Introductory Lecture

Barry Hurley

National University of Ireland, Galway.

Layout of Talk

1. What is Artificial Intelligence?

Layout of Talk

1. What is Artificial Intelligence?
2. Breaking down the problem

Layout of Talk

1. What is Artificial Intelligence?
2. Breaking down the problem
3. Computer Conversations

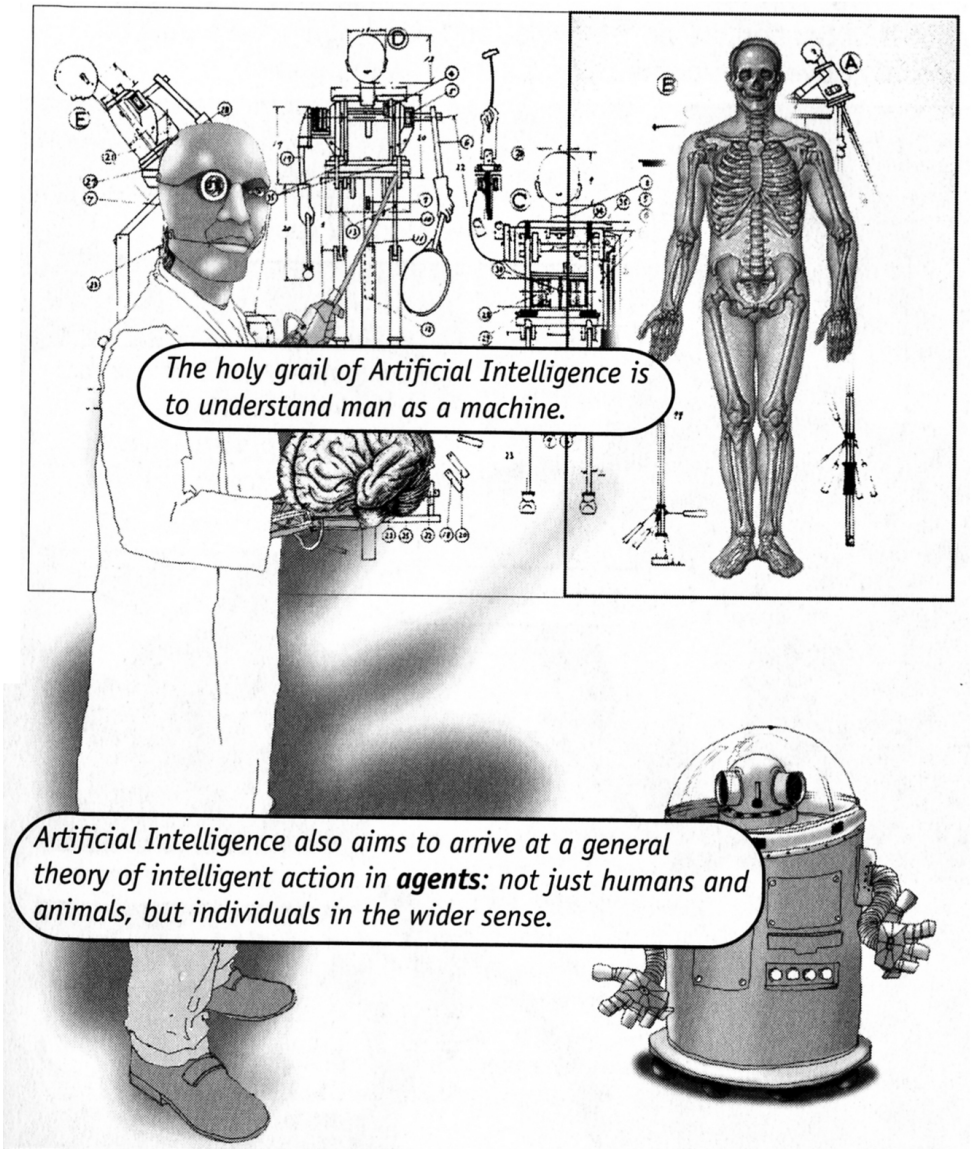
Layout of Talk

1. What is Artificial Intelligence?
2. Breaking down the problem
3. Computer Conversations
4. Computer Translation

Layout of Talk

1. What is Artificial Intelligence?
2. Breaking down the problem
3. Computer Conversations
4. Computer Translation
5. Ruining the fun of games.

1. What is Artificial Intelligence?



1. What is Artificial Intelligence?

What is the problem we are actually trying to solve?

1. What is Artificial Intelligence?

What is the problem we are actually trying to solve?

Let's first define intelligence:

1. What is Artificial Intelligence?

What is the problem we are actually trying to solve?

Let's first define intelligence:

> *Intelligence is the computational part of the ability to achieve goals in the world.*

1. What is Artificial Intelligence?

What is the problem we are actually trying to solve?

Let's first define intelligence:

> *Intelligence is the computational part of the ability to achieve goals in the world.*

Artificial Intelligence

1. What is Artificial Intelligence?

What is the problem we are actually trying to solve?

Let's first define intelligence:

> *Intelligence is the computational part of the ability to achieve goals in the world.*

Artificial Intelligence

> *The study of how computer systems or agents can* **simulate** *intelligent processes such as learning, reasoning, and understanding symbolic information in context.*

1. What is Artificial Intelligence?

What is the problem we are actually trying to solve?

Let's first define intelligence:

> *Intelligence is the computational part of the ability to achieve goals in the world.*

Artificial Intelligence

> *The study of how computer systems or agents can* **simulate** *intelligent processes such as learning, reasoning, and understanding symbolic information in context.*

Our Goal

1. What is Artificial Intelligence?

What is the problem we are actually trying to solve?

Let's first define intelligence:

> *Intelligence is the computational part of the ability to achieve goals in the world.*

Artificial Intelligence

> *The study of how computer systems or agents can* **simulate** *intelligent processes such as learning, reasoning, and understanding symbolic information in context.*

Our Goal

> *For a machine to simulate intelligent processes.*

1. What is Artificial Intelligence?

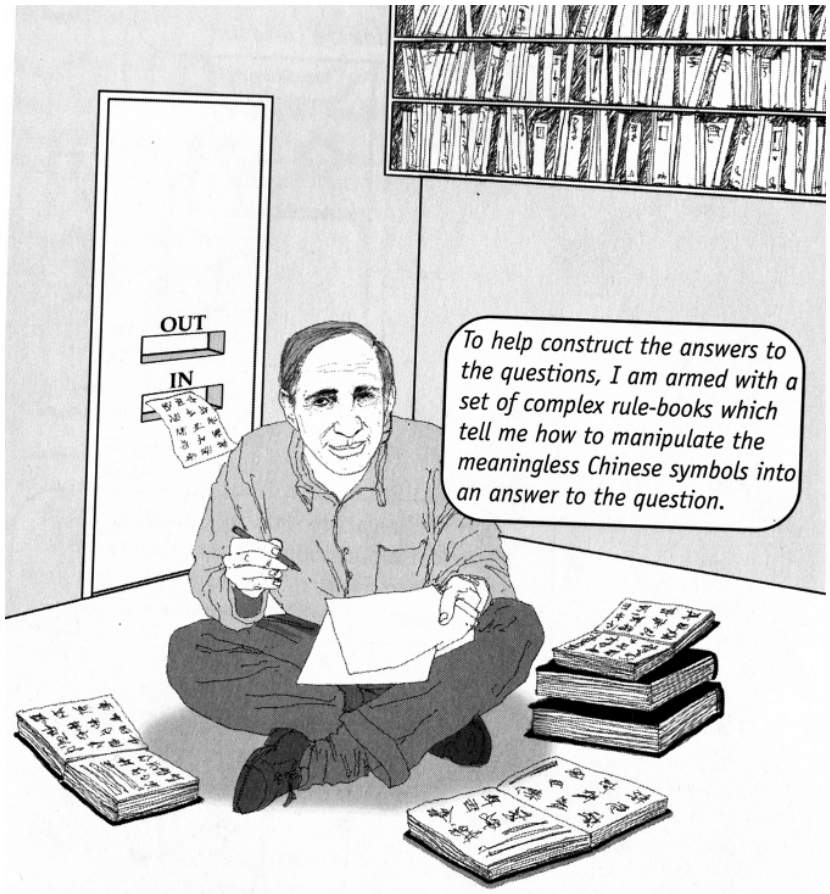
Brings up many philosophical arguments?

1. What is Artificial Intelligence?

Brings up many philosophical arguments?
E.g. Can a machine think?

1. What is Artificial Intelligence?

Brings up many philosophical arguments?
E.g. Can a machine think?



John Searle's Chinese Room Example

2. Breaking down the problem

Look at Natural Intelligence:

2. Breaking down the problem

Look at Natural Intelligence:

- Computer -vs- Brain

2. Breaking down the problem

Look at Natural Intelligence:

- Computer -vs- Brain
- Both perform logical and mathematical tasks

2. Breaking down the problem

Look at Natural Intelligence:

- Computer -vs- Brain
- Both perform logical and mathematical tasks
- Computer faster at doing logic and computations.

2. Breaking down the problem

Look at Natural Intelligence:

- Computer -vs- Brain
- Both perform logical and mathematical tasks
- Computer faster at doing logic and computations.
- Brain is better at interpreting the outside world and creating new ideas.

2. Breaking down the problem

Look at Natural Intelligence:

- Computer -vs- Brain
- Both perform logical and mathematical tasks
- Computer faster at doing logic and computations.
- Brain is better at interpreting the outside world and creating new ideas.

Bridging the gap

2. Breaking down the problem

Look at Natural Intelligence:

- Computer -vs- Brain
- Both perform logical and mathematical tasks
- Computer faster at doing logic and computations.
- Brain is better at interpreting the outside world and creating new ideas.

Bridging the gap

- understanding of natural language

2. Breaking down the problem

Look at Natural Intelligence:

- Computer -vs- Brain
- Both perform logical and mathematical tasks
- Computer faster at doing logic and computations.
- Brain is better at interpreting the outside world and creating new ideas.

Bridging the gap

- understanding of natural language
- ability to adapt to new situations

2. Breaking down the problem

Look at Natural Intelligence:

- Computer -vs- Brain
- Both perform logical and mathematical tasks
- Computer faster at doing logic and computations.
- Brain is better at interpreting the outside world and creating new ideas.

Bridging the gap

- understanding of natural language
- ability to adapt to new situations
- think for themselves

2. Breaking down the problem

If we break down problem into finite numerical and logical steps then we know a computer can deal with this.

2. Breaking down the problem

If we break down problem into finite numerical and logical steps then we know a computer can deal with this.

Representation of knowledge

2. Breaking down the problem

If we break down problem into finite numerical and logical steps then we know a computer can deal with this.

Representation of knowledge

- relate real-life examples to something a computer can understand.

2. Breaking down the problem

If we break down problem into finite numerical and logical steps then we know a computer can deal with this.

Representation of knowledge

- relate real-life examples to something a computer can understand.
- Logical facts and rules

2. Breaking down the problem

If we break down problem into finite numerical and logical steps then we know a computer can deal with this.

Representation of knowledge

- relate real-life examples to something a computer can understand.
- Logical facts and rules
- Use laws of classical logic

2. Breaking down the problem

If we break down problem into finite numerical and logical steps then we know a computer can deal with this.

Representation of knowledge

- relate real-life examples to something a computer can understand.
- Logical facts and rules
- Use laws of classical logic
- Infer knowledge $\longrightarrow$ Computer acting intelligently

2. Breaking down the problem

Classical Logic

- From knowledge base of facts and rules infer knowledge.

2. Breaking down the problem

Classical Logic

- From knowledge base of facts and rules infer knowledge.

Example

Fact 1: *Mary is female.*

2. Breaking down the problem

Classical Logic

- From knowledge base of facts and rules infer knowledge.

Example

Fact 1: *Mary is female.*

Fact 2: *Mary is a parent of John.*

2. Breaking down the problem

Classical Logic

- From knowledge base of facts and rules infer knowledge.

Example

Fact 1: *Mary is female.*

Fact 2: *Mary is a parent of John.*

Rule: *If a person is a parent and is female then they are a mother.*

2. Breaking down the problem

Classical Logic

- From knowledge base of facts and rules infer knowledge.

Example

Fact 1: *Mary is female.*

Fact 2: *Mary is a parent of John.*

Rule: *If a person is a parent and is female then they are a mother.*

What can we infer from the above facts and rule?

2. Breaking down the problem

Classical Logic

- From knowledge base of facts and rules infer knowledge.

Example

Fact 1: *Mary is female.*

Fact 2: *Mary is a parent of John.*

Rule: *If a person is a parent and is female then they are a mother.*

What can we infer from the above facts and rule?
Mary is a mother.

2. Breaking down the problem

But there are limits to classical logic as regards A.I.

2. Breaking down the problem

But there are limits to classical logic as regards A.I.

Not all knowledge available is certain or clear.

2. Breaking down the problem

But there are limits to classical logic as regards A.I.

Not all knowledge available is certain or clear.

- Ignorance, e.g. limits of our knowledge

2. Breaking down the problem

But there are limits to classical logic as regards A.I.

Not all knowledge available is certain or clear.

- Ignorance, e.g. limits of our knowledge
- Physical Randomness or Indeterminism, e.g. rolling a dice

2. Breaking down the problem

But there are limits to classical logic as regards A.I.

Not all knowledge available is certain or clear.

- Ignorance, e.g. limits of our knowledge
- Physical Randomness or Indeterminism, e.g. rolling a dice
- Vagueness, e.g. someone is classified as a mathematician, or they a group theorist, analyst, etc.?

2. Breaking down the problem

Reasoning with uncertainty

2. Breaking down the problem

Reasoning with uncertainty

- Levels of uncertainty defined by experts, e.g. probabilities, fuzziness

2. Breaking down the problem

Reasoning with uncertainty

- Levels of uncertainty defined by experts, e.g. probabilities, fuzziness
- Bayesian Inference:

2. Breaking down the problem

Reasoning with uncertainty

- Levels of uncertainty defined by experts, e.g. probabilities, fuzziness
- Bayesian Inference:

$$P(A_i \mid B) = \frac{P(A_i)P(B \mid A_i)}{\sum_{k=1}^{n} P(A_k)P(B \mid A_k)}$$

2. Breaking down the problem

Reasoning with uncertainty

- Levels of uncertainty defined by experts, e.g. probabilities, fuzziness
- Bayesian Inference: $$P(A_i \mid B) = \frac{P(A_i)P(B \mid A_i)}{\sum_{k=1}^{n} P(A_k)P(B \mid A_k)}$$
- Fuzzy Logic:

2. Breaking down the problem

Reasoning with uncertainty

- Levels of uncertainty defined by experts, e.g. probabilities, fuzziness
- Bayesian Inference:

$$P(A_i \mid B) = \frac{P(A_i)P(B \mid A_i)}{\sum_{k=1}^{n} P(A_k)P(B \mid A_k)}$$

- Fuzzy Logic:

$\mu_A : X \to [0, 1]$ where
$\mu_A(x) = 1$ if x is totally in A;
$\mu_A(x) = 0$ if $x \notin A$;
$0 < \mu_A(x) < 1$ if x is partly in A.

2. Breaking down the problem

Intelligent isn't only about solving a problem.

2. Breaking down the problem

Intelligent isn't only about solving a problem.

Method is important.

2. Breaking down the problem

Intelligent isn't only about solving a problem.

Method is important.

- Exhaustive search may be inefficient or impossible

2. Breaking down the problem

Intelligent isn't only about solving a problem.

Method is important.

- Exhaustive search may be inefficient or impossible
- Need clever search algorithms

2. Breaking down the problem

Intelligent isn't only about solving a problem.

Method is important.

- Exhaustive search may be inefficient or impossible
- Need clever search algorithms
- Use specialised search trees and heuristics

2. Breaking down the problem

Even if we can do all this, there still is a big gap!

2. Breaking down the problem

Even if we can do all this, there still is a big gap!

Problems

2. Breaking down the problem

Even if we can do all this, there still is a big gap!

Problems

- Halt Tester: Impossible to construct a computer program to successfully determine if every computer program halts.

2. Breaking down the problem

Even if we can do all this, there still is a big gap!

Problems

- Halt Tester: Impossible to construct a computer program to successfully determine if every computer program halts.
- Gödel's Incompleteness Theorem.

2. Breaking down the problem

Even if we can do all this, there still is a big gap!

Problems

- Halt Tester: Impossible to construct a computer program to successfully determine if every computer program halts.
- Gödel's Incompleteness Theorem.
  $\Longrightarrow$ We can't just use logical rules and axioms.

2. Breaking down the problem

Even if we can do all this, there still is a big gap!

Problems

- Halt Tester: Impossible to construct a computer program to successfully determine if every computer program halts.
- Gödel's Incompleteness Theorem.
  $\Longrightarrow$ We can't just use logical rules and axioms.

Conclusion:

There will always be limitations on what a computer can prove?

3. Computer Conversations

Natural Language Processing

3. Computer Conversations

Natural Language Processing

- Originally Americans wanted to translate Russian Scientific Documents.

3. Computer Conversations

Natural Language Processing

- Originally Americans wanted to translate Russian Scientific Documents.
- Thought to be easy problem.

3. Computer Conversations

Natural Language Processing

- Originally Americans wanted to translate Russian Scientific Documents.
- Thought to be easy problem.
- Difficulty lay in ambiguity and knowledge in context.

3. Computer Conversations

Natural Language Processing

- Originally Americans wanted to translate Russian Scientific Documents.
- Thought to be easy problem.
- Difficulty lay in ambiguity and knowledge in context.

Turing Test

3. Computer Conversations

Natural Language Processing

- Originally Americans wanted to translate Russian Scientific Documents.
- Thought to be easy problem.
- Difficulty lay in ambiguity and knowledge in context.

Turing Test

- Human judge asks questions via computer to remote computer and human.

3. Computer Conversations

Natural Language Processing

- Originally Americans wanted to translate Russian Scientific Documents.
- Thought to be easy problem.
- Difficulty lay in ambiguity and knowledge in context.

Turing Test

- Human judge asks questions via computer to remote computer and human.
- Computer & human respond.

3. Computer Conversations

Natural Language Processing

- Originally Americans wanted to translate Russian Scientific Documents.
- Thought to be easy problem.
- Difficulty lay in ambiguity and knowledge in context.

Turing Test

- Human judge asks questions via computer to remote computer and human.
- Computer & human respond.
- Judge should not be able to reliably tell the difference.

3. Computer Conversations

Natural Language Processing

- Originally Americans wanted to translate Russian Scientific Documents.
- Thought to be easy problem.
- Difficulty lay in ambiguity and knowledge in context.

Turing Test

- Human judge asks questions via computer to remote computer and human.
- Computer & human respond.
- Judge should not be able to reliably tell the difference.
- No computer has passed this test to date

3. Computer Conversations

Prolog

3. Computer Conversations

Prolog

- A programming language used for Artificial Intelligence

3. Computer Conversations

Prolog

- A programming language used for Artificial Intelligence
- Can be used in natural language processing (NLP for short).

3. Computer Conversations

Prolog

- A programming language used for Artificial Intelligence
- Can be used in natural language processing (NLP for short).
- Procedural programming languages tell a computer "how to solve a problem".

3. Computer Conversations

Prolog

- A programming language used for Artificial Intelligence
- Can be used in natural language processing (NLP for short).
- Procedural programming languages tell a computer "how to solve a problem".
- Prolog says "what problem we want solved".

3. Computer Conversations

Prolog

- A programming language used for Artificial Intelligence
- Can be used in natural language processing (NLP for short).
- Procedural programming languages tell a computer "how to solve a problem".
- Prolog says "what problem we want solved".
- We merely provide clues to the solution.

3. Computer Conversations

Prolog

- A programming language used for Artificial Intelligence
- Can be used in natural language processing (NLP for short).
- Procedural programming languages tell a computer "how to solve a problem".
- Prolog says "what problem we want solved".
- We merely provide clues to the solution.

We'll now engage in our first conversation with the computer.

3. Computer Conversations - Lion loose at NUIG.

```
smarter(lion, lecturer).
chases(lion, lecturer).
chases(lion, nuig_student).
deadend(nuig_student).
caught(Y) ← chases(X,Y) ∧ smarter(X,Y).
caught(Y) ← chases(X,Y) ∧ deadend(Y).
```

Caught by lion:

3. Computer Conversations - Lion loose at NUIG.

```
smarter(lion, lecturer).
chases(lion, lecturer).
chases(lion, nuig_student).
deadend(nuig_student).
caught(Y) ← chases(X,Y) ∧ smarter(X,Y).
caught(Y) ← chases(X,Y) ∧ deadend(Y).

caught(X)?
```

Caught by lion:

3. Computer Conversations - Lion loose at NUIG.

```
smarter(lion, lecturer).
chases(lion, lecturer).
chases(lion, nuig_student).
deadend(nuig_student).
caught(Y) ← chases(X,Y) ∧ smarter(X,Y).
caught(Y) ← chases(X,Y) ∧ deadend(Y).

caught(X)?
yes(X) ← caught(X).
```

Caught by lion:

3. Computer Conversations - Lion loose at NUIG.

```
smarter(lion, lecturer).
chases(lion, lecturer).
chases(lion, nuig_student).
deadend(nuig_student).
caught(Y) ← chases(X,Y) ∧ smarter(X,Y).
caught(Y) ← chases(X,Y) ∧ deadend(Y).

caught(X)?
yes(X) ← caught(X).
    caught(Y′) ← chases(X′, Y′) ∧ smarter(X′, Y′).
    {Y′/X, X′/_}
yes(X) ← chases(X′, X) ∧ smarter(X′, X).
```

Caught by lion:

3. Computer Conversations - Lion loose at NUIG.

```
smarter(lion, lecturer).
chases(lion, lecturer).
chases(lion, nuig_student).
deadend(nuig_student).
caught(Y) ← chases(X,Y) ∧ smarter(X,Y).
caught(Y) ← chases(X,Y) ∧ deadend(Y).

caught(X)?
yes(X) ← caught(X).
    caught(Y′) ← chases(X′, Y′) ∧ smarter(X′, Y′).
    {Y′/X, X′/_}
yes(X) ← chases(X′, X) ∧ smarter(X′, X).
    chases(lion, lecturer).
    {X′/lion, X/lecturer}
yes(lecturer) ← smarter(lion, lecturer).
```

Caught by lion:

3. Computer Conversations - Lion loose at NUIG.

```
smarter(lion, lecturer).
chases(lion, lecturer).
chases(lion, nuig_student).
deadend(nuig_student).
caught(Y) ← chases(X,Y) ∧ smarter(X,Y).
caught(Y) ← chases(X,Y) ∧ deadend(Y).

caught(X)?
yes(X) ← caught(X).
    caught(Y′) ← chases(X′, Y′) ∧ smarter(X′, Y′).
    {Y′/X, X′/_}
yes(X) ← chases(X′, X) ∧ smarter(X′, X).
    chases(lion, lecturer).
    {X′/lion, X/lecturer}
yes(lecturer) ← smarter(lion, lecturer).
yes(lecturer) ←.
```

Caught by lion:

3. Computer Conversations - Lion loose at NUIG.

```
smarter(lion, lecturer).
chases(lion, lecturer).
chases(lion, nuig_student).
deadend(nuig_student).
caught(Y) ← chases(X,Y) ∧ smarter(X,Y).
caught(Y) ← chases(X,Y) ∧ deadend(Y).

caught(X)?
yes(X) ← caught(X).
    caught(Y′) ← chases(X′, Y′) ∧ smarter(X′, Y′).
    {Y′/X, X′/_}
yes(X) ← chases(X′, X) ∧ smarter(X′, X).
    chases(lion, lecturer).
    {X′/lion, X/lecturer}
yes(lecturer) ← smarter(lion, lecturer).
yes(lecturer) ←.
```

Caught by lion:
lecturer

3. Computer Conversations - Lion loose at NUIG.

```
smarter(lion, lecturer).
chases(lion, lecturer).
chases(lion, nuig_student).
deadend(nuig_student).
caught(Y) ← chases(X,Y) ∧ smarter(X,Y).
caught(Y) ← chases(X,Y) ∧ deadend(Y).

caught(X)?
yes(X) ← caught(X).
    caught(Y′) ← chases(X′, Y′) ∧ smarter(X′, Y′).
    {Y′/X, X′/_}
yes(X) ← chases(X′, X) ∧ smarter(X′, X).
```

Caught by lion:
lecturer

3. Computer Conversations - Lion loose at NUIG.

```
smarter(lion, lecturer).
chases(lion, lecturer).
chases(lion, nuig_student).
deadend(nuig_student).
caught(Y) ← chases(X,Y) ∧ smarter(X,Y).
caught(Y) ← chases(X,Y) ∧ deadend(Y).

caught(X)?
yes(X) ← caught(X).
    caught(Y′) ← chases(X′, Y′) ∧ smarter(X′, Y′).
    {Y′/X, X′/_}
yes(X) ← chases(X′, X) ∧ smarter(X′, X).
    chases(lion, nuig_student).
    {X′/lion, X/nuig_student}
yes(nuig_student) ← smarter(lion, nuig_student).
```

Caught by lion:
lecturer

3. Computer Conversations - Lion loose at NUIG.

```
smarter(lion, lecturer).
chases(lion, lecturer).
chases(lion, nuig_student).
deadend(nuig_student).
caught(Y) ← chases(X,Y) ∧ smarter(X,Y).
caught(Y) ← chases(X,Y) ∧ deadend(Y).
```

Caught by lion:
lecturer

```
caught(X)?
yes(X) ← caught(X).
    caught(Y′) ← chases(X′, Y′) ∧ smarter(X′, Y′).
    {Y′/X, X′/_}
yes(X) ← chases(X′, X) ∧ smarter(X′, X).
    chases(lion, nuig_student).
    {X′/lion, X/nuig_student}
yes(nuig_student) ← smarter(lion, nuig_student).
```

FAIL!

3. Computer Conversations - Lion loose at NUIG.

```
smarter(lion, lecturer).
chases(lion, lecturer).
chases(lion, nuig_student).
deadend(nuig_student).
caught(Y) ← chases(X,Y) ∧ smarter(X,Y).
caught(Y) ← chases(X,Y) ∧ deadend(Y).

caught(X)?
yes(X) ← caught(X).
    caught(Y′) ← chases(X′, Y′) ∧ smarter(X′, Y′).
    {Y′/X, X′/_}
yes(X) ← chases(X′, X) ∧ smarter(X′, X).
    chases(lion, nuig_student).
    {X′/lion, X/nuig_student}
yes(nuig_student) ← smarter(lion, nuig_student).
```

Caught by lion:
lecturer

3. Computer Conversations - Lion loose at NUIG.

```
smarter(lion, lecturer).
chases(lion, lecturer).
chases(lion, nuig_student).
deadend(nuig_student).
caught(Y) ← chases(X,Y) ∧ smarter(X,Y).
caught(Y) ← chases(X,Y) ∧ deadend(Y).

caught(X)?
yes(X) ← caught(X).
    caught(Y′) ← chases(X′, Y′) ∧ smarter(X′, Y′).
    {Y′/X, X′/_}
yes(X) ← chases(X′, X) ∧ smarter(X′, X).
```

Caught by lion:
lecturer

3. Computer Conversations - Lion loose at NUIG.

```
smarter(lion, lecturer).
chases(lion, lecturer).
chases(lion, nuig_student).
deadend(nuig_student).
caught(Y) ← chases(X,Y) ∧ smarter(X,Y).
caught(Y) ← chases(X,Y) ∧ deadend(Y).

caught(X)?
yes(X) ← caught(X).
```

Caught by lion:
lecturer

3. Computer Conversations - Lion loose at NUIG.

```
smarter(lion, lecturer).
chases(lion, lecturer).
chases(lion, nuig_student).
deadend(nuig_student).
caught(Y) ← chases(X,Y) ∧ smarter(X,Y).
caught(Y) ← chases(X,Y) ∧ deadend(Y).

caught(X)?
yes(X) ← caught(X).
    caught(Y′) ← chases(X′, Y′) ∧ deadend(Y′).
    {Y′/X, X′/_}
yes(X) ← chases(X′, X) ∧ deadend(X).
```

Caught by lion:
lecturer

3. Computer Conversations - Lion loose at NUIG.

```
smarter(lion, lecturer).
chases(lion, lecturer).
chases(lion, nuig_student).
deadend(nuig_student).
caught(Y) ← chases(X,Y) ∧ smarter(X,Y).
caught(Y) ← chases(X,Y) ∧ deadend(Y).

caught(X)?
yes(X) ← caught(X).
    caught(Y′) ← chases(X′, Y′) ∧ deadend(Y′).
    {Y′/X, X′/_}
yes(X) ← chases(X′, X) ∧ deadend(X).
    chases(lion, lecturer).
    {X′/lion, X/lecturer}
yes(lecturer) ← deadend(lecturer).
```

Caught by lion:
lecturer

3. Computer Conversations - Lion loose at NUIG.

```
smarter(lion, lecturer).
chases(lion, lecturer).
chases(lion, nuig_student).
deadend(nuig_student).
caught(Y) ← chases(X,Y) ∧ smarter(X,Y).
caught(Y) ← chases(X,Y) ∧ deadend(Y).

caught(X)?
yes(X) ← caught(X).
    caught(Y′) ← chases(X′, Y′) ∧ deadend(Y′).
    {Y′/X, X′/_}
yes(X) ← chases(X′, X) ∧ deadend(X).
    chases(lion, lecturer).
    {X′/lion, X/lecturer}
yes(lecturer) ← deadend(lecturer).
```

FAIL!

Caught by lion:
lecturer

3. Computer Conversations - Lion loose at NUIG.

```
smarter(lion, lecturer).
chases(lion, lecturer).
chases(lion, nuig_student).
deadend(nuig_student).
caught(Y) ← chases(X,Y) ∧ smarter(X,Y).
caught(Y) ← chases(X,Y) ∧ deadend(Y).

caught(X)?
yes(X) ← caught(X).
    caught(Y′) ← chases(X′, Y′) ∧ deadend(Y′).
    {Y′/X, X′/_}
yes(X) ← chases(X′, X) ∧ deadend(X).
    chases(lion, lecturer).
    {X′/lion, X/lecturer}
yes(lecturer) ← deadend(lecturer).
```

Caught by lion:
lecturer

3. Computer Conversations - Lion loose at NUIG.

```
smarter(lion, lecturer).
chases(lion, lecturer).
chases(lion, nuig_student).
deadend(nuig_student).
caught(Y) ← chases(X,Y) ∧ smarter(X,Y).
caught(Y) ← chases(X,Y) ∧ deadend(Y).

caught(X)?
yes(X) ← caught(X).
    caught(Y′) ← chases(X′, Y′) ∧ deadend(Y′).
    {Y′/X, X′/_}
yes(X) ← chases(X′, X) ∧ deadend(X).
```

Caught by lion:
lecturer

3. Computer Conversations - Lion loose at NUIG.

```
smarter(lion, lecturer).
chases(lion, lecturer).
chases(lion, nuig_student).
deadend(nuig_student).
caught(Y) ← chases(X,Y) ∧ smarter(X,Y).
caught(Y) ← chases(X,Y) ∧ deadend(Y).

caught(X)?
yes(X) ← caught(X).
    caught(Y′) ← chases(X′, Y′) ∧ deadend(Y′).
    {Y′/X, X′/_}
yes(X) ← chases(X′, X) ∧ deadend(X).
    chases(lion, nuig_student).
    {X′/lion, X/nuig_student}
yes(nuig_student) ← deadend(nuig_student).
```

Caught by lion:
lecturer

3. Computer Conversations - Lion loose at NUIG.

```
smarter(lion, lecturer).
chases(lion, lecturer).
chases(lion, nuig_student).
deadend(nuig_student).
caught(Y) ← chases(X,Y) ∧ smarter(X,Y).
caught(Y) ← chases(X,Y) ∧ deadend(Y).

caught(X)?
yes(X) ← caught(X).
    caught(Y′) ← chases(X′, Y′) ∧ deadend(Y′).
    {Y′/X, X′/_}
yes(X) ← chases(X′, X) ∧ deadend(X).
    chases(lion, nuig_student).
    {X′/lion, X/nuig_student}
yes(nuig_student) ← deadend(nuig_student).
yes(nuig_student) ←
```

Caught by lion:
lecturer

3. Computer Conversations - Lion loose at NUIG.

```
smarter(lion, lecturer).
chases(lion, lecturer).
chases(lion, nuig_student).
deadend(nuig_student).
caught(Y) ← chases(X,Y) ∧ smarter(X,Y).
caught(Y) ← chases(X,Y) ∧ deadend(Y).

caught(X)?
yes(X) ← caught(X).
    caught(Y′) ← chases(X′, Y′) ∧ deadend(Y′).
    {Y′/X, X′/_}
yes(X) ← chases(X′, X) ∧ deadend(X).
    chases(lion, nuig_student).
    {X′/lion, X/nuig_student}
yes(nuig_student) ← deadend(nuig_student).
yes(nuig_student) ←
```

Caught by lion:
lecturer
nuig_student

4. Computer Translation

- Translate given facts (i.e. translation for basic words)

4. Computer Translation

- Translate given facts (i.e. translation for basic words)
- and rules (grammar) (e.g. number (singular or plural), person (1st, 2nd or 3rd), etc.)

4. Computer Translation

- Translate given facts (i.e. translation for basic words)
- and rules (grammar) (e.g. number (singular or plural), person (1st, 2nd or 3rd), etc.)
- and of course exceptions

4. Computer Translation

- Translate given facts (i.e. translation for basic words)
- and rules (grammar) (e.g. number (singular or plural), person (1st, 2nd or 3rd), etc.)
- and of course exceptions
- Difference lists are used in Prolog to hold grammatical structures

4. Computer Translation

- Translate given facts (i.e. translation for basic words)
- and rules (grammar) (e.g. number (singular or plural), person (1st, 2nd or 3rd), etc.)
- and of course exceptions
- Difference lists are used in Prolog to hold grammatical structures
- Agreements made (e.g. between noun and verb)

4. Computer Translation

- Translate given facts (i.e. translation for basic words)
- and rules (grammar) (e.g. number (singular or plural), person (1st, 2nd or 3rd), etc.)
- and of course exceptions
- Difference lists are used in Prolog to hold grammatical structures
- Agreements made (e.g. between noun and verb)

Need to breakdown a sentence into its components

4. Computer Translation

- Translate given facts (i.e. translation for basic words)
- and rules (grammar) (e.g. number (singular or plural), person (1st, 2nd or 3rd), etc.)
- and of course exceptions
- Difference lists are used in Prolog to hold grammatical structures
- Agreements made (e.g. between noun and verb)

Need to breakdown a sentence into its components
e.g. Simple Sentence = noun phrase + verb phrase

4. Computer Translation

- Translate given facts (i.e. translation for basic words)
- and rules (grammar) (e.g. number (singular or plural), person (1st, 2nd or 3rd), etc.)
- and of course exceptions
- Difference lists are used in Prolog to hold grammatical structures
- Agreements made (e.g. between noun and verb)

Need to breakdown a sentence into its components
e.g. Simple Sentence = noun phrase + verb phrase

But things can get a lot more complicated than this!

4. Computer Translation

- Translate given facts (i.e. translation for basic words)
- and rules (grammar) (e.g. number (singular or plural), person (1st, 2nd or 3rd), etc.)
- and of course exceptions
- Difference lists are used in Prolog to hold grammatical structures
- Agreements made (e.g. between noun and verb)

Need to breakdown a sentence into its components
e.g. Simple Sentence = noun phrase + verb phrase

But things can get a lot more complicated than this!

We'll take a look at an example in Prolog.

5. Ruining the fun of games

One of the few successes in AI:

- Famous victory for Deep Blue over Gary Kasparov at Chess.

5. Ruining the fun of games

One of the few successes in AI:

- Famous victory for Deep Blue over Gary Kasparov at Chess.
- Used learning algorithms and clever heuristics.

5. Ruining the fun of games

One of the few successes in AI:

- Famous victory for Deep Blue over Gary Kasparov at Chess.
- Used learning algorithms and clever heuristics.

Game of Draughts Solved:

5. Ruining the fun of games

One of the few successes in AI:

- Famous victory for Deep Blue over Gary Kasparov at Chess.
- Used learning algorithms and clever heuristics.

Game of Draughts Solved:

- will always end in a stalemate if neither player makes a mistake

5. Ruining the fun of games

One of the few successes in AI:

- Famous victory for Deep Blue over Gary Kasparov at Chess.
- Used learning algorithms and clever heuristics.

Game of Draughts Solved:

- will always end in a stalemate if neither player makes a mistake
- most complex game ever solved.

5. Ruining the fun of games

One of the few successes in AI:

- Famous victory for Deep Blue over Gary Kasparov at Chess.
- Used learning algorithms and clever heuristics.

Game of Draughts Solved:

- will always end in a stalemate if neither player makes a mistake
- most complex game ever solved.

2-Player Strategy Games

5. Ruining the fun of games

One of the few successes in AI:

- Famous victory for Deep Blue over Gary Kasparov at Chess.
- Used learning algorithms and clever heuristics.

Game of Draughts Solved:

- will always end in a stalemate if neither player makes a mistake
- most complex game ever solved.

2-Player Strategy Games

- Draw Minimax trees

5. Ruining the fun of games

One of the few successes in AI:

- Famous victory for Deep Blue over Gary Kasparov at Chess.
- Used learning algorithms and clever heuristics.

Game of Draughts Solved:

- will always end in a stalemate if neither player makes a mistake
- most complex game ever solved.

2-Player Strategy Games

- Draw Minimax trees
- Use heuristics function to find best strategy

5. Ruining the fun of games

One of the few successes in AI:

- Famous victory for Deep Blue over Gary Kasparov at Chess.
- Used learning algorithms and clever heuristics.

Game of Draughts Solved:

- will always end in a stalemate if neither player makes a mistake
- most complex game ever solved.

2-Player Strategy Games

- Draw Minimax trees
- Use heuristics function to find best strategy
- Follow best path in tree

5. Ruining the fun of games

One of the few successes in AI:

- Famous victory for Deep Blue over Gary Kasparov at Chess.
- Used learning algorithms and clever heuristics.

Game of Draughts Solved:

- will always end in a stalemate if neither player makes a mistake
- most complex game ever solved.

2-Player Strategy Games

- Draw Minimax trees
- Use heuristics function to find best strategy
- Follow best path in tree

Search trees are interesting in that they appear all over AI.

5. Ruining the fun of games

Constraint Satisfaction Problems

5. Ruining the fun of games

Constraint Satisfaction Problems

A triple $\langle X, D, C \rangle$:

X *set of variables*

$\mathcal{D}$ *domain of values*

C *Constraints of form* $\langle t, R \rangle$ *where*

$t =$ *tuples of variables*

$R =$ *tuples of relations.*

5. Ruining the fun of games

Constraint Satisfaction Problems

A triple $\langle X, D, C \rangle$:

X	*set of variables*
$\mathcal{D}$	*domain of values*
C	*Constraints of form* $\langle t, R \rangle$ *where*
	$t =$ *tuples of variables*
	$R =$ *tuples of relations.*

Examples

5. Ruining the fun of games

Constraint Satisfaction Problems

A triple $\langle X, D, C \rangle$:

X *set of variables*

$\mathcal{D}$ *domain of values*

C *Constraints of form* $\langle t, R \rangle$ *where*

$t =$ *tuples of variables*

$R =$ *tuples of relations.*

Examples

- Sudoku

5. Ruining the fun of games

Constraint Satisfaction Problems

A triple $\langle X, D, C \rangle$:

X *set of variables*

$\mathcal{D}$ *domain of values*

C *Constraints of form* $\langle t, R \rangle$ *where*

$t =$ *tuples of variables*

$R =$ *tuples of relations.*

Examples

- Sudoku
- N-Queens

5. Ruining the fun of games

Constraint Satisfaction Problems

A triple $\langle X, D, C \rangle$:

X	*set of variables*
$\mathcal{D}$	*domain of values*
C	*Constraints of form* $\langle t, R \rangle$ *where*
	$t =$ *tuples of variables*
	$R =$ *tuples of relations.*

Examples

- Sudoku
- N-Queens
- Minesweeper

5. Ruining the fun of games

Constraint Satisfaction Problems

A triple $\langle X, D, C \rangle$:

X *set of variables*

$\mathcal{D}$ *domain of values*

C *Constraints of form* $\langle t, R \rangle$ *where*

$t =$ *tuples of variables*

$R =$ *tuples of relations.*

Examples

- Sudoku
- N-Queens
- Minesweeper
- Timetabling

5. Ruining the fun of games - CSPs

Techniques

5. Ruining the fun of games - CSPs

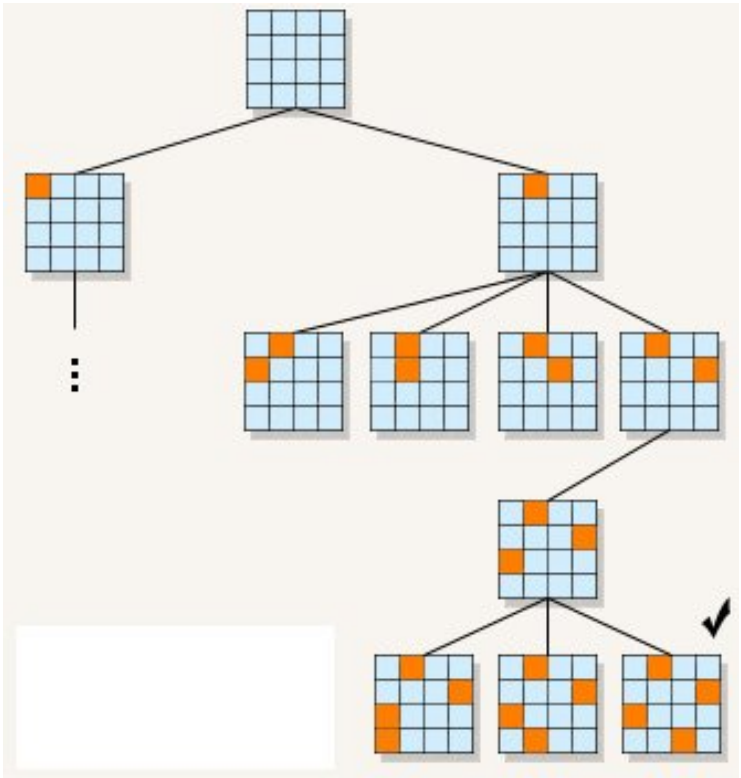
Techniques

- Try all possible positions and backtrack when goal not reached

5. Ruining the fun of games - CSPs

Techniques

- Try all possible positions and backtrack when goal not reached



4-Queens problem with backtracking.

5. Ruining the fun of games - CSPs

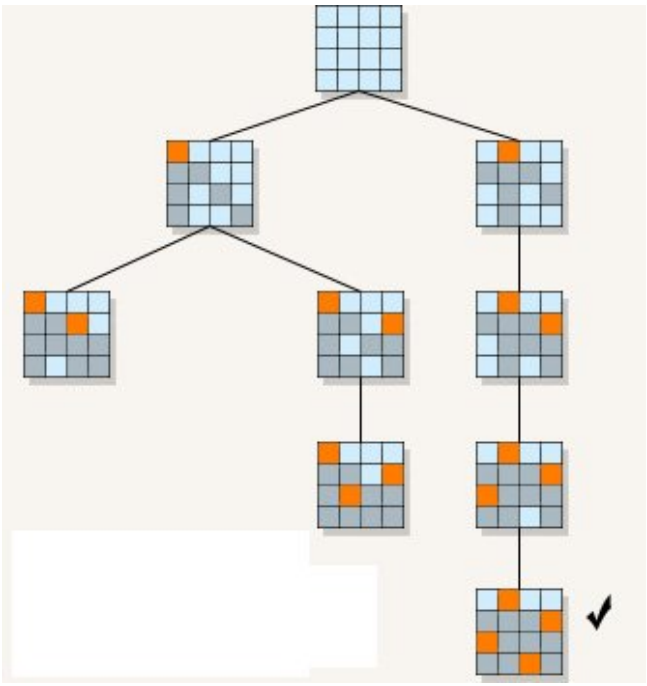
Techniques

- Forward checking - limit positions to check.

5. Ruining the fun of games - CSPs

Techniques

- Forward checking - limit positions to check.



4-Queens problem forward checking with backtracking.

5. Ruining the fun of games - CSPs

Techniques

- There is a better solution called "Arc Consistency" which we'll look at in the course along with many other things.

5. Ruining the fun of games - CSPs

Techniques

- There is a better solution called "Arc Consistency" which we'll look at in the course along with many other things.

The End - Any Questions?

5. Ruining the fun of games - CSPs

Techniques

- There is a better solution called "Arc Consistency" which we'll look at in the course along with many other things.

The End - Any Questions?

Ask the robot!