Background on Computation/AI

Agent acting intelligently in its own environment (†)

- Actions appropriate for goals
- Flexibile to changing environments & goals
- Learns from Experience
- Appropriate choices for limitations, finite computation

Symbol-System Hypothesis

> A physical symbol system has the necessary and sufficient means for general intelligent action

- Necessity: Anything capable of intelligent action is a physical symbol system
- Sufficiency: Any (sufficiently sophisticated) PSS is capable of intelligent action
- Reasoning is Symbol Manipulation
- Symbols are `1` and `0` of computers, in which case it only means intelligence can be digitized

Doubts

- The brain is not merely a computer, *computation* is not a complete model for intelligence

Church Turing Thesis and its relevance to AI

- Any symbol manipulation possible on a Turing Machine
- Combined with *SSH*, any Intelligent Action possible on a Turing Machine
- Computation turns into Graph Search (express problem as a graph)

Turing machine

A theoretical machine to reason about computation. Reads and writes symbols to and from a tape (†'s environment). A 6-tuple of:

- Σ : Alphabet of Symbols
- Q : Set of possible internal states
- $Q_0 \in Q$: Initial State
- $\epsilon \in \Sigma$: Blank symbol
- A : Accepting/Final States

- $\delta \subseteq (Q \setminus A \times \Sigma) \times (Q \times \Sigma \times \{L, R\})$: Relation on State-Symbol pairs, mapping to State-Symbol-Left/Right Movement

Example: $(q, \sigma), (r, \alpha, L)$ would enocde, if in state q and a σ is read, move to state r, write an α to the tape, and move Left.

- **Deterministic Turing Machine (DTM)** has one State-Symbol-Left/Right triple per State-Symbol pair (it's a function)
- **Non-Deterministic Turing Machine (NTM)** can have more than on S-S-LR per S-S pair.
- †: The environment is the tape, and the agent is the TM. If the TM reaches a state in A, then the action of the agent has been complete/solution to given problem found.

Non-determinism

- Turing machine defines next state depending on current state, read symbol. An NTM has more than one S-S-LR triple per S-S pair.
- With computation as graph search

  ```
  search(Node) :- goal(Node).
  search(Node) :- arc(Node, Next), search(Next).
  ```

  there may exist more than one $Next$ for some $Node$.
- **Don't know (Prolog does this)** : If one choice doesn't lead to solution another might
  - Choose
- **Don't care (Paralog)** : If one selection doesn't lead do a solution, no point in trying others
  - select

†: Problem may be a non-deterministic one, and so needs an NTM to implement the Intelligent Agent to solve it feasibly.

P vs NP

Cobham's Thesis

Feasible computation is defined as being solved by a Deterministic Turing Machine in Polynomial time (it's in P).

$P :=$ {problems solved by a Deterministic Turing Machine in Polynomial Time}

$NP :=$ {problems solved by a Non-deterministic Turing Machine in Polynomial Time}

P is clearly $\subseteq NP$ as all DTMS can just be an NTM with only one triple in its δ relation (in which δ then defines a function).

SAT

For some boolean expression ϕ, of variables $x_1, x_2, \cdots, x_n$, find an assignment makes ϕ evaluate to True. Checking that it's True in P is easy, linear in n. Finding is hard due to non-determinism, many assignment to check.

- **Cook-Levin Theorem** says SAT $\in P \iff P = NP$.
- **CSAT** : ϕ is a *conjunction* of *clauses* where a *clause* is a disjunction of *literals* and a literal is either a non-negated variable x_i (positive) or a negated variable $\neg x_i$ (negative).
- **k-SAT** : Says each clause has k literals
- **3-SAT** : Says each clause has 3 literals. Is as hard as SAT, but 2-SAT is in P.
  - **horn-SAT** : A conjunction of *horn* clause s, where a horn clause has at most 1 positive literal. Linear.

Halting Problem

Given a program P, and data D return 1 if P halts on D, otherwise 0 (if it loops indefinitely). It is undecidable.

Proof (by contadiction)

- Assume for contadiction's sake $\exists$ program $halt(P, D)$ that returns 1 iff P halts on D, otherwise 0.

- Now construct a new program/string Z

  ```
  def Z(String x)
      if halt(x, x) then
           loop forever
      else
          halt
      end
  end
  ```

  and run it on itself, i.e. `Z(Z)`. There are 2 cases:

1. Z halts on Z. Then the call to `Halt(Z, Z)` will return `True`, so Z loops forever on Z.
   - Contradiction
2. Z loops forever on Z. Then the call to `Halt(Z, Z)` returns `False`, so Z halts on Z
   - Contradiction.

- Conclude that *halt* cannot exist, so there is no general method to decide if some P will halt on some D.

Prolog consequence is that there's no general algorithm to detect loops caused by KBs such as

```
p :- q.
q :- p.

a :- a.
```

etc.

Church-Turing Thesis

> A function is effectively calculable if its values can be found by some purely mechanical process.

Halting Problem implications:

Can define functions which are not computable.

- `busy_beaver(n)`: given a TM with n possible states, how many symbols can it write before halting when run with no input?

Can't get an upper bound on this without solving the halting problem. Since *CTT* says you can compute anything on a *TM* this is uncomputable by any method.

Cantor's Theorem

$\forall\omega, |2^\omega| > |\omega|$

TODO: Prove this

Knowlege Representation and Reasoning

Representation and Reasoning System

Definitions

- **Formal Language** : legal sentences
- **Semantics** : meaning of the symbols
- **Reasoning theory/proof procedure** *nondeterministic* specification for how to produce and answer

Implementation

- **Language Parser** : Sentences → Data Structures
- **Reasoning Procedure** : implementation of reasoning theory, search strategey
  - Does *not* reflect semantics (It's a symbol-system manipluation)

Datalog

propositional definite clause : one of these?

- variable : starts with upper case
- constant : starts with lower case, or is a numeral
- predicate symbol : stars with lower case
- term : variable or a constant
- **atomic symbol (atom)** : p or $p(t_1, \cdots, t_n)$, with p a predicate and each t_i is a term.
- **definite clause** : $a \leftarrow b_1 \wedge \cdots \wedge b_m$
- **query** : $?b1 \wedge \cdots \wedge bm$
- **knowlege base** : set of definite clauses

Semantics

Meaning of sentences in a language

Interpretation

What is in the world, symbol-to-real-things-and-relations correspondence

Triple $I = \langle D, \phi, \pi \rangle$

- **D** : Nonempty domain set. Elements are *individuals*

- ϕ : mapping each constant to an individual. A constant c denotes an individual $\phi(c)$
- π : maps each to n-ary predicate symbol a relation
  - ie $\pi : D^n \mapsto \{\text{True}, \text{False}\}$

Notes

- D can be actual real things, not confined to being in a computer.
- $\pi(p)$ specifies truthiness for the predicate symbol p, for each n-tuple of individuals
- if p has no arguments, then $\pi(p)$ either True or False.

Truth in Interpretation

- *c denotes in I* the individual $\phi(c)$.
- **Ground** (variable free) atom $p(t_1, \cdots, t_n)$, in **Interpretation** I is
  - **True** if $\pi(p)(t'_1, \cdots, t'_n) = \text{True}$
  - **False** if $\pi(p)(t'_1, \cdots, t'_n) = \text{False}$
    * Where t_i denotes t'_i in interpretation I
- Ground clause $h \leftarrow b_1 \wedge \cdots b_m$ is **False in** I if h is False in I and each b_i is True in I.
  - Otherwise it's $True$ in I.

Models, Logical Consequence

- A Knowledge Base KB is True in interpretation I iff every clause in KB is True in interpretation I iff every clause in KB is True in I.
- a **model** is an interpretation in which all clauses are True
- g is a **logical consequence** of KB ($KB \models g$), if g is True in every models of KB.
  - ($KB \models g$) if there's no I such that KB is True $\wedge$ g is False.

For Users

1. Come up with an **intended interpretation** I
   - the problem domain
2. Pick constants for the relevant individuals
   - e.g. `shibe` for your pet
3. Pick a predicate symbol for the relations
   - `is_dog` to denote a constant's individual being a dog
4. Tell it things that are True in I
   - build up the knowlege base by **axiomatizing the domain**)
   - `is_dog(X) :- barks(X).`
   - `barks(shibe).`, etc.

5. Ask it things
    - `? is_dog(shibe)` $\rightarrow$ `yes`.
6. Now if $KB \models g$, then g must be True in I
    - Your pet must indeed be a dog.

For Computers

- Knows nothing about the interpretation, only KB.
    - What's a dog? What is barks?
- *Can* determine if some g is an LC of KB, so if $KB \models g$, then it's True in I.
- If $\neg(KB \models g)$, then $\exists$ some Interpretation in which g is False. This could be the intended interpretation I.

Proofs

Search

Basic graph searching (Depth first)

```
arc(Node, Next) :- % Something that relates Node to Next.
search(Node) :- goal(Node).
search(Node) :- arc(Node, Next), search(Next).
```

- Nondeterminism if `arc/2` has multiple solutions
    - Choose the best one (A*, Best first, etc)
- Computation eliminates the non-determinism
- Can bound the number of calls to `arc`, the number of search iterations.

Frontier Search

- With a **graph**, **start nodes** and **goal nodes**, incrementally explore paths from start nodes, hoping to reach goal nodes.
- Maintain a **frontier** of paths from start that have been explored.
- Search is complete once frontier hits a goal.
- How the frontier expands (how the child nodes of the current frontier are inserted to the frontier) can vary, defines the **search strategy**

```
frontier := { s : s is a start node }
until frontier.empty
    select and remove path <n_0, ... ,n_k> from frontier;
    if goal(n_k)
        return <n_0, ... , n_k>;
    end
```

```
    for each neighbor n of n_k;
        add <n_0, ... , n_k, n> to frontier
    end
end
```

Prolog Examples

```
search(Start) :- frontier_search([Start]).

frontier_search([Node|_]) :- goal(Node).
frontier_search([Node|Rest]) :-
    findall(Next, arc(Node, Next), Children),
    add_to_frontier(Children, Rest, New),  % add the neighbours to the frontier
    frontier_search(New).
```

The strategy depends on how `add_to_frontier/3` is defined:

Depth-first, empty the `Children` before adding the `New`:

```
add_to_frontier([], Rest, Rest).
add_to_frontier([H|T], Rest, [H, New]) :- add_to_frontier(T, Rest, New)
```

Breadth-first, empty the `New` before adding the `Children`:

```
add_to_frontier(Children, [], Children).
add_to_frontier(Children, [H|T], Children) add_to_frontier(Children, T, New).
```

Bounded Depth First by adding an iteration-counting term:

```
bounded_search(Node, _) :- goal(Node).
bounded_search(Node, s(B)) :- bounded_search(B).
```

Iterative deepening is depth first with a variable bound:

```
iterative_deepening(Node) :- bound(B), bs(Next, B).

bound(0).
bound(s(B)) :- bound(B). % easily adjustable
```

Knowlege Representation and Reasoning

Representation and Reasoning System

Definitions

- **Formal Language** : legal sentences

- **Semantics** : meaning of the symbols
- **Reasoning theory/proof procedure** *nondeterministic* specification for how to produce and answer

Implementation

- **Language Parser** : Sentences $\rightarrow$ Data Structures
- **Reasoning Procedure** : implementation of reasoning theory, search strategey
  - Does *not* reflect semantics (It's a symbol-system manipluation)

Datalog

propositional definite clause : one of these?

- variable : starts with upper case
- constant : starts with lower case, or is a numeral
- predicate symbol : stars with lower case
- term : variable or a constant
- **atomic symbol (atom)** : p or $p(t_1, \cdots, t_n)$, with p a predicate and each t_i is a term.
- **definite clause** : $a \leftarrow b_1 \wedge \cdots \wedge b_m$
- **query** : $?b1 \wedge \cdots \wedge bm$
- **knowlege base** : set of definite clauses

Semantics

Meaning of sentences in a language

Interpretation

What is in the world, symbol-to-real-things-and-relations correspondence

Triple $I = \langle D, \phi, \pi \rangle$

- **D** : Nonempty domain set. Elements are *individuals*
- ϕ : mapping each constant to an individual. A constant c denotes an individual $\phi(c)$
- π : maps each to n-ary predicate symbol a relation
  - ie $\pi : D^n \mapsto \{\text{True}, \text{False}\}$

Notes

- D can be actual real things, not confined to being in a computer.
- $\pi(p)$ specifies truthiness for the predicate symbol p, for each n-tuple of individuals
- if p has no arguments, then $\pi(p)$ either True or False.

Truth in Interpretation

- c *denotes in* I the individual $\phi(c)$.
- **Ground** (variable free) atom $p(t_1, \cdots, t_n)$, in **Interpretation** I is
  - **True** if $\pi(p)(t'_1, \cdots, t'_n) = \text{True}$
  - **False** if $\pi(p)(t'_1, \cdots, t'_n) = \text{False}$
    - Where t_i denotes t'_i in interpretation I
- Ground clause $h \leftarrow b_1 \wedge \cdots b_m$ is **False in** I if h is False in I and each b_i is True in I.
  - Otherwise it's $True$ in I.

Models, Logical Consequence

- A Knowledge Base KB is True in interpretation I iff every clause in KB is True in interpretation I iff every clause in KB is True in I.
- a **model** is an interpretation in which all clauses are True
- g is a **logical consequence** of KB ($KB \models g$), if g is True in every models of KB.
  - ($KB \models g$) if there's no I such that KB is True $\wedge$ g is False.

For Users

1. Come up with an **intended interpretation** I
   - the problem domain
2. Pick constants for the relevant individuals
   - e.g. `shibe` for your pet
3. Pick a predicate symbol for the relations
   - `is_dog` to denote a constant's individual being a dog
4. Tell it things that are True in I
   - build up the knowlege base by **axiomatizing the domain**)
   - `is_dog(X) :- barks(X).`
   - `barks(shibe).`, etc.
5. Ask it things
   - `? is_dog(shibe)` $\rightarrow$ `yes`.
6. Now if $KB \models g$, then g must be True in I
   - Your pet must indeed be a dog.

For Computers

- Knows nothing about the interpretation, only KB.
  - What's a dog? What is barks?
- *Can* determine if some g is an LC of KB, so if $KB \models g$, then it's True in I.
- If $\neg(KB \models g)$, then $\exists$ some Interpretation in which g is False. This could be the intended interpretation I.

Proofs

- **Proof** : mechanical derivation that formula *follows* from KB.
  - $KB \vdash g$: g can be derived from KB.
  - $KB \models g$: g is true in *all models* of KB.
- **soundness** : $KB \vdash g \Rightarrow KB \models g$
- **completeness** : $KB \models g \Rightarrow KB \vdash g$

Bottom Up

A *rule of derivarion*, modus ponens *generalisé*.

- Given $h \leftarrow b_1 \wedge \cdots \wedge b_m$ in the KB, if each b_i has been derived, then h van be derived.
- **Forward chaining** on the clause. Also covers $m = 0$.

Procedure

If $g \in C$ at the end, then $KB \vdash g$.

```
C := {}
until (no more clauses can be selected) {
    select clause h <- b_1 && ... && b_n such that
        b_i in C for all i
        and h not in C ;
    C := C union {h};
}
```

Proof of Soundness:

$KB \vdash g \Rightarrow KB \models g$

- Suppose $\exists g : KB \vdash g \wedge \neg(KB \models g)$.
- Let h be the first atom added to C, which is not true in every model of KB.
- Suppose h is not true in model I of KB.
- There must be some clause $h \leftarrow b_1 \wedge \cdots \wedge b_m$

- Now each b_i is True in I. h is False in I, so the clause is False in I.
- So I can't be a model of KB.
- This is a contradiction, so no such g exists.

Fixed point

C, at the end, is a **fixed point**.

Now if we let I be the interpretation such that every element of the *fixed point* is True, and every other atom is False, then I is a model of KB.

- Suppose $h \leftarrow b_1 \wedge \cdots \wedge b_m \in KB$ is False in I. Then h is False, and each b_i is true in I. So we can, by the method, add h to C.
- This contradicts C being a fixed point.
- The I is a **minimal model**.

Proof of Completeness

- $KB \models g \Rightarrow KB \vdash g$.
- Suppose $KB \models g$. Then g is true in all models of KB. Thus g is true in the minimal model.
- Thus g is generated by the bottom up algorithm.
- Thus $KB \vdash g$.

Top Down

- Go back from a query to see if it's a logical consequence of the KB.
- **Answer Clause** α is Yes $\leftarrow a_1 \wedge \cdots \wedge c_m$
- **SLD Resolution** of α with atom a_i is α with a_i substituted for the clauses of a_i
- **Answer** is an answer clause with $m = 0$, i.e. Yes $\leftarrow$.
- **derivation** is a sequence of answer clauses, $\gamma_0, \gamma_1, \cdots, \gamma_n$
  - Each γ_i is the resolution of γ_{i-1} with some clause in the KB.
  - γ_n is the 'final' answer.

Procedure

To solve *ac*.

```
ac := "yes <- q_1 && ... && q_k"
until ac is an answer (i.e. until ac matches "yes <-"
    select conjunct a_i from the body of a_c;
    choose C from the KB that has a_i at its head;
```

```
    replace a_i in body of ac with body of C;
end
```

There's nondeterminism in the choice of C here:

- **Don't care** : If one doesn't lead to solution, none of the others will.
- **Don't know** : If one choice doesn't lead to the solution, others might.

Knowlege Representation

- How to represent *"Coco is a Shiba Inu"*
- Could do something like `shibe(coco)`
    - "Who are the shibes?" easy to solve
- Or `breed(coco, shibe)`
    - "What breed is Coco"
    - "What dogs are the Shiba Inu"
    - ~~"What property is"Shiba Inu¿'~~ It's a `breed`, but we can't resolve this easily.
- Solution: `prop(coco, breed, shibe)`. Now we can solve it all with the usual strategies. Called **object-attribute-value** representation.
    - `prop(coco, is_a, shibe)`
    - `prop(coco, shibe, true)`
- **Reification** : translating a scenario into object.

Frames

This can all be brought into a **frame**, collection of attribute-value pairs:

```
[ owned_by = craig
, deliver_to = ming
, model = lemon_laptop_1000 ...  ]
```

etc.

Relations

- **Primitive knowlege** : defined explicity
- **Derived knowlege** : defined by rules

With an `is_a` attribute, we can do **property inheritance**. Every individual in a class has n for some attribute p.

No reason not to allow **multiple inheritance**, where an object is a member of multiple classes. There can be conflicts, for example if both classes define a different default for some property (*multiple inheritance problem*).

Associate most general class with an attribute, don't add properties willy-nilly, and axiomatize in the causal direction.

Complete Knowlege Assumption (CKA)

> Any fact not listed in a Knowlege Base is False

The definite clause system is **monotonic**, that is, if we add a clause, it doesn't cause other clauses to be false. (It doesn't invalidate previous conclusions)

Adding the CKA, the system is **non monotonic**; we *can* invalidate a conclusion by adding more clauses.

Example

```
student(mary).
student(john).
student(ying).
```

- Under the CKA, this means $\text{student}(X) \iff X = \text{mary} \vee X = \text{john} \vee X = \text{ying}$
- So to prove that $\neg\text{student(alan)}$, you need $\text{alan} \neq \text{mary} \wedge \text{alan} \neq \text{john} \wedge \text{alan} \neq \text{ying}$
    - Need unique names assumption

Clarke Completion

```
mem(X, [X|T]).
mem(X, [H|T]) :- mem (X, T).
```

becomes

```
mem(X, Y) <=> (eT Y == [X | T]) or
              (eH eT T == [H | T] and mem(X, T)).
```

Where `e` *is* $\exists$

- Completion of every predicate, and equality/inequality axioms.
- If p in the KB is defined by no clauses, then it completes to $p \leftrightarrow$ False. i.e. $\neg p$.
- **Negation as Failure** : Interpret negations in clause bodies. $\sim p$ means p is False under CKA.

Bottom up Negation as Failure proof Procedure

```
C = {};
until no more selections possible
    either
        select "h <- b1 or ... or b_n" in KB such that
            bi in C for all i, and h not in C;
        C = C union {h}
    or
        select h such that
            for each "h <- b1 or ... or b_n" in KB
                either
                    exists b_i such that ~b_i in C
                or
                    exists b_i such that b_i = ~g, and g in C
                end
        C = C union {~h}
    end
end
```

- If this procedure fails, then $\neg a$ can be concluded.
- Say we have $a \leftarrow b_1, \cdots, a \leftarrow b_n$,, need all of them to fail. It needs to be *finite* however, like $p \leftarrow p$ is not decidable with bottom up.
  - Halting problem means these things are undetectable in the general case too!
- If trying to NAF a query that has unbound variables, the NAF must be **delayed** until the variable is bound, otherwise it **flounders**.

Integrity Constraints

- false $\leftarrow a_1 \wedge \cdots \wedge a_k$
  - With a_i atoms, false an atom that's False in all interpretations.
- **Horn Clauses** are either definite clauses or integrity constraints.
- $\neg\alpha$ is a formula which is
  - True in I if α is False in I and
  - False in I if α is True in I

Suppose we have KB

- false $\leftarrow a \wedge b$
- $a \leftarrow c$
- $b \leftarrow c$
  - Then $KB \models \neg c$

Can also be **disjuctive conclusions**:

- false $\leftarrow a \wedge b$
- $a \leftarrow c$
- $b \leftarrow d$

- Then $KB \models \neg c \vee \neg d$

Questions and Answers

- **assumable** : an atom whose negation is acceptably included in the (disjuntive answer)
    - Hand-wavily 'assumably' true
- **conflict** : a set of assumables which imply *False* for a KB
- **minimal conflict** : a conflict with no strict subsets that are also conflicts
- **consistency-based diagnosis** : set of assumables that has one element in each conflict
- **minimal diagnosis** : no strict subset is a diagnosis

Bottom Up Conflict Finding

- **conclusion** : a pair $\langle a, A \rangle$ with a an atom, and A a set of assumbables which imply a.

Rules and Consistency

- g is KB-**consistent** if it's true in some model of KB.