Building a Knowledge Base

CPSC 470 – Artificial Intelligence
Brian Scassellati
Syntax and Semantics of First-Order Logic

Sentence → AtomicSentence
    | Sentence Connective Sentence
    | Quantifier Variable,...Sentence
    | ¬Sentence
    | (Sentence)
AtomicSentence → Predicate(Term,...)
    | Term = Term
Term → Function(Term,...)
    | Constant
    | Variable
Connective → ⇒ | ∧ | ∨ | ⇔
Quantifier → ∀ | ∃
Variable → a | b | c | ...
Function → Mother | LeftLegOf | ...
Predicate → Before | HasColor | Raining | ...
Constant → A | X₁ | John | ...

• Quantifiers (∃, ∀)
  – The real power of first-order logic
  – Express properties of entire collections of objects rather than having to enumerate all the objects by name
  – Universal Quantifier (∀)
    • “all cats are mammals”
      ∀x Cat(x) ⇒ Mammal(x)
  – Existential Quantifier (∃)
    • “there exists a fish that can fly”
      ∃x Fish(x) ∧ CanFly(x)
Situation Calculus

- Situations are indexed
  \[ \text{At(Agent,[1,1], S_0)} \land \text{At(Agent,[1 2], S_1)} \]

- Changes from one situation to the next
  \[ \text{Result(Forward, S_0) \implies S_1} \]
  \[ \text{Result(Turn(R), S_1) \implies S_2} \]
Analogies to Programming

Today we will:

• Develop a methodology for building knowledge bases for particular domains and the world in general

• Write some sample “programs” by developing a few example knowledge bases
What is knowledge engineering?

• What do I need that for?
  – I can just use really long variable names
    • Not machine readable/interpretable
    • Does not help when adding new facts
      – Degenerate case: propositional logic
  – Any method of building structures should do the job
    • Yes, but you might avoid some common pitfalls
Properties of Good Knowledge Representation

• Expressive
• Concise
• Unambiguous
• Context-insensitive
• Effective
• Clear
• Correct
How to develop a Knowledge Base (in 5 easy steps)

• Decide what to talk about
• Decide on a vocabulary of predicates, functions, and constants
  – Ontology
• Encode general knowledge within the domain
  – Limiting errors
• Encode a description of the specific problem
• Pose queries and get answers
Ontology

• Choices that you make in specifying the basic elements of the logic (the functions, predicates, and terms) dictate a vocabulary

• This vocabulary gives a way of thinking about the world, a way of dividing the world into meaningful units, a theory of the nature of existence
Limiting Errors

• A properly designed knowledge base will have most common errors isolated to a single statement.

• Errors in a program might be at the line `x=x+1`.
  – But this tells us little about how to solve the error.

• Errors in a KB should be more self-contained (rely on less external context).
Electronic Circuits Domain

- Domain specific knowledge representation example
- This circuit claims to add two bits with a carry bit
- Can we build a logic to analyze this claim?
Electronic Circuits Domain: Decide what to talk about

- Circuits
- Gate Types
- Individual Gates
- Terminals of Gates and Circuits
  - Inputs
  - Outputs
- Connectivity
- Signals
Electronic Circuits Domain: Decide on a Vocabulary

- Name individual gates with constants ($X_1$, $X_2$, $A_1$, $A_2$, …)
- Gate types with a function (Type($X_1$)=$X_OR$)
  - Could use alternate notations (XOR($X_1$) or Type($X_1$,XOR))
  - But using a function guarantees that each gate has only one type
- Terminals (Out(1,$X_1$) is the first output of gate $X_1$)
- Connectivity (Connected(Out(1, $X_1$), In(2, $A_2$)))
- Signal values as objects (Signal(In(1,$X_1$))=On)
Electronic Circuits Domain: Encode General Rules

- **OR gates**: output is on iff any inputs are on
  \[ \forall g \ Type(g) = \text{OR} \Rightarrow \]
  \[ \text{Signal(Out(1,g))=On} \iff \exists n \ \text{Signal(In(n,g))=On} \]

- **AND gates**: output is off iff any inputs are off
  \[ \forall g \ Type(g) = \text{AND} \Rightarrow \]
  \[ \text{Signal(Out(1,g))=Off} \iff \exists n \ \text{Signal(In(n,g))=Off} \]

- **NOT gate**: output is different from input
  \[ \forall g \ Type(g) = \text{NOT} \Rightarrow \]
  \[ \text{Signal(Out(1,g))} \neq \text{Signal(In(1,g))} \]

- **XOR gates**: output is on iff inputs differ
  \[ \forall g \ Type(g) = \text{XOR} \Rightarrow \]
  \[ \text{Signal(Out(1,g))=On} \iff \text{Signal(In(1,g))} \neq \text{Signal(In(2,g))} \]
Electronic Circuits Domain: Encode General Rules

• If two terminals are connected, then they have the same signal
  \[ \forall t_1,t_2 \, \text{Connected}(t_1,t_2) \implies \text{Signal}(t_1) = \text{Signal}(t_2) \]

• The signal at every terminal is either on or off, but not both
  \[ \forall t \, \text{Signal}(t) = \text{On} \lor \text{Signal}(t) = \text{Off} \]
  \[ \text{On} \neq \text{Off} \]

• Connected is commutative
  \[ \forall t_1,t_2 \, \text{Connected}(t_1,t_2) \iff \text{Connected}(t_2,t_1) \]
Electronic Circuits Domain: Encode Specific Instance

- Circuit C1
- Type(X1) = XOR
- Type(X2) = XOR
- Type(A1) = AND
- Type(A2) = AND
- Type(O1) = OR

\[
\begin{align*}
\text{Connected}(\text{Out}(1,X1), \text{In}(1,X2)) \\
\text{Connected}(\text{Out}(1,X1), \text{In}(2,A2)) \\
\text{Connected}(\text{Out}(1,A2), \text{In}(1,O1)) \\
\text{Connected}(\text{Out}(1,A1), \text{In}(2,O1)) \\
\text{Connected}(\text{Out}(1,X2), \text{Out}(1,C1)) \\
\text{Connected}(\text{Out}(1,O1), \text{Out}(2,C1)) \\
\text{Connected}(\text{In}(1,C1), \text{In}(1,X1)) \\
\text{Connected}(\text{In}(1,C1), \text{In}(1,A1)) \\
\text{Connected}(\text{In}(2,C1), \text{In}(2,X1)) \\
\text{Connected}(\text{In}(2,C1), \text{In}(2,A1)) \\
\text{Connected}(\text{In}(3,C1), \text{In}(2,X2)) \\
\text{Connected}(\text{In}(3,C1), \text{In}(1,A2))
\end{align*}
\]
Electronic Circuits Domain: Pose Queries and Get Answers

- What values are output given input (1,0,1)?
  - Assert
    \[
    \text{Signal(In(1,C1))}=\text{On} \land \text{Signal(In(2,C1))}=\text{Off} \land \text{Signal(In(3,C1))}=\text{On}
    \]
  - Infer values of
    \[
    \text{Signal(Out(1,C1))} \text{ and Signal(Out(2,C1))}
    \]
  - Rewrite as a quantifier:
    \[
    \exists v_1,v_2 \text{ \ Signal(In(1,C1))}=\text{On} \land \text{Signal(In(2,C1))}=\text{Off} \land \text{Signal(In(3,C1))}=\text{On} \land \text{Signal(Out(1,C1))}=v_1 \land \text{Signal(Out(2,C2))}=v_2
    \]
Electronic Circuits Domain:
Pose Queries and Get Answers

• What combinations of inputs would cause the output (0,1)?
  – Assert
    \[ \text{Signal(Out(1,C1))=Off} \land \text{Signal(Out(2,C1))=On} \]
  – Infer values of inputs
    \[ \text{Signal(In(1,C1)) \ and \ Signal(In(2,C1))} \]
    \[ \text{and \ Signal(In(3,C1))} \]
  – Rewrite as a quantifier:
    \[ \exists i_1,i_2,i_3 \ \text{Signal(In(1,C1))=i_1} \land \text{Signal(In(2,C1))=i_2} \land \text{Signal(In(3,C1))=i_3} \land \text{Signal(Out(1,C1))=Off} \land \text{Signal(Out(2,C2)=On} \]
General Ontology

• Rather than building domain-specific representations, can we build just one domain-general representation and use it for everything?
Topics for a General Ontology

• How can we represent these types within our general knowledge base?
  – Categories
  – Measures
  – Composite objects
  – Events and processes
  – Time, space, and change
  – Physical objects
  – Substances
  – Mental objects (beliefs, desires, etc.)
Categories

- So far, we have defined categories by using a predicate: Fish(x)
- **Reification** is the process of turning a predicate or function into an object
  - Vegetables is the set of all veggies
    - BobTheTomato ∈ Vegetables
- Reified categories allow us to make assertions about the entire categories
  - Population(Humans) = 7,700,000,000
- Categories allow us to organize the KB through inheritance
Measures

• Quantitative properties of objects like mass, length, and cost
  \[ \text{Length(Box13)} = \text{Meters}(1.4) \]
  \[ \text{Price(Orange13)} = \text{Cents}(20) \]

• Distinguish between amounts and instruments
  \[ \forall d \ d \in \text{Days} \Rightarrow \text{Duration}(d) = \text{Hours}(24) \]
  \[ \forall b \ b \in \text{DollarBills} \Rightarrow \text{CashValue}(b) = \$(1.00) \]
Composite Objects

• An object that has parts is a composite object

• Define a relation to indicate
  – PartOf(Nose, Face)
  – PartOf(Face, Head)
  – PartOf(Head, Body)

• Transitive!
  – Infer PartOf(Nose, Body)
Events

• Why not just rely on situation calculus?
  – Situations are only instantaneous points in time
  – Only works well when a single action links situations

• If the world can change on its own, or if multiple agents are involved, then situation calculus is not sufficient
Events

- Introduce a new **event calculus**
- Events are chunks of the universe in “space” and time
- Intervals are sections along the time dimension
- Places are sections along the “space” dimension
- New notation for events

\[
\forall c,i \ E(c,i) \iff \exists e \ e \in c \land SubEvent(e,i)
\]

\[E(Drive(Scanz,Boston,NewHaven), LastMonday)\]
Predicates on Time Intervals

- Interval is defined by a start time and an end time
- Define intervals in first-order logic

\[ \forall i,j \text{ Meet}(i,j) \iff \text{Time(End}(i))=\text{Time(Start}(j)) \]
\[ \forall i,j \text{ After}(j,i) \iff \text{Before}(i,j) \]
\[ \forall i,j \text{ Overlap}(i,j) \iff \exists k \text{ During}(k,i) \wedge \text{During}(k,j) \]
Physical Objects

- Physical objects can also be viewed as events...
  - They have a spatial and a temporal extent

- Objects that change across time/space are called **fluents**
Substances

• Can we also represent things like sand, glass, butter, etc.? 

• **Intrinsic properties** are part of the substance itself
  – Melting point, density, etc.
  – Survive division

• **Extrinsic properties** are specific to an object
  – Weight, temperature, etc.
  – Do not survive division

• *A substance is defined only by intrinsic properties*
Mental Objects
(Beliefs, Desires, etc.)

• It might be useful to know what you know (and what you don’t know)
  – Stopping pointless searches
  – Attempting to acquire missing information

• Requires a new level of representation
  – First order logic is referentially transparent
    • (You can freely substitute a term for an equal term)
  – Beliefs are opaque
    • (You can’t substitute Superman for Clark)

• Allow a new form of representation: strings
  – “Clark” is a string of five characters
  – “Clark”≠”Superman”
Coming Up

- Real-world uses of logical systems

**Programming**
- Choose a language
- Write a program
- Choose or write a compiler
- Run your compiled program

**Knowledge Engineering**
- Choose a logic
- Build a Knowledge Base
- Implement the proof theory
- Infer new facts

How do we automate the process of inferring new facts?
Administrivia

- PS #2 due tonight
- PS #3 out today (no programming)
- Hopefully, more office hours coming soon…
- Up next: Inference