Semantics and Pragmatics for Pragbot

Ian Perera

Overview of Semantics and Pragmatics Components

- Goals
  - Assign semantic roles to words in a given parse tree using VerbNet[1]
  - Construct underspecified semantic structures from semantic roles
  - Satisfy semantic structures using context
  - Store information, respond to queries, or transform commands into LTL

[1]Karin Kipper-Schuler. 2005. VerbNet: A broad-coverage, comprehensive verb lexicon. Ph.D. thesis, Computer and Information Science Dept., University of Pennsylvania, Philadelphia, PA, June.

VerbNet

- Expert maintained database that maps verb frames to semantic roles
- VerbNet Structure contains
  - **Equivalent Senses:** Search, Scout, Scavenge, Check
  - **Syntactic frame:** np v np pp.theme
  - **Example sentence:** "i searched the cave for treasure."
  - **Semantic role frame:** agent v location {for} theme

Process:

1. Find the verbs in the tree
2. Split clauses into individual trees
3. Fit each transformed clause to the frame for each verb
4. Choose the match that expresses the most semantic roles

Transformations

- VerbNet syntactic frames are declarative
- Transformations are required to handle:
  - Passive voice
    - “Tell me if you get flipped.” → “Tell me if something flips you.”
  - Existential there
    - A “to be” verb frame also needs to be added
  - Questions

Question Semantics

- VerbNet frames can be used to parse questions after wh-movements and clause inversion
- We don’t use a grammar, just manipulate the trees in code
- Perform
  - Clause Inversion
  - Wh-movement

VerbNet Results

- “While I kill the bad guys in the hallway, you search the rooms for bombs.”
  - search
    - {'Theme': 'bombs', 'VERB': 'search', 'Location': 'rooms', 'Agent': 'you'}
  - <while>
  - kill
    - {'VERB': 'kill', 'Patient': 'the bad guys in the hallway', 'Agent': 'I'}

VerbNet Advantages/Disadvantages

- Advantages
  - Can extract semantics from a large database of verbs (over 5000 verb senses)
  - Provides a consistent “interface” for language understanding
- Disadvantages
  - No implementation standard
  - Syntactic restrictions are not well-defined, and difficult to enforce
    - We can use semantic tags from parse to help restrict roles
      - Locations, directions, etc.
  - Gaps in the database
    - However, we can easily add verbs and frames to the database in XML

Semantic Representations

- Another level of abstraction from the VerbNet frame object
- Contain predicates that can be left unsatisfied and fulfilled with pragmatic inference
- Statements and queries consist of Entity Classes and/or predicates
- Events represent occurrences in the environment and can trigger behavioral or conversational commands

Entity Class

- Consists of a parameterized, generalized quantifier and a list of predicates
- Represents a set of possible sets of Entities
- Used to designate a set of Entities in the environment, or to query the existence or properties of a set of Entities

Quantifiers

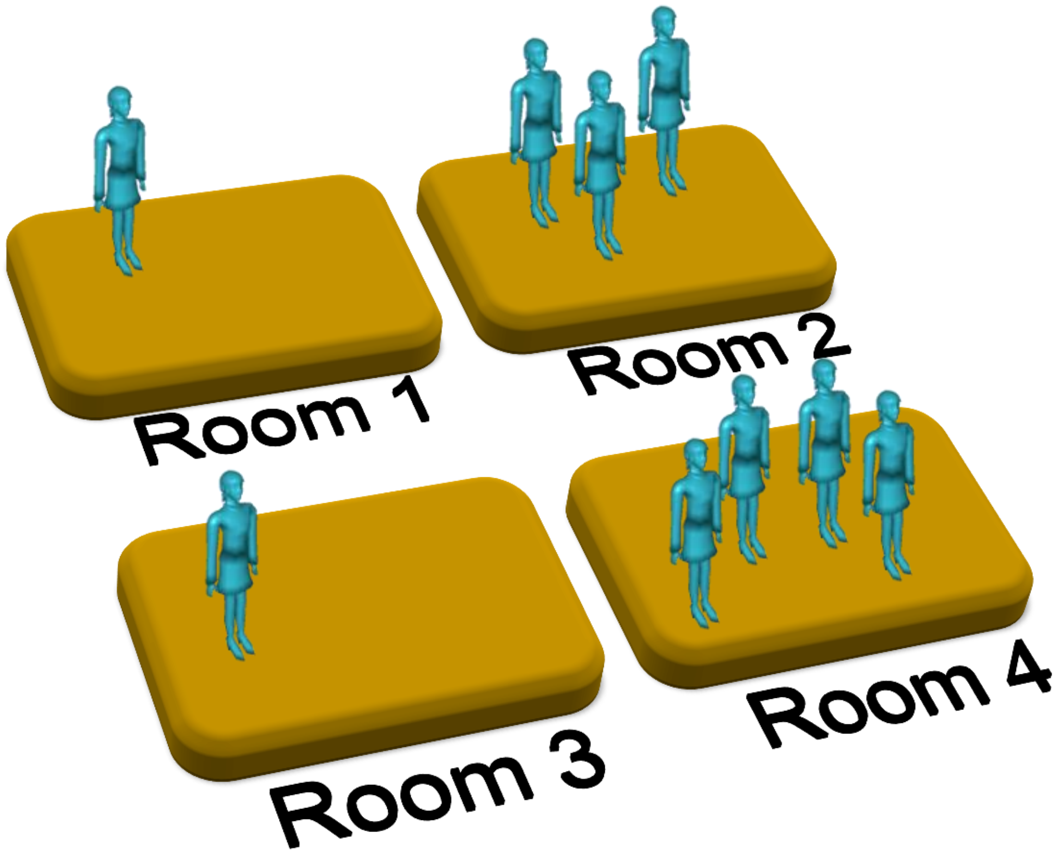
- Encompasses the quantifier and the determiner in a referring expression
- Quantifier properties are expressed as parameters:
  - Definite: Refers to a specific entity
  - Number
  - Proportionality: at least, at most, exact

Entity Class Example

"Where are at least 3 hostages?"

(Given room-based indexing)

- WhQuery(EntityClass, queried predicate type)
- EntityClass:
  - *Quantifier*:
    - Definite: False
    - Number: 3
    - Proportionality: "at least"
  - *Predicates*: {**EntityType**: "hostage"}
- Queried Predicate Type:
  - **Location**

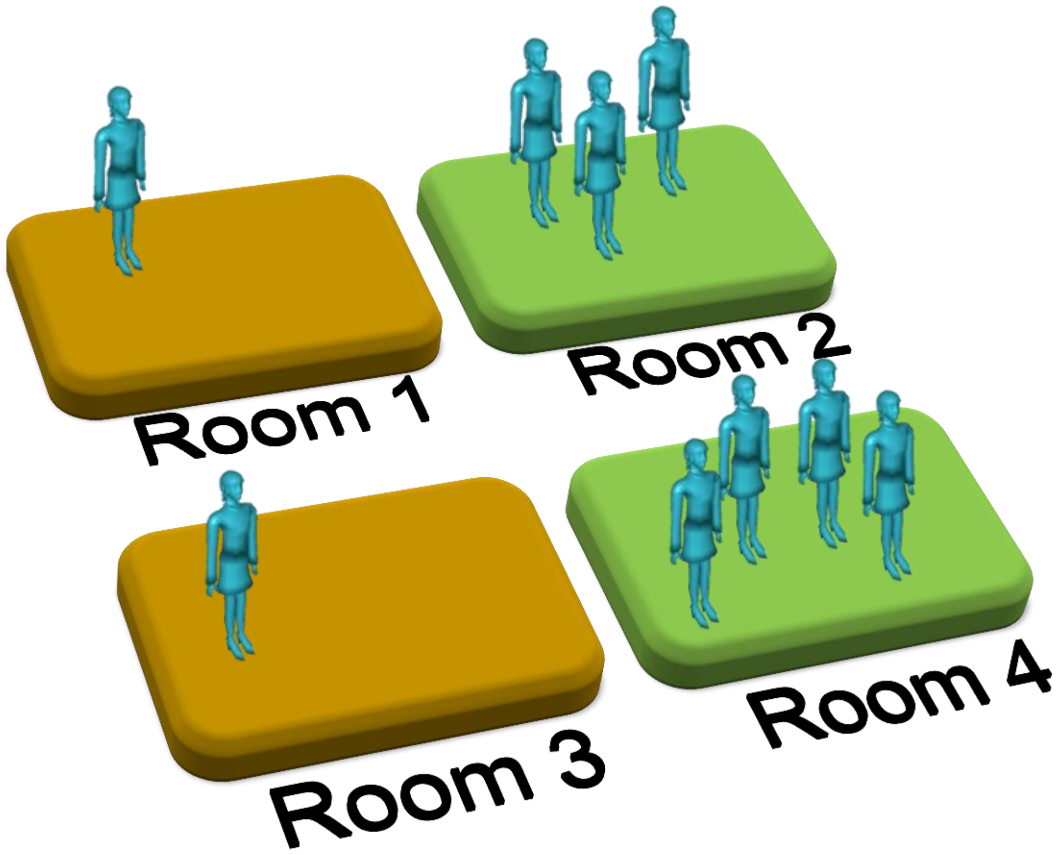


Entity Class Example

"Where are at least 3 hostages?"

(Given room-based indexing)

- WhQuery(EntityClass, queried predicate type)
- EntityClass:
  - *Quantifier*:
    - Definite: False
    - Number: 3
    - Proportionality: "at least"
  - *Predicates*: {**EntityType**: "hostage"}
- Queried Predicate Type:
  - **Location**



Nested Entity Classes

- An Entity Class can fulfill the value of a predicate
- “the three rooms with two hostages”
- EntityClass
  - *Quantifier*: “the three”
  - *Predicates*: { **EntityType**: “room”,
    Contains: < EntityClass:
    Quantifier: “two”
    Predicates: { **EntityType**: “hostages”>}

Commands

- "Look for User 1 in the library and the classroom."
  - Verbnet:

  {Verb: "look", Theme: "User 1", Location: "Library"}
  and

  {Verb: "look", Theme: "User 1", Location: "Classroom"}
  - *EntityClass*(Q: <definite>, P: {EntityType: "user",
    Name: "User 1",
    Location: [Library, Classroom] })

Events

- “Tell me if you see a hostage.”
- Event
  - Condition:

  <*see*, EntityClass(“a”, {**Theme**: hostage})>
  - Command: <tell>
    - Special notification command

Context

- Multiple contexts provide common ground
  - Discourse context (anaphoric reference)
  - Environment context (exophoric reference)
    - Mutual knowledge
  - Implicit goal context
- Inferences can come from one or more contexts
- Context fills in unspecified parameters in the semantic representations by finding matching references

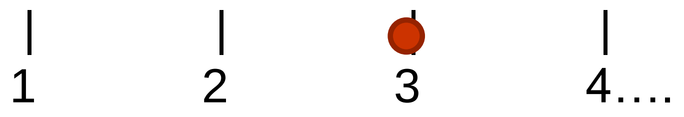
Quantifier Agreement

- Resolving anaphoric and exophoric references is a problem of quantifier and predicate agreement
- Predicates:
  - An Entity Class must contain the queried predicate
- Quantifier:
  - Parameterization of quantifier characteristics allows for comparison of referring expressions
  - A set of entities denoted by quantifier q_1 can be referred to by a different quantifier q_2 as long as q_2 subsumes q_1.
  - The new quantifier can either select the entire set or a piece of it

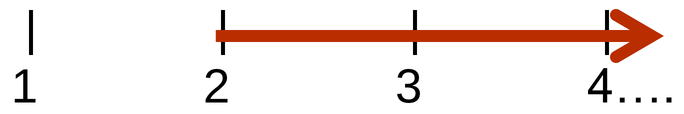
Quantifier Agreement (cont.)

- “There are three bombs in room 2. Defuse them.”

“Three”



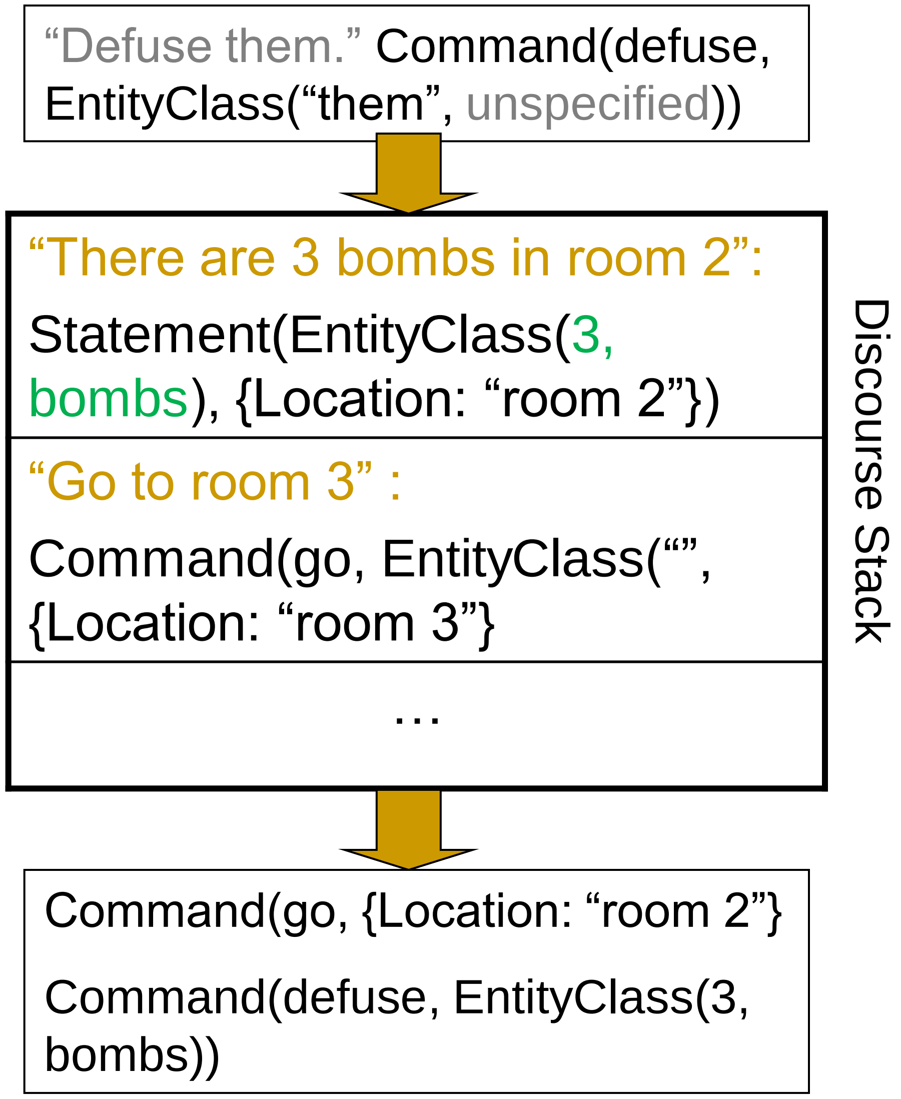
“Them”



- May need to relax this in some cases

Discourse Context

- Implemented using a discourse stack
- Underspecified semantic structures search the stack for the first valid substitution of an Entity Class with compatible quantifier and parameters



“Defuse them.” Command(defuse, EntityClass(“them”, unspecified))

“There are 3 bombs in room 2”:
Statement(EntityClass(3, bombs), {Location: “room 2”})

“Go to room 3” :
Command(go, EntityClass(“”, {Location: “room 3”}

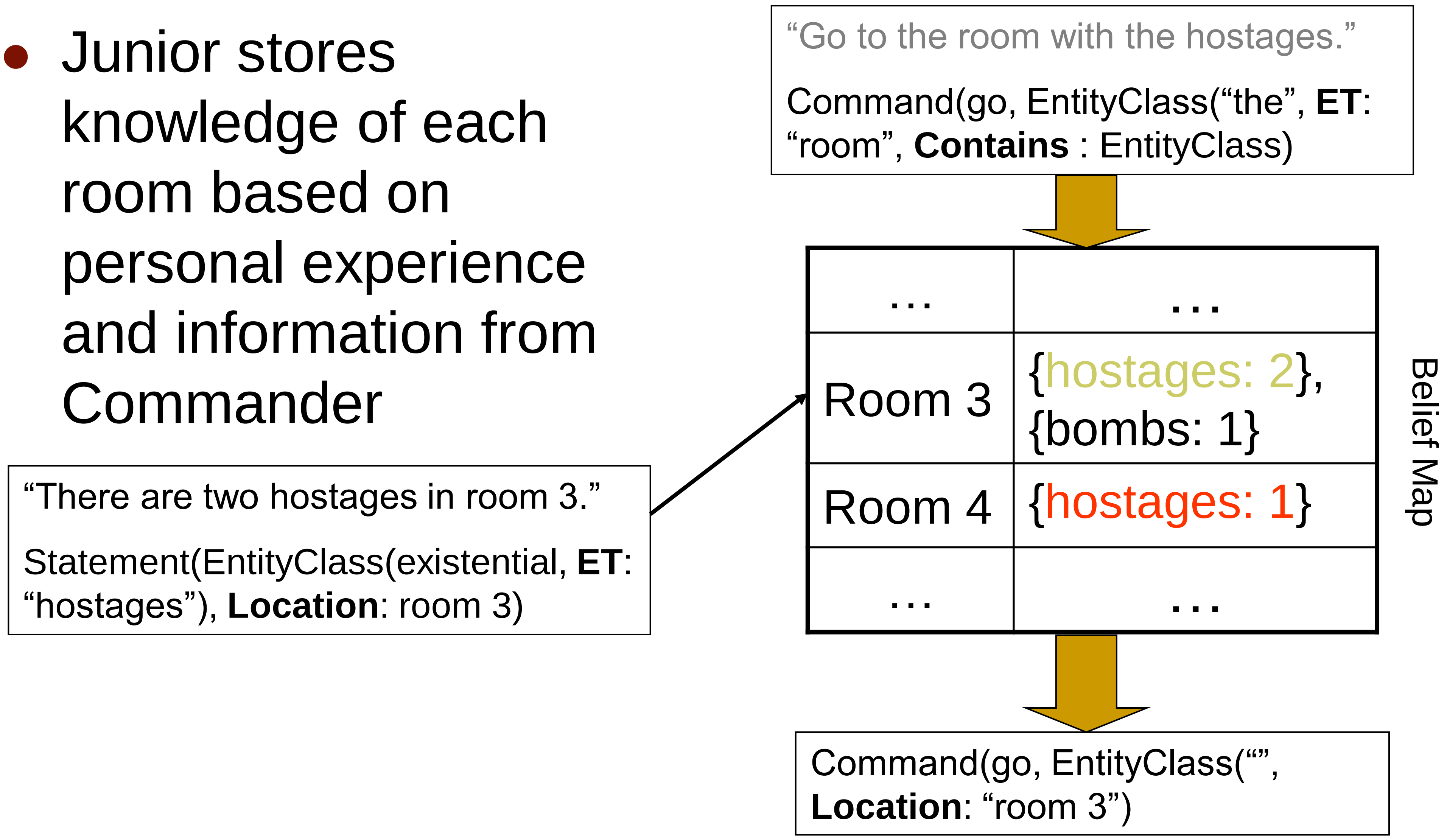
…

Command(go, {Location: “room 2”}

Command(defuse, EntityClass(3, bombs))

Environment Context

- Junior stores knowledge of each room based on personal experience and information from Commander



"Go to the room with the hostages."

Command(go, EntityClass("the", **ET**: "room", **Contains** : EntityClass)

"There are two hostages in room 3."

Statement(EntityClass(existential, **ET**: "hostages"), **Location**: room 3)

…	…
Room 3	{hostages: 2}, {bombs: 1}
Room 4	{hostages: 1}
…	…

Command(go, EntityClass("", **Location**: "room 3")

Mutual Knowledge

- Junior also stores what Junior thinks Commander knows
- Commander knows something if:
  - Commander told Junior
  - Junior told Commander
  - Commander and Junior were together when they witnessed it (Triple copresence)
  - Junior saw Commander go into a room

Clark, Herbert H. and Marshall, Catherine R. “Definite Reference and Mutual Knowledge.” Elements of Discourse Understanding (1981). Volume: 10-63, Issue: 1, Publisher: Cambridge University Press, Pages: 10-63.

Maxim of Quantity

- Don’t say what the other person already knows
  - Junior conveys information differently when it knows the Commander probably knows
- If Junior and Commander are separated: “I see a bomb.”
- If they are together: “Do you see the bomb?”
  - Indefinite articles are infelicitous when both parties know of the referent

Implicit Goal Context

- Junior accepts both “standing orders” and goals
  - Standing orders persist throughout the simulation
  - Goals are forgotten once they are completed
- A standing order provides an action to be carried out upon learning of a particular object

Implicit Goal Example

- Commander says “Defuse any bombs.”
- Then, “There is a bomb in room 3.”
- Generates implicit goal through LTL equivalent to:
  - Command(go, {Location: room 3})
  - Command(defuse, {Theme: “Bomb”
    Location: “room 3”})

Which context to use?

- Currently we search as follows:
  1. Discourse stack to some fixed depth
     - “Commander is referring to something we are currently talking about.”
  2. Junior’s model of Commander’s knowledge
     - “Commander is referring to something he believes I know about, and I do.”
  3. Junior’s model
     - “Commander is referring to something he knows that I know about, but I never told him.”

Instantiating Entity Classes

- Once we have Entity Classes for the commands, we instantiate them
- Find a set of entities that satisfy the quantifier and predicates
  - Send the set to the Language Generation component for questions
    - Including Yes/No questions – Instant Pragmatics!
  - Send the set to the LTL generation for commands

Overview of Semantics and Pragmatics Components

- Assign semantic roles to words in a given parse tree using VerbNet
- Construct underspecified semantic structures from semantic roles
- Satisfy semantic structures using context
- Store information, respond to queries, or transform commands into LTL