Knowledge Representation and Reasoning

COMP3431 Robot Software Architectures
A Three-Level Architecture

- **Game – Controller**
  - **Roles** (Robot-behaviour, eg Goalie, Striker, Supporter)
  - **Skills** (localise, position, find-ball, track-ball, get-behind-ball, kick-ball, getup)
  - **State Estimation** (Field-State + Robot-State)
  - **Sensing** (cameras, IR, hall-effect, joint-angles, foot-sensors, IMU, bumpers, sonar, wireless)
  - **Actuation** (head-movement, leg/arm-movement, camera-switching, speakers, LEDs, wireless)

- **Environment** (this robot, ball, other robots)
Where are we now?

• We’ve done a whirlwind tour of perception and action
• Now moving up to planing and problem solving
• and the kind of learning that goes with them
Why do we need symbols?

• How do we ask “where is Tim’s office”?

• How do we know that if we want to get a cold drink, we should find the fridge and it’s probably in the kitchen?
Automated Reasoning

• Expressions in a formal language conform to unambiguous rules of construction.

• Inferences are drawn by following strict laws for manipulating expressions in a formal language.

• The language we use most often is clausal form logic.
Propositional Calculus

- A propositional constant is a symbol (like p, q, r, ...) that stands for some like “Sydney is a city”.

- Propositions are atomic formulae.

- A well-formed (wff) formula is
  - an atom, \( \Psi \)
  - the negation of a wff, \( \neg \Psi \)
  - the disjunction (or) of a pair of wffs, \( \Psi \lor \Phi \)
  - Everything else can be derived
Derived Expressions

- $\Psi \land \Phi$ is defined as $\neg(\neg\Psi \lor \neg\Phi)$
- $\Psi \supset \Phi$ is defined as $\neg\Psi \lor \Phi$
- $\Psi \equiv \Phi$ is defined as $(\Psi \supset \Phi) \land (\Phi \supset \Psi)$
Predicate Calculus

• Propositional calculus cannot deal with statements of generality like,

  'All men are mortal'

• To do this, we need predicates, arguments, variables and quantifiers. eg.

  \((\forall X)(\text{man}(X) \supset \text{mortal}(X))\)
Clausal Form

• In clausal form, positive literals are placed to the left of an arrow symbol and negative atoms to the right, e.g.

\[ p,q \leftarrow p \]
\[ p,q \leftarrow q \]

• In general, a clause is an expression of the form:

\[ p_1, \ldots, p_m \leftarrow q_1, \ldots, q_n \]

• The literals on the left are disjoined conclusions.

• The literals on the right are conjoined conditions.
A Horn clause is one which only has a single positive literal, eg.

\[ p_1 \leftarrow q_1, \ldots, q_n \]

The programming language, Prolog, consists of Horn clause definitions, eg.

\begin{verbatim}
on(a, b).
on(b, c).
above(X, Y) :- on(X, Y).
above(X, Y) :- on(Z, Y), above(X, Z).
\end{verbatim}
Resolution

- To prove $p$ follows from some theory, $T$, assume $\neg p$ and then try to derive a contradiction from its conjunction with $T$.

- Resolution requires a pattern matching operation, called *unification*.

- When matching literals, we look for variable substitutions that will make the two expressions identical. Eg.

  \[
  \text{runs\_faster\_than}(X, zeno) \\
  \text{runs\_faster\_than}(\text{tortoise}, Y)
  \]

  are identical under the substitution \{X/tortoise, Y/zeno\}
Resolving Clauses

• A clause that contains no variables is called a ground clause.

• To resolve two non-ground clauses, you must find a unifier for complimentary literals. Eg.

  \{\text{beats\_in\_race}(X, \ zeno), \ \neg \text{younger\_than}(X, \ zeno)\}

and

  \{\neg \text{beats\_in\_race}(\text{tortoise}, \ Y), \ \neg \text{philosopher}(Y)\}

have unifier \text{n} = \{X/\text{tortoise}, \ Y/\text{zeno}\} and generate the resolvent

  \{\neg \text{philosopher}(\text{zeno}), \ \neg \text{younger\_than}(\text{tortoise}, \ \text{zeno})\}
• We can prove a formula, $p$, if we can derive it from a theory, $T$, by a sequence of resolution steps.

  Written as $T \vdash p$.

• If the theory is very large, there may be many ways of deriving a proof.

• How can we find a short derivation?

• We try a proof by refutation, ie. add negation of goal to theory and show that the new theory is inconsistent, ie. implies false.

• The empty clause, {}, is interpreted as false. So if theory derives false, we have an inconsistent theory.
A Prolog Proof Tree

:- above(a, c).

above(X, Y) :- on(Z, Y), above(X, Z).

:- on(b, c).
above(X, Y) :- on(X, Y).
above(X, Y) :- on(Z, Y), above(X, Z).

:- above(a, b).

on(a, b).

:- on(a, b).

on(b, c).

:- on(Z, c), above(a, Z).

on(a, b).

:- on(a, b).

above(X, Y) :- on(X, Y).

on(a, b).

{}
Resolution Search

• Resolution uses backward chaining to focus search for clauses to resolve.

• There are many refinements to this search.

• We will stick to the Prolog method which resolves clauses and their literals in input order, ie, top-to-bottom, left-to-right.
Soundness and Completeness

• A proof procedure is sound if every formula it derives is true. I.e. it cannot prove something it shouldn't.

• A proof procedure is complete if it can derive every thing that is possible to derive from a theory. I.e. There is no true statement that it cannot prove.

• Decidability means that we can always show if a proposition follows from a theory.

• Prolog's proof procedure is sound and complete for Horn clauses.

• Unrestricted first-order logic is undecidable.