formalism.

Home Automation Example

Speech recognizer returns sentences of form "Turn on light in kitchen"

Want to map the words into a simpler representation that a computer can process

Could pass this to a procedure that executes this:

\[ \text{turn(State,Device,Room)} \]

Or just use: State Device Room

Could use following representation:

- Could pass this to a procedure that executes this
- Computer can process

Speech recognizer returns sentences of form "Turn on light in kitchen"
For home automation, let's use a representation language that maps on to:

- Terminal Substitution Formalism
- Have speech recognizer output the wanted symbols instead of the actual words that were said.

<table>
<thead>
<tr>
<th>Transformation</th>
<th>Term</th>
<th>In</th>
<th>Living Room</th>
<th>Kitchen</th>
<th>Light</th>
<th>Off</th>
<th>On</th>
</tr>
</thead>
<tbody>
<tr>
<td>on</td>
<td></td>
<td>in</td>
<td>living room</td>
<td>kitchen</td>
<td>light</td>
<td>off</td>
<td>on</td>
</tr>
<tr>
<td>kitchen</td>
<td></td>
<td>on</td>
<td>living room</td>
<td>kitchen</td>
<td>light</td>
<td>off</td>
<td>on</td>
</tr>
<tr>
<td>light</td>
<td></td>
<td>on</td>
<td>living room</td>
<td>kitchen</td>
<td>light</td>
<td>off</td>
<td>on</td>
</tr>
<tr>
<td>living room</td>
<td></td>
<td>on</td>
<td>living room</td>
<td>kitchen</td>
<td>light</td>
<td>off</td>
<td>on</td>
</tr>
<tr>
<td>turn</td>
<td></td>
<td>on</td>
<td>living room</td>
<td>kitchen</td>
<td>light</td>
<td>off</td>
<td>on</td>
</tr>
</tbody>
</table>

So, semantic building language formalism needs to map on to:

- Terminal Substitution

For home automation, let's use a representation language or form.

Simple Semantic Processing

Overview

- What is Semantics?
- Interpretation
- Example Grammars
- Regular Grammars
- Shortcomings of FSM
- Slightly Beyond FSM
- FSMs
- Introduction
Incorporating into Regular Grammars

Terminal Substitution Formalism can be embedded into the

CSLU Grammar Constructs

% When following a word, will skip the word in the recognition results if the first word is recognized

% When following a word, will substitute the next word into the recognition output

Example

room = (living%livingroom room%%) | kitchen | ...
command = turn%% state $device in%% [ the%% room ]

The recognition output