Common Sense Reasoning

Exploiting Background Knowledge in Automated Planning

Miguel García Remesal
mgremesal@fi.upm.es
The Frame Problem

• Arises when reasoning in dynamic domains (i.e. reasoning with actions and effects):
  «If move object A, then object A changes the position»
  (but object B remains the same position,
   and object C remains the same position,
   and, ...)

• Frame problem: reasoning about things that do not change when an event occurs.

• Humans usually describe actions by what they change,
  – We assume that everything else (i.e. the frame) remains unchanged.
  – This cannot be assumed using FOL. We can use situation calculus to deal this issue.
Situation Calculus

- Way of describing dynamic domains in FOL.

- We use fluents to denote that a particular predicate holds in a given state:
  - The statement “object \( a \) is at position \( p \) in state \( s_0 \)” can be represented as \( \text{position}(a, p, s_0) \)

- World is conceived as consisting into a sequence of situations, each being a “snapshot” of the state of the world.

- Situations are generated from previous situations by actions.

- We use the fluent \( \text{result}(a, s) \) to denote the state reached after executing the action \( a \).
  - \( \text{result}(a_1, s_0) = s_1 \)
  - \( \text{result}(a_2, s_1) = s_2 \)
  - ... 
  - \( \text{result}(a_n, s_{n-1}) = s_n \)
Solving the Frame Problem (Example)

• Let us suppose a scenario with:
  – A door, which can be open or closed.
  – A light, which can be on or off.

• This can be represented using the following fluents:
  – open(s): the door is open at state s.
  – on(s): the light is on at state s.

• Four actions:
  – open_door
  – close_door
  – switch_on_light
  – switch_off_light
Solving the Frame Problem (Example)

• We need to represent the effects of each operator (aka effect axioms):
  – \( \forall s \neg \text{open}(s) \rightarrow \text{open}的结果(open\_door, s) \)
  – \( \forall s \text{open}(s) \rightarrow \neg \text{open}的结果(close\_door, s) \)
  – \( \forall s \neg \text{on}(s) \rightarrow \text{on}的结果(switch\_on\_light, s) \)
  – \( \forall s \text{on}(s) \rightarrow \neg \text{on}的结果(switch\_off\_light, s) \)

• We also need to specify what does NOT change after applying each operator! (aka frame axioms):
  – \( \forall s \text{open}(s) \rightarrow \text{open}的结果(switch\_on\_light, s) \)
  – \( \forall s \neg \text{open}(s) \rightarrow \neg \text{open}的结果(switch\_off\_light, s) \)
  – \( \forall s \text{on}(s) \rightarrow \text{on}的结果(close\_door, s) \)
  – \( \forall s \neg \text{on}(s) \rightarrow \neg \text{on}的结果(open\_door, s) \)
  – ...

An alternative (and more elegant) solution

• A more elegant representation is obtained by combining *effect* and *frame axioms* into single axioms called *successor state axioms*.

• Successor state axioms list all the ways a given predicate may change its value over time:
  
  - \( \forall a, s \; \text{open}(\text{result}(a, s)) \iff [(\neg \text{open}(s) \land a = \text{open}\_door) \lor (\text{open}(s) \land a \neq \text{close}\_door)] \)
  
  - \( \forall a, s \; \neg \text{on}(\text{result}(a, s)) \iff [(\text{on}(s) \land a = \text{switch}\_off\_light) \lor (\neg \text{on}(s) \land a \neq \text{switch}\_on\_light)] \)
  
  - ...
Planning Systems for solving the Frame Problem

• We would like to be able to make only the changes required by action descriptions and have the frame without further effort.

• This is the natural thing to do, since we do not expect more than a small fraction of the world to change at a given moment.

• Neither FOL nor SC are adequate for this task.

• We can use planning systems instead (e.g. STRIPS).
The Blocks World Revisited

- Infinitely wide table, finite number of blocks
- Ignore where a block is located on the table
- A block can sit on the table or on another block
- There’s a robot gripper that can hold at most one block
- Want to move blocks from one configuration to another
  - e.g.,

```
initial state
  a  b  c
  |   |
  d   e

goal
  a  b
  |   |
  c
```
The Blocks World Revisited

- Constant symbols:
  - The blocks: a, b, c, d, e

- Predicates:
  - on-table(x) - block x is on the table
  - on(x,y) - block x is on block y
  - clear(x) - block x has nothing on it
  - holding(x) - the robot hand is holding block x
  - handempty - the robot hand isn’t holding anything
The Blocks World Revisited

• Representing states:

initial state

\[ s_0 = \{ \text{on-table}(a), \text{on-table}(b), \text{on-table}(e), \text{on}(c, a), \text{clear}(c), \text{clear}(b), \text{clear}(e), \text{holding}(d) \} \]

goal

\[ g = \{ \text{on-table}(c), \text{on}(b, c), \text{on}(a, b), \text{clear}(a) \} \]
The Blocks World Revisited

unstack(x,y)
Precond: on(x,y), clear(x), handempty
Effects: \neg on(x,y), \neg clear(x), \neg handempty, holding(x), clear(y)

stack(x,y)
Precond: holding(x), clear(y)
Effects: \neg holding(x), \neg clear(y), on(x,y), clear(x), handempty

pickup(x)
Precond: on-table(x), clear(x), handempty
Effects: \neg on-table(x), \neg clear(x), \neg handempty, holding(x)

putdown(x)
Precond: holding(x)
Effects: \neg holding(x), on-table(x), clear(x), handempty
Exploiting Background Knowledge for Planning

• Exploit background knowledge for planning more efficiently.

• Use a domain-configurable planning algorithm.
  – Domain independent inference engine.
  – Background knowledge is converted into logical rules to prune the search tree.

• Logical rules for pruning are formalized using Linear Temporal Logic.
Linear Temporal Logic

• Used for representing dynamic domains (since this cannot be made using FOL). We need:
  – A infinite sequence \(<0, 1, 2, ...>\) of time instants.
  – An infinite sequence \(M = <s_0, s_1, s_0, ...>\) of states of the world.

• We will also need:
  – FOL.
  – Propositional symbols \(\text{TRUE}\) and \(\text{FALSE}\).
  – Modal operators.
  – Bounded quantifiers.
  – The \(\text{GOAL}\) predicate.
# Modal Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Symbol</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Next</td>
<td>$\bigcirc f$</td>
<td>$f$ will hold in the next state</td>
<td>$\bigcirc \text{on}(A, B)$</td>
</tr>
<tr>
<td>Eventually</td>
<td>$\Diamond f$</td>
<td>$f$ will hold in a future state</td>
<td>$\Diamond \text{clear}(A)$</td>
</tr>
<tr>
<td>Always</td>
<td>$\Box f$</td>
<td>$f$ holds in the current state and will hold in all future states</td>
<td>$\Box \text{on-table}(A)$</td>
</tr>
<tr>
<td>Until</td>
<td>$f_1 \mathbin{\mathcal{U}} f_2$</td>
<td>$f_2$ either holds either in the current state or it will hold in a future state and $f_1$ will hold until then (including the current state)</td>
<td>$\text{clear}(B) \cup \text{on-table}(A)$</td>
</tr>
</tbody>
</table>
Let $g(x)$ such that $\{x : g(x)\} = \{x_1, x_2, ..., x_n\}$ is finite and easily computable:

\[
\forall [x : g(x)] f(x) = f(x_1) \land f(x_2) \land ... \land f(x_n)
\]

\[
\exists [x : g(x)] f(x) = f(x_1) \lor f(x_2) \lor ... \lor f(x_n)
\]
Bounded Quantifiers (Example)

∀x[on-table(x)]◊holding(x)

◊holding(B) ∧ ◊holding(C)

∃x[clear(x)]○holding(x)

○holding(A) ∨ ○holding(C)
The GOAL Predicate

• **GOAL**(f) indicates that f holds in all goal states

• Examples:
  
  – **GOAL**(on(C, B))
  
  – **GOAL**(hand-empty)
  
  – **GOAL**(on-table(A))
  
  – ...

![Diagram](image)
Control Rules

• Aim: represent background knowledge using LTL formulas
  – Define helper predicates to be used by the control rules if necessary.
  – Define the control rules.

• Control rules will be used to prune the search tree.
Defining Helper Predicates

• **goodtower(x):** predicate that indicates that x is the block at the top of a tower that **MUST NOT** be modified to reach the goal state.

• **badtower(x):** predicate that indicates that x is the block at the top of a tower that **MUST** be modified to reach the goal state.
Defining Helper Predicates

goodtower\( (x) \leftrightarrow \text{clear}(x) \land \neg \text{GOAL}\( \text{holding}(x) \)) \land \text{goodtower-below}\( (x) \)

goodtower-below\( (x) \leftrightarrow (f_1 \land f_2) \lor \exists \[ y : \text{on}(x, y) \] (f_3 \land f_4 \land f_5 \land f_6 \land f_7 \land f_8) \]

\[ f_1 = \text{on-table}(x) \]
\[ f_2 = \neg \exists \[ y : \text{GOAL}\( \text{on}(x, y) \) \] \]
\[ f_3 = \neg \text{GOAL}\( \text{on-table}(x) \) \]
\[ f_4 = \neg \text{GOAL}\( \text{holding}(y) \) \]
\[ f_5 = \neg \text{GOAL}\( \text{clear}(y) \) \]
\[ f_6 = \forall \[ z : \text{GOAL}\( \text{on}(x, z) \) \] z = y \]
\[ f_7 = \forall \[ z : \text{GOAL}\( \text{on}(z, y) \) \] z = x \]
\[ f_8 = \text{goodtower-below}(y) \]

badtower\( (x) \leftrightarrow \text{clear}(x) \land (\text{GOAL}\( \text{holding}(x) \)) \lor \neg \text{goodtower-below}(x) \)
Defining Control Rules

• Control Rule 1:
  – A goodtower must always remain goodtower

\[ CR_1 = \square (\forall x : \text{clear}(x)) \text{goodtower}(x) \rightarrow \Diamond (\text{clear}(x) \lor \exists y : \text{on}(y, x) \text{goodtower}(y)) \]
Defining Control Rules

• Control Rule 2:
  – Similar to the previous rule.
  – The only difference is that we ensure that we will never put a block on a badtower

\[
CR_2 = CR_1 \land \Box (\text{badtower}(x) \rightarrow \Diamond (\neg \exists y : \text{on}(y, x)))
\]
PROCEDURE TLPlan(s, f, g, π)
1. f⁺ ← Progress(f, s)
2. IF f⁺ = FALSE THEN RETURN FAIL
3. IF s satisfies g THEN RETURN π
4. A ← {Actions that can be executed from s}
5. IF A = Ø THEN RETURN FAIL
6. Choose an action a ∈ A
7. DO s⁺ ← γ(s, a)
8. DO π⁺= π.a
9. RETURN TLPlan(s⁺, f⁺, g, π⁺)
Progress Pseudocode

PROCEDURE Progress(f, s)

SWITCH(f)

CASE (f does not contain modal operators):
    IF s |= f THEN f^+ := TRUE ELSE f^+ := FALSE

CASE (f = f_1 \land f_2):
    f^+ := Progress(f_1, s) \land Progress(f_2, s)

CASE (f = \neg f_1):
    f^+ := \neg Progress(f_1, s)

CASE (f = \Diamond f_1):
    f^+ := f_1

CASE (f = f_1 \lor f_2):
    f^+ := Progress(f_2, s) \lor (Progress(f_1, s) \land f)

CASE (f = \Diamond f_1):
    f^+ := Progress(f_1, s) \lor f

CASE (f = \Box f_1):
    f^+ := Progress(f_1, s) \land f

CASE (f = \forall [x:g(x)] f_1):

    f^+ := \bigwedge \{Progress(\theta(f_1), s) : s |= g(c)\}, \text{ where } \theta = \{x \leftarrow c\}

CASE (f = \exists [x:g(x)] f_1):

    f^+ := \bigvee \{Progress(\theta(f_1), s) : s |= g(c)\}, \text{ where } \theta = \{x \leftarrow c\}
**Progress Example #1**

- \( f = \Box \text{on}(A, B) \)

- \( f^+ = \text{Progress}(\Box \text{on}(A, B), s) \)
- \( f^+ = \text{Progress}(\text{on}(A, B), s) \land \Box \text{on}(A, B) \)

- **Two possibilities:**
  - \( f^+ = \text{TRUE} \land \Box \text{on}(A, B) \)
  - \( f^+ = \Box \text{on}(A, B) \)
  - \( f^+ = \text{FALSE} \land \Box \text{on}(A, B) \)
  - \( f^+ = \text{FALSE} \)

- **Conclusion:**
  - \( \Box \) checks that the control rule holds in the current state.
  - If the rule holds, then \( \Box \) propagates the rule to the next state
  - If the rule does not hold in the current state, the node is pruned (dead end)
Progress Example #2

• \( f = \Box(\text{on}(A, B) \rightarrow \Diamond \text{clear}(A)) \)

• \( f^+ = \text{Progress}(\Box(\text{on}(A, B) \rightarrow \Diamond \text{clear}(A)), s) \)

• \( f^+ = \text{Progress}(\text{on}(A, B) \rightarrow \Diamond \text{clear}(A), s) \land \Box(\text{on}(A, B) \rightarrow \Diamond \text{clear}(A)) \)

• Two possibilities:

  – \( f^+ = \text{Progress}(\Diamond \text{clear}(A), s) \land \Box(\text{on}(A, B) \rightarrow \Diamond \text{clear}(A)) \)
  – \( f^+ = \text{clear}(A) \land \Box(\text{on}(A, B) \rightarrow \Diamond \text{clear}(A)) \)

  – \( f^+ = \text{TRUE} \land \Box(\text{on}(A, B) \rightarrow \Diamond \text{clear}(A)) \)
  – \( f^+ = \Box(\text{on}(A, B) \rightarrow \Diamond \text{clear}(A)) \)
Performance Graphs

- No Control (breadth-first)
- Control 1
- Control 2

Control 1 fails on 1 problem of size 11