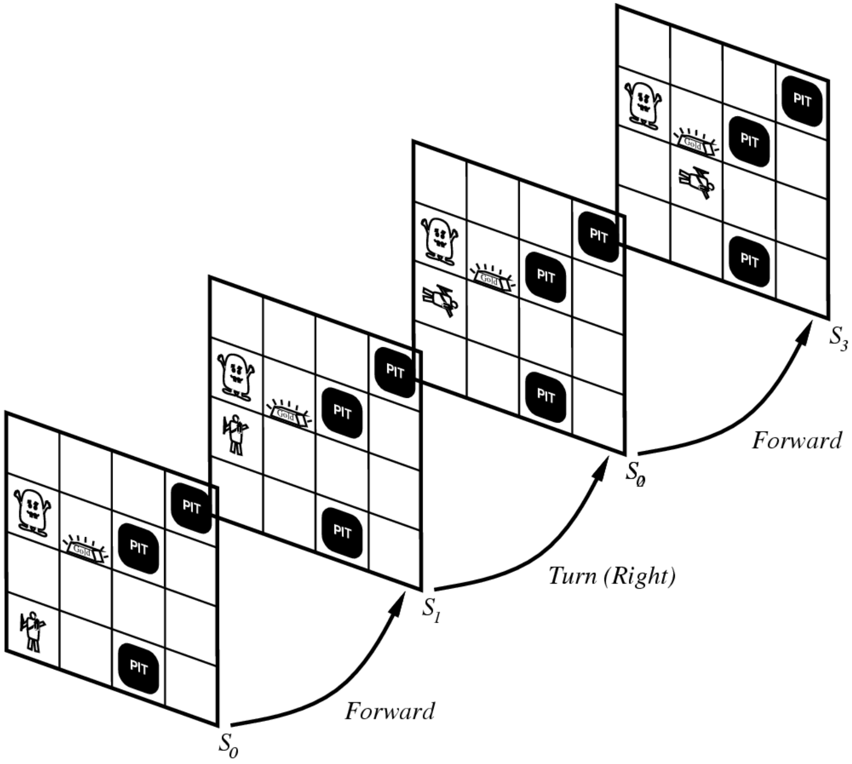
Building a Knowledge Base

Syntax and Semantics of First-Order Logic

$$
\begin{aligned}
Sentence &\rightarrow AtomicSentence \\
&\quad \mid Sentence\ Connective\ Sentence \\
&\quad \mid Quantifier\ Variable, \ldots Sentence \\
&\quad \mid \neg Sentence \\
&\quad \mid (Sentence) \\
AtomicSentence &\rightarrow Predicate(Term, \ldots) \\
&\quad \mid Term = Term \\
Term &\rightarrow Function(Term, \ldots) \\
&\quad \mid Constant \\
&\quad \mid Variable \\
Connective &\rightarrow \Rightarrow \mid \wedge \mid \vee \mid \Leftrightarrow \\
Quantifier &\rightarrow \forall \mid \exists \\
Variable &\rightarrow a \mid b \mid c \mid \ldots \\
Function &\rightarrow Mother \mid LeftLegOf \mid \ldots \\
Predicate &\rightarrow Before \mid HasColor \mid Raining \mid \ldots \\
Constant &\rightarrow A \mid X_1 \mid John \mid \ldots
\end{aligned}
$$

- Quantifiers ($\exists$, $\forall$)
  - 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 ($\forall$)
    - "all cats are mammals"
      $\forall x\ Cat(x) \Rightarrow Mammal(x)$
  - Existential Quantifier ($\exists$)
    - "there exists a fish that can fly"
      $\exists x\ \ Fish(x) \wedge CanFly(x)$

Situation Calculus



- Situations are indexed

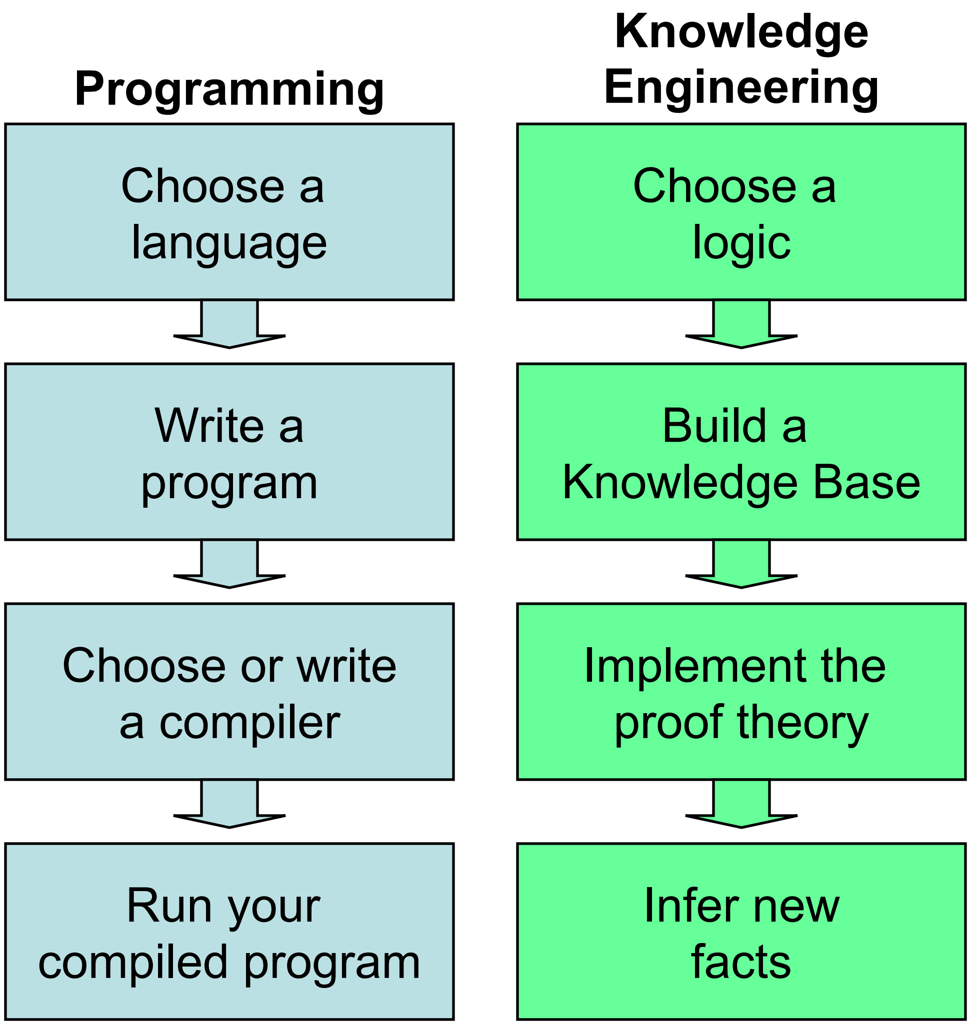
  At(Agent,[1,1], S_0) $\wedge$
  At(Agent,[1 2], S_1)

- Changes from one situation to the next

  Result(Forward, S_0)$\Rightarrow$$S_1$

  Result(Turn(R), S_1)$\Rightarrow$$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

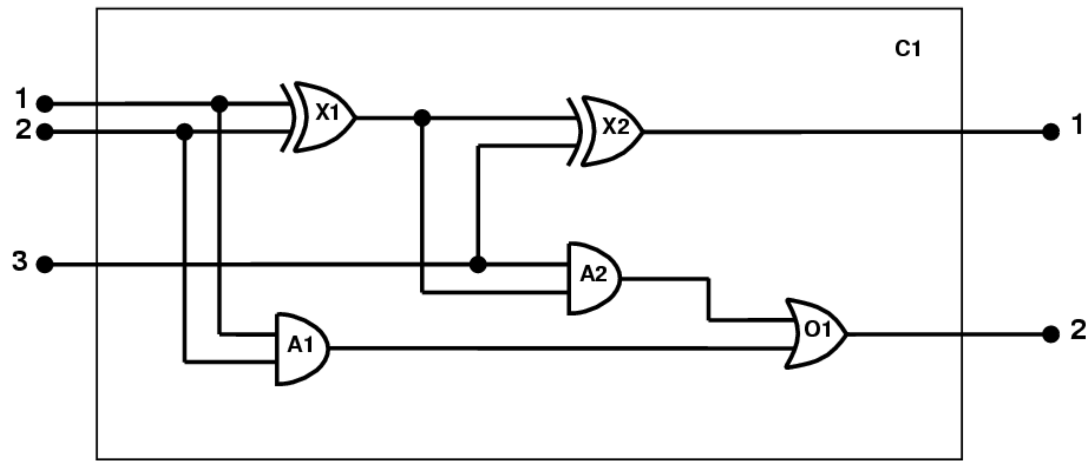
Ontology

- 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

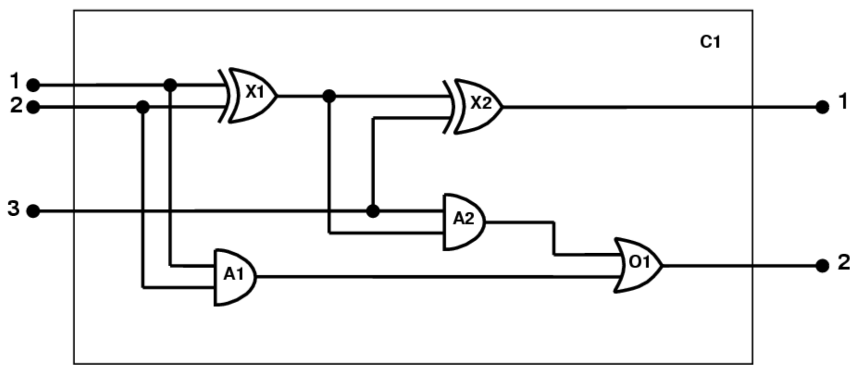
- A properly designed knowledge base will have most common errors isolated to a single statement
- Errors in a program might be at the line x=x+1
  - But this tells us little about how to solve the error
- Errors in a KB should be more self-contained (rely on less external context)

Electronic Circuits Domain



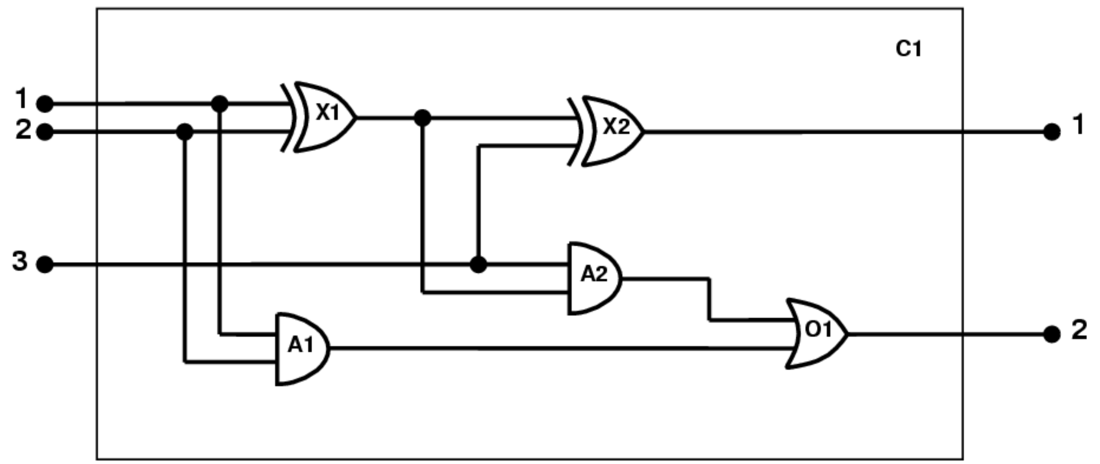
- Domain specific knowledge representation example
- This circuit claims to add two bits with a carry bit
- Can we build a logic to analyze this claim?

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 (X1, X2, A1, A2, …)
- Gate types with a function (Type(X1)=XOR)
  - Could use alternate notations (XOR(X1) or Type(X1,XOR))
  - But using a function guarantees that each gate has only one type
- Terminals (Out(1,X1) is the first output of gate X1)
- Connectivity (Connected(Out(1, X1), In(2, A2)))
- Signal values as objects (Signal(In(1,X1))=On)

Electronic Circuits Domain: Encode General Rules

- OR gates: output is on iff any inputs are on
  $\forall g\ Type(g)=OR \Rightarrow$
  $Signal(Out(1,g))=On \Leftrightarrow \exists n\ Signal(In(n,g))=On$
- AND gates: output is off iff any inputs are off
  $\forall g\ Type(g)=AND \Rightarrow$
  $Signal(Out(1,g))=Off \Leftrightarrow \exists n\ Signal(In(n,g))=Off$
- NOT gate: output is different from input
  $\forall g\ Type(g)=NOT \Rightarrow$
  $Signal(Out(1,g)) \neq Signal(In(1,g))$
- XOR gates: output is on iff inputs differ
  $\forall g\ Type(g)=XOR \Rightarrow$
  $Signal(Out(1,g))=On \Leftrightarrow Signal(In(1,g)) \neq Signal(In(2,g))$

Electronic Circuits Domain: Encode General Rules

- If two terminals are connected, then they have the same signal

  $\forall t1,t2\ \text{Connected}(t1,t2) \Rightarrow \text{Signal}(t1)=\text{Signal}(t2)$

- The signal at every terminal is either on or off, but not both

  $\forall t\ \text{Signal}(t)=\text{On} \vee \text{Signal}(t)=\text{Off}$

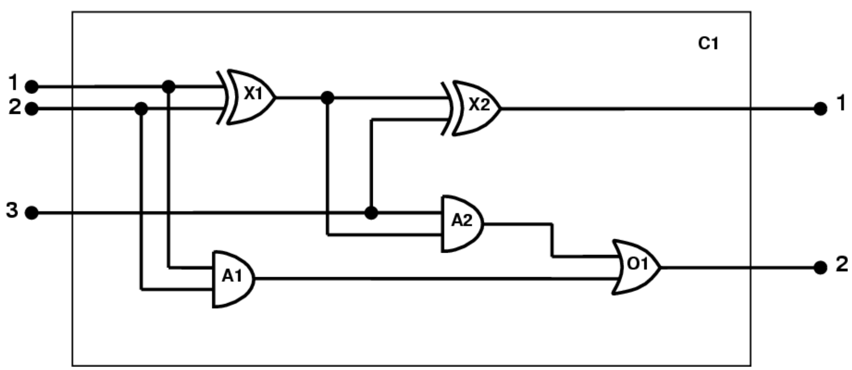
  $\text{On} \neq \text{Off}$

- Connected is commutative

  $\forall t1,t2\ \ \text{Connected}(t1,t2) \Leftrightarrow \text{Connected}(t2,t1)$

Electronic Circuits Domain: Encode Specific Instance

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



- Connected(Out(1,X1), In(1,X2))
- Connected(Out(1,X1), In(2,A2))
- Connected(Out(1,A2), In(1,O1))
- Connected(Out(1,A1), In(2,O1))
- Connected(Out(1,X2), Out(1,C1))
- Connected(Out(1,O1), Out(2,C1))

- Connected(In(1,C1), In(1,X1))
- Connected(In(1,C1), In(1,A1))
- Connected(In(2,C1), In(2,X1))
- Connected(In(2,C1), In(2,A1))
- Connected(In(3,C1), In(2,X2))
- Connected(In(3,C1), In(1,A2))

Electronic Circuits Domain: Pose Queries and Get Answers

- What values are output given input (1,0,1)?
  - Assert

    Signal(In(1,C1))=On ∧ Signal(In(2,C1))=Off ∧
    Signal(In(3,C1))=On
  - Infer values of

    Signal(Out(1,C1)) and Signal(Out(2,C1))
  - Rewrite as a quantifier:

    ∃v1,v2 Signal(In(1,C1))=On ∧ Signal(In(2,C1))=Off ∧
    Signal(In(3,C1))=On ∧ Signal(Out(1,C1))=v1 ∧
    Signal(Out(2,C2)=v2

Electronic Circuits Domain: Pose Queries and Get Answers

- What combinations of inputs would cause the output (0,1)?
  - Assert

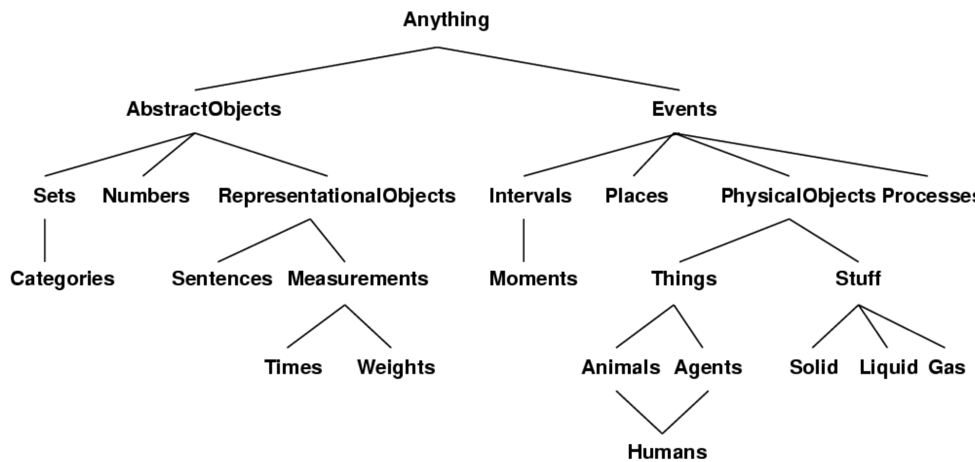
    Signal(Out(1,C1))=Off ∧ Signal(Out(2,C1))=On
  - Infer values of inputs

    Signal(In(1,C1)) and Signal(In(2,C1))

    and Signal(In(3,C1))
  - Rewrite as a quantifier:

    ∃i1,i2,i3 Signal(In(1,C1))=i1 ∧ Signal(In(2,C1))=i2 ∧
    Signal(In(3,C1))=i3 ∧ Signal(Out(1,C1))=Off ∧
    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

  $BobTheTomato \in Vegetables$
- 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

  Length(Box13)=Meters(1.4)

  Price(Orange13)=Cents(20)

- Distinguish between amounts and instruments

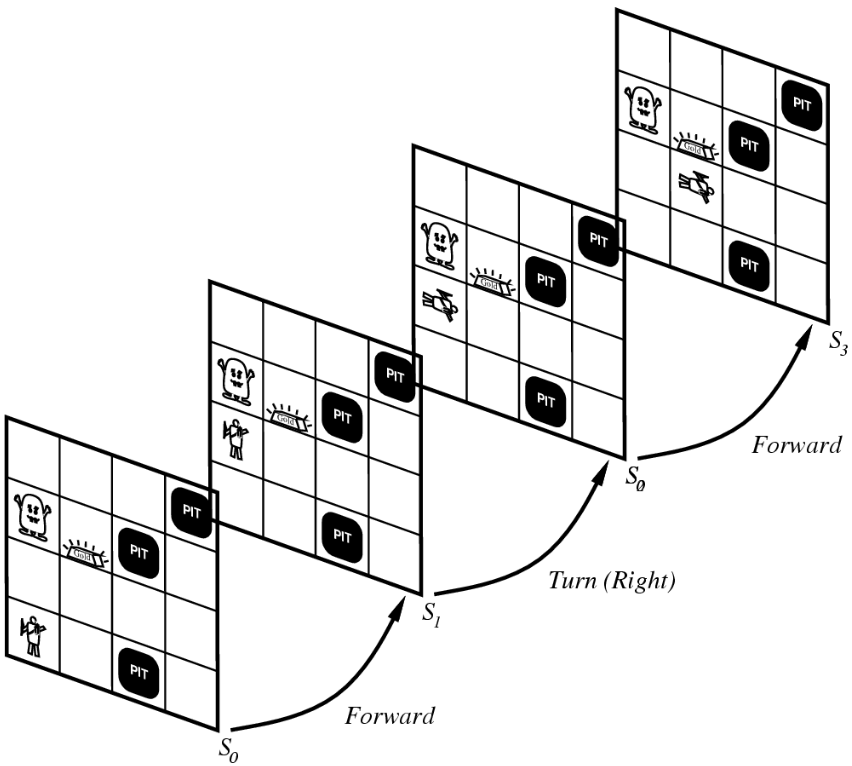
  $\forall d \;\; d \in Days \Rightarrow Duration(d)=Hours(24)$

  $\forall b \;\; b \in DollarBills \Rightarrow CashValue(b)=\$(1.00)$

Composite Objects

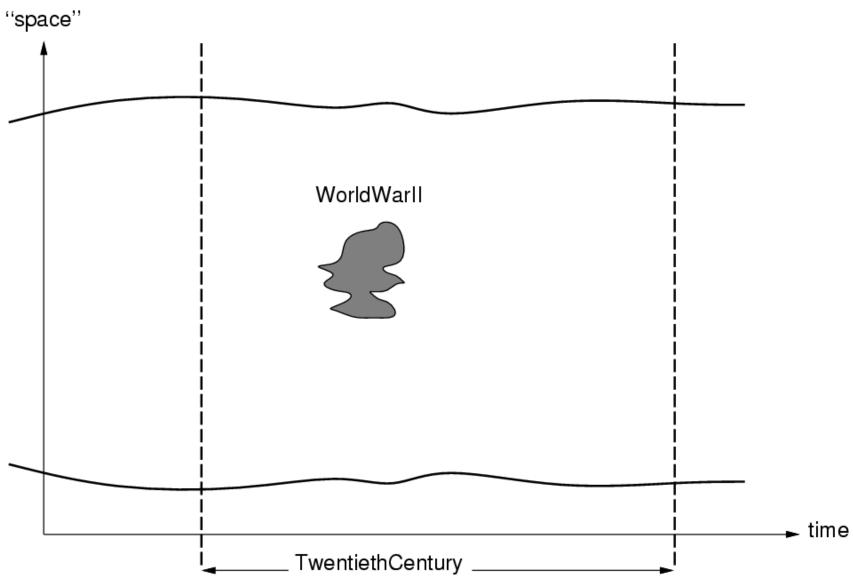
- An object that has parts is a composite object
- Define a relation to indicate
  - PartOf(Nose, Face)
  - PartOf(Face, Head)
  - PartOf(Head, Body)
- Transitive!
  - Infer PartOf(Nose, Body)

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

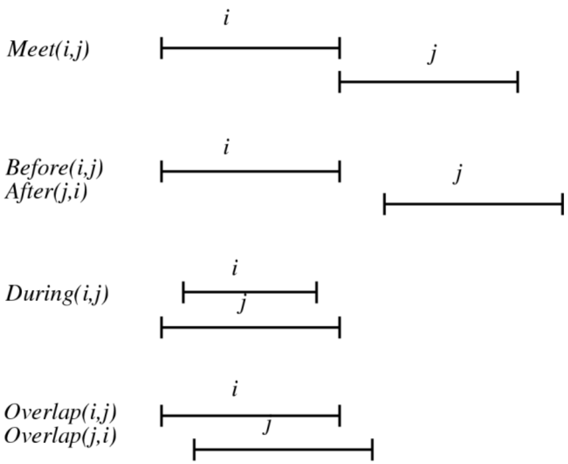


- Introduce a new **event calculus**
- Events are chunks of the universe in "space" and time
- Intervals are sections along the time dimension
- Places are sections along the "space" dimension
- New notation for events

  $\forall c,i\ E(c,i) \Leftrightarrow \exists e\ e \in c \land SubEvent(e,i)$

  E(Drive(Scaz,Boston,NewHaven), LastMonday)

Predicates on Time Invervals



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

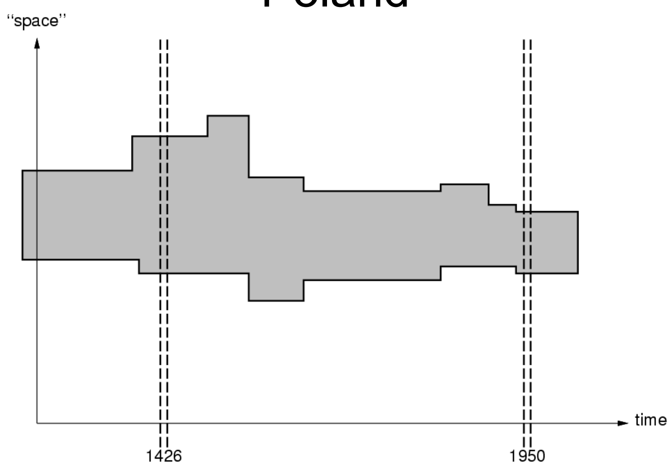
  $\forall$i,j Meet(i,j) $\Leftrightarrow$ Time(End(i))=Time(Start(j))

  $\forall$i,j After(j,i) $\Leftrightarrow$ Before(i,j)

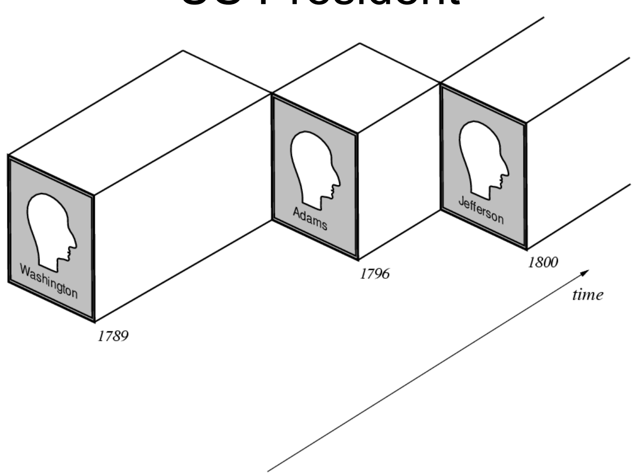
  $\forall$i,j Overlap(i,j) $\Leftrightarrow$ $\exists$k During(k,i) $\wedge$ During(k,j)

Physical Objects

Poland



US President



- 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

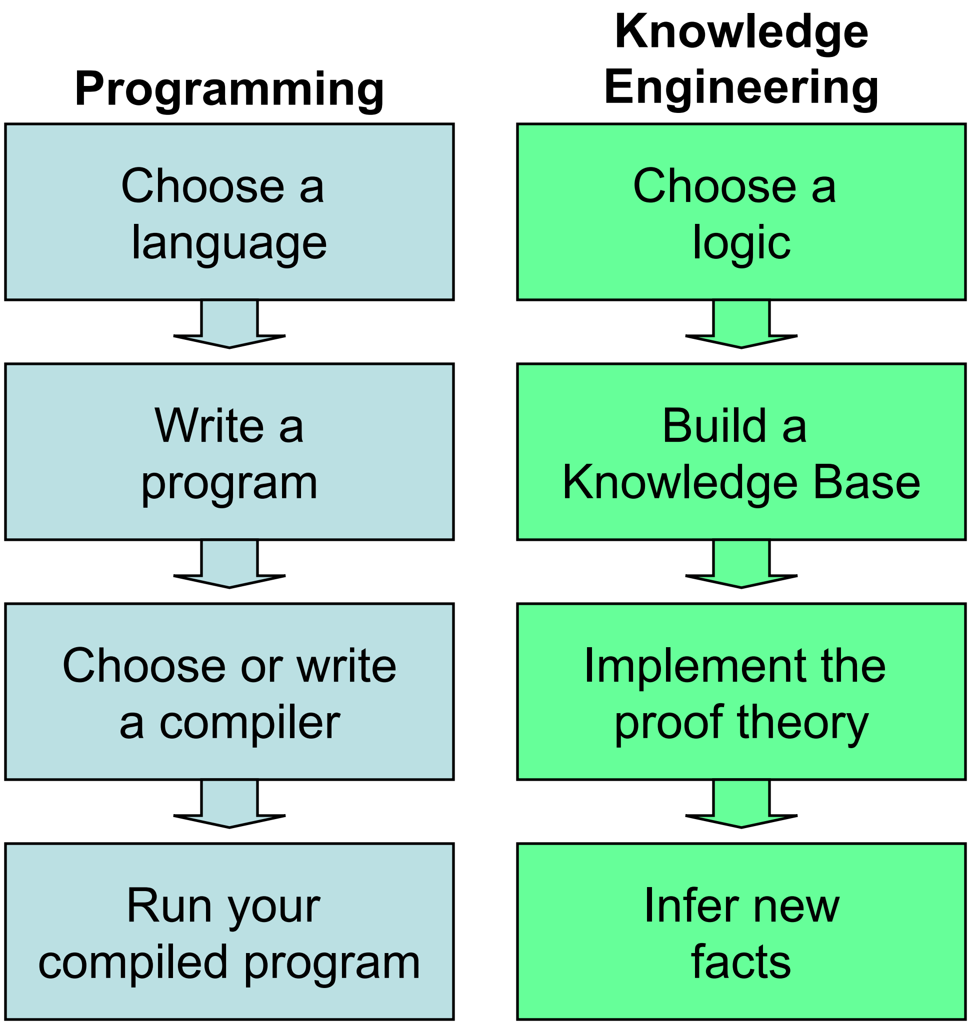
Substances

- 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