Artificial Intelligence -- 15-381
Propositional and First-Order Logic

Jaime Carbonell
10-February-2000

OUTLINE:

- Non-monotonic Inference in Logic
- Semantic Networks
- Inheritance as Search
- Frames, Facets and Procedural Attachment

Beyond Standard Logic:
Non-Monotonic Reasoning

Inferential Monotonicity Property

```
Forall (P,Ai),  {A1, ..., An} |- P =>
              {A1, ..., An, An+1} |- P
```

$$\text{Forall } (P, A_i),\ \{A_1, \ldots, A_n\} \vdash P \Rightarrow \{A_1, \ldots, A_n, A_{n+1}\} \vdash P$$

Monotonic Systems

- Respect inferential monotonicity.
- FOL with *modus ponens* or resolution is *monotonic*
- Strict semantic net inheritance is *monotonic*

Non-Monotonic Systems

- Do not respect inferential monotonicity.
- Default reasoning is *nonmonotonic*
- Semantic net inheritance with exceptions is *nonmonotonic* (typical kind of inheritance)
- Metaknowledge inference is *nonmonotonic*

Example: LOK Inference

1. Items of type T_a if true are normally in KB.
2. Proposition $t_{i,a}$ of type T_a is queried, but not resolvable with KB.
3. Therefore t_a is presumed false, unless/until later proven true.

Semantic Networks

Definition

<$N_{instances}$,$N_{classes}$,L,I,C(I)>, where:

$N_{instances}$ A set of nodes with individual denotations, such as "Clyde", "box-1", "AI core"

$N_{classes}$ = A set of nodes with set or class denotations, such as "CMU students", "Punic War Vets", "Elephants"

L = A set of links (binary relations) over N, such as "brother-of", "eats", "likes"

I = Inheritance relations over N and L, such as "isa", "part-of"

C(I) = Properties of each inheritance relation, e.g. "isa" is transitive, asymmetric (general-to-specific), non-reflexive, and potentially agglomerative.

Uses of Semantic Nets

- Coding static world knowledge
- Built-in fast inference method (inheritance)
- Localization of information

Inheritance in Semantic Networks

Search method

- Depth-first (good iff unique answer)
- Breadth-first (maximally-local answer)
- Preferred-parent-first search (greedy HC)
- All-branch search (agglomerate all closest answers)
- Complete-search (know when answer is exception to the norm)

Coping with "Contradiction"

- Ignore it: stop at first answer
- Default overrides on individual paths
- Report all answers & paths (buck doesn’t stop here)
- Explicit cancel-link semantics (e.g. NETL)
- Resource-bounded search (whatever you can find in < N steps)

Frames in Knowledge Representation

PHILOSOPHY:

Frame system = Semantic Net +
structured nodes +
procedural attachment

INFERENCE PROCESSES:

- Inheritance
- Procedural attachment (demons)
- Frame Matching (a type of unification)

HISTORY:

- Minsky, 1975 (first ideas)
- Bobrow & Winograd, 1977 (KRL)
- By 1980 in wide-spread use (FRL, SRL, Units)
- By 1985 in robust packaged form (CRL, KEE, FrameKit, ...)
- By 1990 in general use for knowledge bases, and evolved into object-oriented data bases (OODBs)

Frames =
Semantic Networks +
Meta-structured Nodes +
Procedural Attachment

```
FRAME   SLOT             FACET            FILLER

[PC     [isa             [value           COMPUTER]]
        [manufacturer    [type-r          COMPANY]]
        [retail-price    [puller          (* &markup
                                             &wholesale)]
                         [range-min       500]
                         [range-max       10000]
                         [unit            USD]]
        [markup          [value           1.5]]
        [owner           [type-r         LEGAL-ANIMATE]]]

[DELL-150/L
        [isa             [value            PC]]
        [manufacturer    [value           DELL]]
        [processor       [value           pentium-150]]
        [wholesale      [value            1400]]]
```

Procedural Attachment

Types of Attachments

- ***Pullers*** (aka "if needed" demons)
  calculate values on demand (optionally cache)
  Caveat: *control forward cascade*
- ***Pushers*** (aka "if added" demons)
  propagate values through network
  Caveat: *check for loops*
- ***If Referenced***
  Tally, check cache validity, ...
- ***If deleted***
  Part of truth-maintenance system
  Check for cache-validation
- ***If changed*** = if deleted + if added

Episodic Knowledge and Scripts

Semantic vs Episodic

- Events vs Facts
- Temporal and Causal sequences
- Use Semantic memory as component

Scripts

- Causally-connected event sequence
- Generalized by alternate paths
    - Tree or DAG structure
    - Conditionals on branches
- Script-role generalization
    - Constants => Typed variables with restrictions
    - Climb a frame hierarchy

Script Application Process

- Match Trigger events, including roles
- Instantiate forwards and backwards ruling out alternate branches
- Interpolation inference (abduction)
- Extrapolation inference (prediction)

Types of Inference

Deduction

- If A, A => B, then B (modus ponens)
- Truth-preserving, formal reasoning
- Examples: inheritance, modus ponens, resolution
- Used in: Proofs, logic, logical argument
- *deductive closure*: everything deducible from premises
- Requires retraction upon contradiction
  IFF non-monotonic.
- Resolution is truth-conditional equivalent to deduction with transitive closure.

Abduction

- If B, A => B, then perhaps A
- Seeks "explanation" for B being true
- Given *closed-world hypothesis*
  Abduction = Deduction (in reverse).

  `E.g. B, A --> B,  C --> B, then A v C`
- *abductive closure of A_i*:
  all legal explanations for A_i.
- Requires retraction upon contradiction.

Types of Inference [Continued]

Induction

- If P(A), P(B), P(C)...,
  then Forall(x) P(x)
- If P(A,A), P(B,B), ~P(A,B)...
  then Forall(x) P(x,x)
- Falsity-preserving inference
- *Generalization from instances*
- Used in Machine Learning
- Requires retraction upon contradiction
  (e.g. if ~P(Boo,Boo) retract P(x,x))

Analogy

- ```
  If [P(A) => P(B)] & [R(A) => R(B)]
   Then perhaps Q(A) --> Q(B)
  ```
- Analogy = Induction + Deduction
- Preserves neither truth nor falsity
- Yet, very useful:
  - argumentation and rhetoric
  - education and explanation
  - insight for scientific discovery
  - case-based reasoning & planning
  ```