Index

Introduction

Logic Programming

MDP

MDP+ Logic + Programming
Index

Introduction

Logic Programming

MDP

MDP+ Logic + Programming
Why do we care about planning?
Sequential Decision Making
Sequential Decision Making
Sequential Decision Making
Sequential Decision Making

Desired Property?
Sequential Decision Making

Desired Property?

Feasible: Logic (Focuses on Feasibility)
Sequential Decision Making

Desired Property?

Feasible: Logic (Focuses on Feasibility)

Optimal: MDPs (Scaling is Hard)
Logic

Goal Directed Search

Start ∨ Goal

State: Set of propositions (e.g. ¬Quite, Garbage, ¬Dinner)

Feasible Plan: Sequence of actions from start to goal
Actions

- **Precondition**: Clean Hands
- **Action**: Cook
- **Effect**: Dinner Ready

**STRIPS, Graph Plan** (16.410/16.413)
Logic Programming

Graph Plan or Forward Search

Not easily scalable

GOLOG: ALGOL in \textsc{Logic} \cite{Levesque97}

Restrict action space with programs

Scales easier

Results are dependent on high-level programs
Situation Calculus (Temporal Logic)

Situation: S0, do(A,S)
Situation Calculus (Temporal Logic)

Situation: S0, do(A,S)

Example: do(putdown(A), do(walk(L), do(pickup(A), S0)))
Situation Calculus (Temporal Logic)

- Situation: S0, do(A,S)

- Example: do(putdown(A), do(walk(L), do(pickup(A), S0)))

- Relational Fluent: relations with values depending on situations
Situation Calculus (Temporal Logic)

- Situation: S0, do(A,S)

  - Example: do(putdown(A), do(walk(L), do(pickup(A), S0)))

- Relational Fluent: relations with values depending on situations

  - Example: is_carrying(robot, p, s)
Situation Calculus (Temporal Logic)

Situation: S0, do(A,S)

Example: do(putdown(A), do(walk(L), do(pickup(A), S0)))

Relational Fluent: relations with values depending on situations

Example: is_carrying(robot, p, s)

Functional Fluent: situation dependent functions
Situation Calculus (Temporal Logic)

- Situation: S0, do(A,S)

- Example: do(putdown(A), do(walk(L), do(pickup(A), S0)))

- Relational Fluent: relations with values depending on situations

- Example: is_carrying(robot, p, s)

- Functional Fluent: situation dependent functions

- Example: loc(robot, s)
Situation Calculus (Temporal Logic)

- Situation: $S_0$, do(A,S)

- Example: $\text{do(putdown(A), do(walk(L), do(pickup(A), S_0)))}$

- Relational Fluent: relations with values depending on situations

- Example: $\text{is_carrying(robot, p, s)}$

- Functional Fluent: situation dependent functions

- Example: $\text{loc(robot, s)}$

More expressive than LTL
GOLOG: Syntax

\( \alpha \quad \text{primitive action} \)

[Scott Sanner - ICAPS08 Tutorial]
GOLOG: Syntax

\[ \alpha \]
\[ \phi? \]

primitive action
condition test

[Scott Sanner - ICAPS08 Tutorial]
| $\alpha$ | primitive action |
| $\phi?$ | condition test |
| $(\delta_1, \delta_2)$ | sequence |
## GOLOG: Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>α</td>
<td>primitive action</td>
</tr>
<tr>
<td>φ?</td>
<td>condition test</td>
</tr>
<tr>
<td>(δ₁, δ₂)</td>
<td>sequence</td>
</tr>
<tr>
<td>if φ then δ₂ endif</td>
<td>conditional</td>
</tr>
</tbody>
</table>
### GOLOG: Syntax

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>primitive action</td>
</tr>
<tr>
<td>$\phi?$</td>
<td>condition test</td>
</tr>
<tr>
<td>$(\delta_1, \delta_2)$</td>
<td>sequence</td>
</tr>
<tr>
<td>if $\phi$ then $\delta_2$ endIf</td>
<td>conditional</td>
</tr>
<tr>
<td>while $\phi$ then $\delta$ endWhile</td>
<td>loop</td>
</tr>
</tbody>
</table>

[Scott Sanner - ICAPS08 Tutorial]
<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>primitive action</td>
</tr>
<tr>
<td>$\phi?$</td>
<td>condition test</td>
</tr>
<tr>
<td>$(\delta_1, \delta_2)$</td>
<td>sequence</td>
</tr>
<tr>
<td><strong>if $\phi$ then $\delta_2$ endIf</strong></td>
<td>conditional</td>
</tr>
<tr>
<td><strong>while $\phi$ then $\delta$ endWhile</strong></td>
<td>loop</td>
</tr>
<tr>
<td>$(\delta_1</td>
<td>\delta_2)$</td>
</tr>
</tbody>
</table>

[Scott Sanner - ICAPS08 Tutorial]
GOLOG: Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \alpha )</td>
<td>primitive action</td>
</tr>
<tr>
<td>( \phi? )</td>
<td>condition test</td>
</tr>
<tr>
<td>((\delta_1, \delta_2))</td>
<td>sequence</td>
</tr>
<tr>
<td>if ( \phi ) then ( \delta_2 ) endif</td>
<td>conditional</td>
</tr>
<tr>
<td>while ( \phi ) then ( \delta ) endwhile</td>
<td>loop</td>
</tr>
<tr>
<td>((\delta_1</td>
<td>\delta_2))</td>
</tr>
<tr>
<td>(\pi \vec{x} [\delta])</td>
<td>nondeterministic choice of arguments</td>
</tr>
</tbody>
</table>

[Scott Sanner - ICAPS08 Tutorial]
<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\alpha)</td>
<td>primitive action</td>
</tr>
<tr>
<td>(\phi?)</td>
<td>condition test</td>
</tr>
<tr>
<td>((\delta_1, \delta_2))</td>
<td>sequence</td>
</tr>
<tr>
<td>if (\phi) then (\delta_2) endif</td>
<td>conditional</td>
</tr>
<tr>
<td>while (\phi) then (\delta) endwhile</td>
<td>loop</td>
</tr>
<tr>
<td>((\delta_1</td>
<td>\delta_2))</td>
</tr>
<tr>
<td>(\pi \vec{x} [\delta])</td>
<td>nondeterministic choice of arguments</td>
</tr>
<tr>
<td>(\delta^*)</td>
<td>nondeterministic iteration</td>
</tr>
<tr>
<td>Syntax</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>primitive action</td>
</tr>
<tr>
<td>$\phi?$</td>
<td>condition test</td>
</tr>
<tr>
<td>$(\delta_1, \delta_2)$</td>
<td>sequence</td>
</tr>
<tr>
<td>if $\phi$ then $\delta_2$ endIf</td>
<td>conditional</td>
</tr>
<tr>
<td>while $\phi$ then $\delta$ endWhile</td>
<td>loop</td>
</tr>
<tr>
<td>$(\delta_1</td>
<td>\delta_2)$</td>
</tr>
<tr>
<td>$\pi \vec{x} [\delta]$</td>
<td>nondeterministic choice of arguments</td>
</tr>
<tr>
<td>$\delta^*$</td>
<td>nondeterministic iteration</td>
</tr>
<tr>
<td>proc $\beta(\vec{x})$ $\delta$ endProc</td>
<td>procedure call definition</td>
</tr>
</tbody>
</table>
## GOLOG: Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>primitive action</td>
</tr>
<tr>
<td>$\phi?$</td>
<td>condition test</td>
</tr>
<tr>
<td>$(\delta_1, \delta_2)$</td>
<td>sequence</td>
</tr>
<tr>
<td>if $\phi$ then $\delta_2$ endIf</td>
<td>conditional</td>
</tr>
<tr>
<td>while $\phi$ then $\delta$ endWhile</td>
<td>loop</td>
</tr>
<tr>
<td>$(\delta_1</td>
<td>\delta_2)$</td>
</tr>
<tr>
<td>$\pi \vec{x} [\delta]$</td>
<td>nondeterministic choice of arguments</td>
</tr>
<tr>
<td>$\delta^*$</td>
<td>nondeterministic iteration</td>
</tr>
<tr>
<td>proc $\beta(\vec{x})$ $\delta$ endProc</td>
<td>procedure call definition</td>
</tr>
<tr>
<td>$\beta(\vec{t})$</td>
<td>procedure call</td>
</tr>
</tbody>
</table>

[Scott Sanner - ICAPS08 Tutorial]
Aren’t we hard-coding the whole solution?

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>primitive action</td>
</tr>
<tr>
<td>$\phi?$</td>
<td>condition test</td>
</tr>
<tr>
<td>$(\delta_1, \delta_2)$</td>
<td>sequence</td>
</tr>
<tr>
<td><strong>if</strong> $\phi$ <strong>then</strong> (\delta_2) <strong>endIf</strong></td>
<td>conditional</td>
</tr>
<tr>
<td><strong>while</strong> $\phi$ <strong>then</strong> (\delta) <strong>endWhile</strong></td>
<td>loop</td>
</tr>
<tr>
<td>$(\delta_1</td>
<td>\delta_2)$</td>
</tr>
<tr>
<td>$\pi \vec{x}[\delta]$</td>
<td>nondeterministic choice of arguments</td>
</tr>
<tr>
<td>$\delta^*$</td>
<td>nondeterministic iteration</td>
</tr>
<tr>
<td><strong>proc</strong> $\beta(\vec{x})$ $\delta$ <strong>endProc</strong></td>
<td>procedure call definition</td>
</tr>
<tr>
<td>$\beta(\vec{t})$</td>
<td>procedure call</td>
</tr>
</tbody>
</table>
Blocks World

Actions: pickup(b), putOnTable(b), putOn(b1,b2)

Fluents: onTable(b, s), on(b1, b2, s)

GOLOG Program:

while (\exists b) \neg onTable(b) then
   (pickup(b), putOnTable(b))
endWhile
Blocks World

Actions: pickup(b), putOnTable(b), putOn(b1,b2)

Fluents: onTable(b, s), on(b1, b2, s)

GOLOG Program:

while (∃b) ¬onTable(b) then
  (pickup(b), putOnTable(b))
endWhile

What does it do?
GOLOG Execution
GOLOG Execution

Given Start S0, Goal S’, and program \( \delta \)
GOLOG Execution

- Given Start S₀, Goal S', and program δ
- Call Do(δ,S₀,S')
GOLOG Execution

- Given Start $S_0$, Goal $S'$, and program $\delta$
- Call $\text{Do}(\delta, S_0, S')$
- First-order logic proof system
GOLOG Execution

- Given Start S₀, Goal S', and program δ
- Call Do(δ,S₀,S')
- First-order logic proof system
- Prolog
GOLOG Execution

- Given Start $S_0$, Goal $S'$, and program $\delta$
- Call $\text{Do}(\delta, S_0, S')$
- First-order logic proof system
- Prolog
Given Start $S_0$, Goal $S'$, and program $\delta$

Call $\text{Do}(\delta, S_0, S')$

First-order logic proof system

Prolog
Given Start $S_0$, Goal $S'$, and program $\delta$

Call $\text{Do}(\delta, S_0, S')$

First-order logic proof system

Prolog
Given Start $S_0$, Goal $S'$, and program $\delta$

Call $\text{Do}(\delta, S_0, S')$

First-order logic proof system

Prolog
Given Start $S_0$, Goal $S'$, and program $\delta$

Call $\text{Do}(\delta, S_0, S')$

First-order logic proof system

Prolog

Non-deterministic
\[ \langle S, A, P_{ss'}, R_{ss'}, \gamma \rangle \]

B.S., Working, +60, B.S. Studying, -50, M.S., ...
**Policy** $\pi(s)$: How to act in each state

**Value Function** $V(s)$: How good is it to be in each state?

$$V^\pi(s) = E \left[ \sum_{t=1}^{\infty} \gamma^{t-1} r_t \mid s_0 = s, \pi \right].$$
Goal

Find the optimal policy \((\pi^*)\), which maximizes the value function for all states

\[
V^*(s) = \max_a E \left[ r + \gamma V^*(s', s) \right]
\]

\[
\pi^*(s) = \arg\max_a \ldots
\]
Example (Value Iteration)

\[ V(s) = \max_a E \left[ r + \gamma V(s') \right] \]
Example (Value Iteration)

\[ V(s) = \max_a E \left[ r + \gamma V(s') \right] \]
Example (Value Iteration)

\[ V(s) = \max_a E \left[ r + \gamma V(s') \right] \]
Example (Value Iteration)

\[ V(s) = \max_a \mathbb{E} \left[ r + \gamma V(s') \right] \]

Reward:

\[ \begin{array}{c|c}
0 & -1 \\
-1 & -2 \\
-2 & -2 \\
-2 & -2 \\
\end{array} \]

Actions:
Example (Value Iteration)

\[ V(s) = \max_a E \left[ r + \gamma V(s') \right] \]
Index

Introduction

Logic Programming

MDP

MDP+ Logic + Programming
DT-GOLOG: Decision Theoretic GOLOG

[Boutilier, et. al 00]
DT-GOLOG: Decision Theoretic GOLOG

[Boutilier, et. al 00]
DT-GOLOG: Decision Theoretic GOLOG

GOLOG

No Optimization

Scaling is challenging

MDP

[Boutilier, et. al 00]
DT-GOLOG: Decision Theoretic GOLOG

No Optimization

GOLOG

Scaling is challenging

MDP

All non-deterministic actions are optimized using MDP concepts.

[Boutilier, et. al 00]
DT-GOLOG: Decision Theoretic GOLOG

- All non-deterministic actions are optimized using MDP concepts.

- Uncertainty, Reward

[Boutilier, et. al 00]
Decision Theoretic GOLOG

Programming (GOLOG)

Planning (MDP)
Decision Theoretic GOLOG

- Known Solution
- High-level Structure

- Programming (GOLOG)
  - Planning (MDP)
Decision Theoretic GOLOG

- Known Solution
- High-level Structure

- Not enough grasping
- Low-level Structure

Programming (GOLOG)
Planning (MDP)
Example

Build a tower with least effort
Example

- Build a tower with least effort
  - Pick a block as base
  - Stack all other blocks on top of it
Example

- Build a tower with least effort
  - Pick a block as base
  - Stack all other blocks on top of it
  - Use which block for base?
  - In which order pick up the blocks
Example

Build a tower with least effort

- Pick a block as base

- Stack all other blocks on top of it

Use which block for base?

In which order pick up the blocks
Example

Build a tower with least effort

- Pick a block as base
- Stack all other blocks on top of it

GOLOG

- Use which block for base?
- In which order pick up the blocks

MDP

Guaranteed to be optimal?
DT-GOLOG Execution

- Given Start $S_0$, Goal $S'$, and program $\delta$
- Add rewards
- Formulate the problem as an MDP
DT-GOLOG Execution

- Given Start S0, Goal S’, and program δ
- Add rewards
- Formulate the problem as an MDP
DT-GOLOG Execution

- Given Start S0, Goal S’, and program δ
- Add rewards
- Formulate the problem as an MDP
Given Start $S_0$, Goal $S'$, and program $\delta$

Add rewards

Formulate the problem as an MDP

$$(\exists b) \neg \text{onTable}(b), b \in \{b_1, ..., b_n\}$$

$$\neg \text{onTable}(b_1) \lor \neg \text{onTable}(b_2) \lor ... \lor \neg \text{onTable}(b_n)$$
First Order Dynamic Programming

- Resulting MDP can still be intractable.

- Idea:
  - Logical Structure
  - Abstract Value Function
  - Avoid curse of dimensionality!

[Sanner 07]
Symbolic Dynamic Programming (Deterministic)

Tabular:

\[ V(s) = \max_a \left( r + \gamma V(s') \right) \]
Symbolic Dynamic Programming (Deterministic)

Tabular:

\[ V(s) = \max_a \left( r + \gamma V(s') \right) \]
Symbolic Dynamic Programming (Deterministic)

Tabular:

\[ V(s) = \max_a \left( r + \gamma V(s') \right) \]
Symbolic Dynamic Programming (Deterministic)

Tabular:

\[ V(s) = \max_{\alpha} \left( r + \gamma V(s') \right) \]
Symbolic Dynamic Programming (Deterministic)

Tabular:

\[ V(s) = \max_a \left( r + \gamma V(s') \right) \]
Symbolic Dynamic Programming (Deterministic)

Tabular:

\[ V(s) = \max_a \left( r + \gamma V(s') \right) \]

Symbolic?

- Representation of Reward and Values
- Adding rewards and values
- Max Operator
- Find S
Reward and Value Representation

Case Representation

\[ r \text{Case} = \begin{cases} \exists b, b \neq b_1, \text{on}(b, b_1) & 10 \\ \nexists b, b \neq b_1, \text{on}(b, b_1) & 0 \end{cases} \]
Add Symbolically

\[
\begin{array}{|c|c|}
\hline
A & 10 \\
\hline
\neg A & 20 \\
\hline
\end{array}
\oplus
\begin{array}{|c|c|}
\hline
B & 1 \\
\hline
\neg B & 2 \\
\hline
\end{array}
= \begin{array}{|c|c|}
\hline
A \land B & 11 \\
\hline
A \land \neg B & 12 \\
\hline
\neg A \land B & 21 \\
\hline
\neg A \land \neg B & 22 \\
\hline
\end{array}

[Scott Sanner - ICAPS08 Tutorial]

Similarly defined for \(\ominus\) and \(\otimes\)
Max operator

<table>
<thead>
<tr>
<th>a_1</th>
<th>a_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>φ1</td>
<td>10</td>
</tr>
<tr>
<td>φ2</td>
<td>5</td>
</tr>
<tr>
<td>φ3</td>
<td>3</td>
</tr>
<tr>
<td>φ4</td>
<td>0</td>
</tr>
</tbody>
</table>

max_a =

<table>
<thead>
<tr>
<th>φ1</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>¬φ1∧φ2</td>
<td>5</td>
</tr>
<tr>
<td>¬φ1∧¬φ2∧φ3</td>
<td>3</td>
</tr>
<tr>
<td>¬φ1∧¬φ2∧¬φ3∧φ4</td>
<td>0</td>
</tr>
</tbody>
</table>

[Scott Sanner - ICAPS08 Tutorial]
Find s? Isn’t it obvious?
Find $s$? Isn’t it obvious?

![Diagram showing two states $s$ and $s'$ connected by an arrow labeled $a$.]
Find s? Isn’t it obvious?

Dynamic Programming: Given $V(s’)$ find $V(s)$

In MDPs, we have $s$ explicitly.

In symbolic representation we have it implicitly so we have to build it.
Find $s = \text{Goal Regression}$

Weakest relation that ensures $\Phi_1$ after taking $a$

- $\text{regress}(\Phi_1, a)$
- $\Phi_1 = \text{clear}(b_1)$

Weakest relation that ensures $\Phi_1$ after taking $a$
Find $s = \text{Goal Regression}$

$\text{regress}(\Phi_1, a)$

Weakest relation that ensures $\Phi_1$ after taking $a$

$\Phi_1 = \text{clear}(b1)$

$\text{put}(A,B)$

$\text{On}(A,b1)$

$\text{Clear}(b1) \land B \neq b1$
Symbolic Dynamic Programming (Deterministic)

Tabular:

\[ V(s) = \max_a \left( r + \gamma V(s') \right) \]

Symbolic:
Symbolic Dynamic Programming (Deterministic)

**Tabular:**

\[ V(s) = \max_a \left( r + \gamma V(s') \right) \]

**Symbolic:**

\[ vCase = \max_a \left( rCase \oplus \gamma \times \text{regr}(vCase, a) \right) \]
Classical Example

Goal: Have a box in Paris

\[ rCase = \begin{cases} \exists b, \text{BoxIn}(b,\text{Paris}) & 10 \\ \text{else} & 0 \end{cases} \]
Classical Example

Actions: \texttt{drive(t,c1,c2), load(b,t), unload(b,t), noop}

load and unload have 10\% chance of failure

Fluents: \texttt{BoxIn(b,c), BoxOn(b,t), TruckIn(t,c)}

Assumptions:

- All cities are connected.
- \( \Upsilon = .9 \)
<table>
<thead>
<tr>
<th>s</th>
<th>$V^*(s)$</th>
<th>$\pi^*(s)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\exists b, \text{BoxIn}(b,\text{Paris})$</td>
<td>100</td>
<td>noop</td>
</tr>
<tr>
<td>else, $\exists b,t \text{TruckIn}(t,\text{Paris}) \land \text{BoxOn}(b,t)$</td>
<td>89</td>
<td>unload($b,t$)</td>
</tr>
<tr>
<td>else, $\exists b,c,t \text{BoxOn}(b,t) \land \text{TruckIn}(t,c)$</td>
<td>80</td>
<td>drive($t,c,\text{paris}$)</td>
</tr>
<tr>
<td>else, $\exists b,c,t \text{BoxIn}(b,c) \land \text{TruckIn}(t,c)$</td>
<td>72</td>
<td>load($b,t$)</td>
</tr>
<tr>
<td>else, $\exists b,c1,c2,t \text{BoxIn}(b,c1) \land \text{TruckIn}(t,c2)$</td>
<td>65</td>
<td>drive($t,c2,c1$)</td>
</tr>
<tr>
<td>else</td>
<td>0</td>
<td>noop</td>
</tr>
</tbody>
</table>
Example

<table>
<thead>
<tr>
<th>s</th>
<th>$V^*(s)$</th>
<th>$\pi^*(s)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\exists b,\text{BoxIn}(b,\text{Paris})$</td>
<td>100</td>
<td>noop</td>
</tr>
<tr>
<td>else, $\exists b,t\text{TruckIn}(t,\text{Paris}) \land \text{BoxOn}(b,t)$</td>
<td>89</td>
<td>unload(b,t)</td>
</tr>
<tr>
<td>else, $\exists b,c,t\text{BoxOn}(b,t) \land \text{TruckIn}(t,c)$</td>
<td>80</td>
<td>drive(t,c,paris)</td>
</tr>
<tr>
<td>else, $\exists b,c,t\text{BoxIn}(b,c) \land \text{TruckIn}(t,c)$</td>
<td>72</td>
<td>load(b,t)</td>
</tr>
<tr>
<td>else, $\exists b,c1,c2,t\text{BoxIn}(b,c1) \land \text{TruckIn}(t,c2)$</td>
<td>65</td>
<td>drive(t,c2,c1)</td>
</tr>
<tr>
<td>else</td>
<td>0</td>
<td>noop</td>
</tr>
</tbody>
</table>

What did we gain by going through all of this?
Conclusion
Conclusion

Logic Programming

Planning

Situation Calculus

GOLOG
Conclusion

Logic Programming

Planning

Situation Calculus

GOLOG

MDP

Review

Value Iteration
Conclusion

Logic Programming

Planning

Situation Calculus

GOLOG

MDP

Review

Value Iteration

MDP + Logic + Programming

DT-GOLOG

Symbolic DP
References


Questions ?
Questions ?