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 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 result(a, s) to denote the state reached after executing the action a.
  - result(a_1, s_0) = s_1
  - result(a_2, s_1) = s_2
  - …
  - 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*):
  - $\forall s \neg \text{open}(s) \rightarrow \text{open}(\text{result}(\text{open\_door}, s))$
  - $\forall s \text{ open}(s) \rightarrow \neg \text{open}(\text{result}(\text{close\_door}, s))$
  - $\forall s \neg \text{on}(s) \rightarrow \text{on}(\text{result}(\text{switch\_on\_light}, s))$
  - $\forall s \text{ on}(s) \rightarrow \neg \text{ on}(\text{result}(\text{switch\_off\_light}, s))$

- We also need to specify what does NOT change after applying each operator! (aka *frame axioms*):
  - $\forall s \text{ open}(s) \rightarrow \text{open}(\text{result}(\text{switch\_on\_light}, s))$
  - $\forall s \neg \text{open}(s) \rightarrow \neg \text{open}(\text{result}(\text{switch\_off\_light}, s))$
  - $\forall s \text{ on}(s) \rightarrow \text{on}(\text{result}(\text{close\_door}, s))$
  - $\forall s \neg \text{on}(s) \rightarrow \neg \text{on}(\text{result}(\text{open\_door}, s))$
  - …

An alternative (and more elegant) solution

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

- Successor state axioms list all the ways a given predicate may change its value over time:
  - $\forall a, s$ open(result(a, s)) $\leftrightarrow$[($\neg$open(s) $\wedge$ a = open_door) $\vee$ (open(s) $\wedge$ $a \neq$ close_door)]
  - $\forall a, s$ $\neg$on(result(a, s)) $\leftrightarrow$[(on(s) $\wedge$ a = switch_off_light) $\vee$ ($\neg$on(s) $\wedge$ $a \neq$ switch_on_light)]
  - …

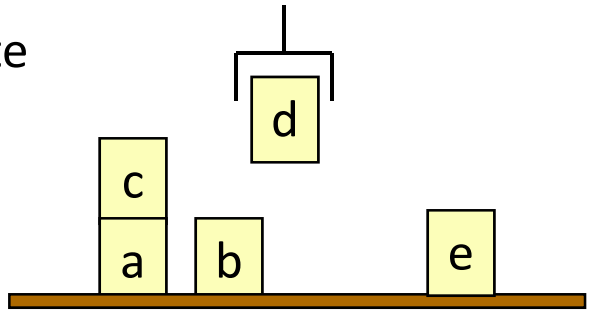
Planning Systems for solving the Frame Problem

- 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).

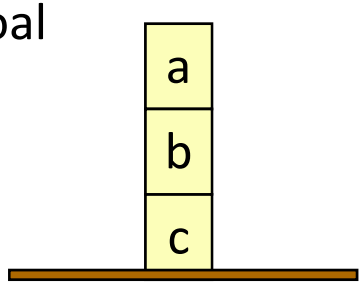
The Blocks World Revisited

- 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



goal



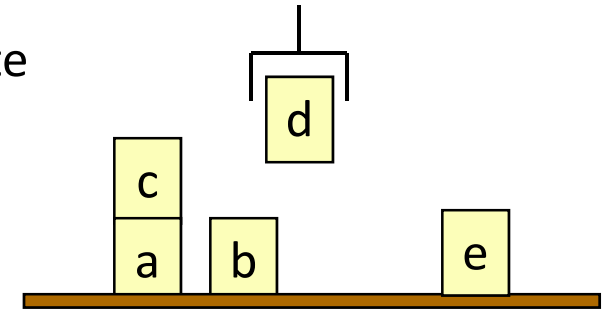
The Blocks World Revisited

- 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

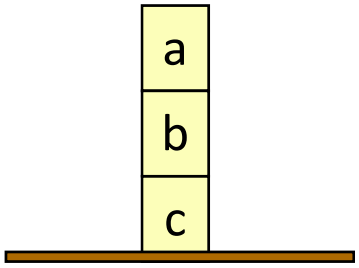
The Blocks World Revisited

- Representing states:

initial state



goal



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

g = {on-table(c), on (b, c), on (a, b), clear(a)}

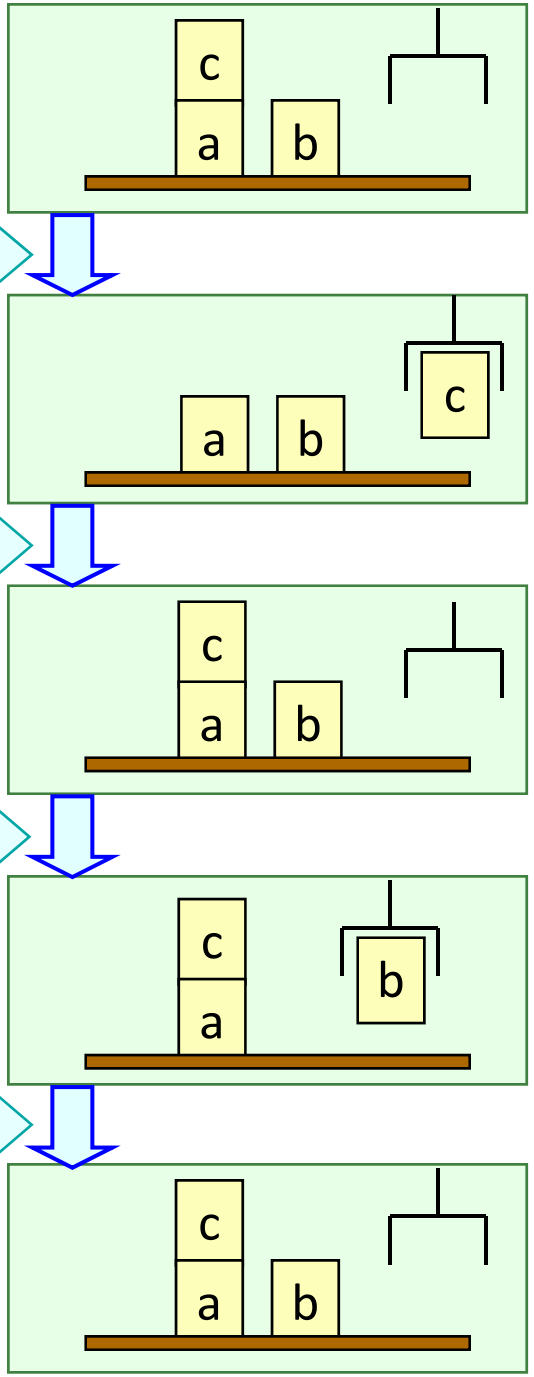
The Blocks World Revisited

unstack(x,y)
- Precond: on(x,y), clear(x), handempty
- Effects: ¬on(x,y), ¬clear(x), ¬handempty, holding(x), clear(y)

stack(x,y)
- Precond: holding(x), clear(y)
- Effects: ¬holding(x), ¬clear(y), on(x,y), clear(x), handempty

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

putdown(x)
- Precond: holding(x)
- Effects: ¬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 **TRUE** and **FALSE**.
  - Modal operators.
  - Bounded quantifiers.
  - The **GOAL** predicate.

Modal Operators

Operator	Symbol	Description	Example
Next	$\bigcirc f$	*f* will hold in the next state	$\bigcirc \text{on(A, B)}$
Eventually	$\Diamond f$	*f* will hold in a future state	$\Diamond \text{clear(A)}$
Always	$\Box f$	*f* holds in the current state and will hold in all future states	$\Box$ on-table(A)
Until	$f_1 \cup f_2$	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)	$\text{clear}(B) \cup \text{on-table}(A)$

Bounded Quantifiers

Let $g(x)$ such that $\{x : g(x)\} = \{x_1, x_2, ..., x_n\}$ is finite and easily computable:

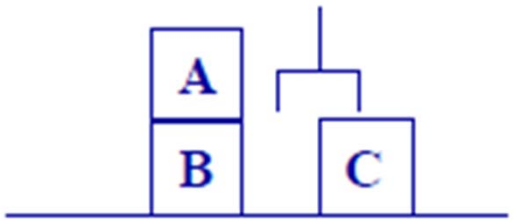
$$\forall [x : g(x)] f(x) = f(x_1) \wedge f(x_2) \wedge ... \wedge f(x_n)$$

$$\exists [x : g(x)] f(x) = f(x_1) \vee f(x_2) \vee ... \vee f(x_n)$$

Bounded Quantifiers (Example)

$$\forall x\left[\text{on-table}(x)\right]\Diamond\text{holding}(x)$$

$$\Diamond\text{holding}(B)\wedge\Diamond\text{holding}(C)$$

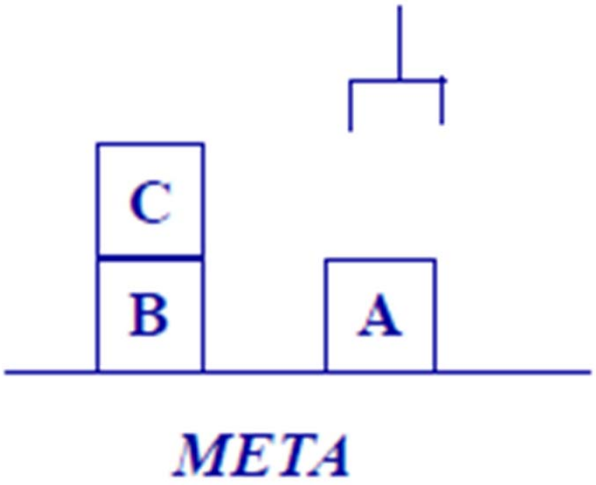


$$\exists x\left[\text{clear}(x)\right]\bigcirc\text{holding}(x)$$

$$\bigcirc\text{holding}(A)\vee\bigcirc\text{holding}(C)$$

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))
  - …



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

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

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

$f_1 = \text{on-table}(x)$

$f_2 = \neg\exists [y : \text{GOAL}(\text{on}(x,y))]$

$f_3 = \neg\text{GOAL}(\text{on-table}(x))$

$f_4 = \neg\text{GOAL}(\text{holding}(y))$

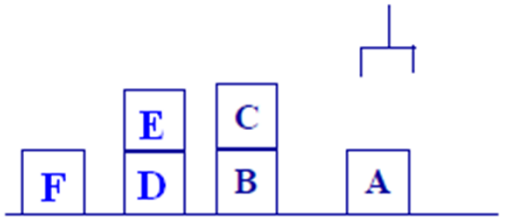
$f_5 = \neg\text{GOAL}(\text{clear}(y))$

$f_6 = \forall [z : \text{GOAL}(\text{on}(x,z))] z = y$

$f_7 = \forall [z : \text{GOAL}(\text{on}(z,y))] z = x$

$f_8 = \text{goodtower-below}(y)$

ESTADO ACTUAL



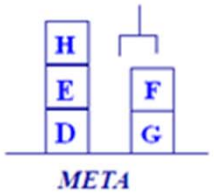
META

$$\text{badtower}(x) \leftrightarrow \text{clear}(x) \wedge (\text{GOAL}(\text{holding}(x)) \vee \neg\text{goodtower-below}(x))$$

Defining Control Rules

- Control Rule 1:
  - A goodtower must always remain goodtower

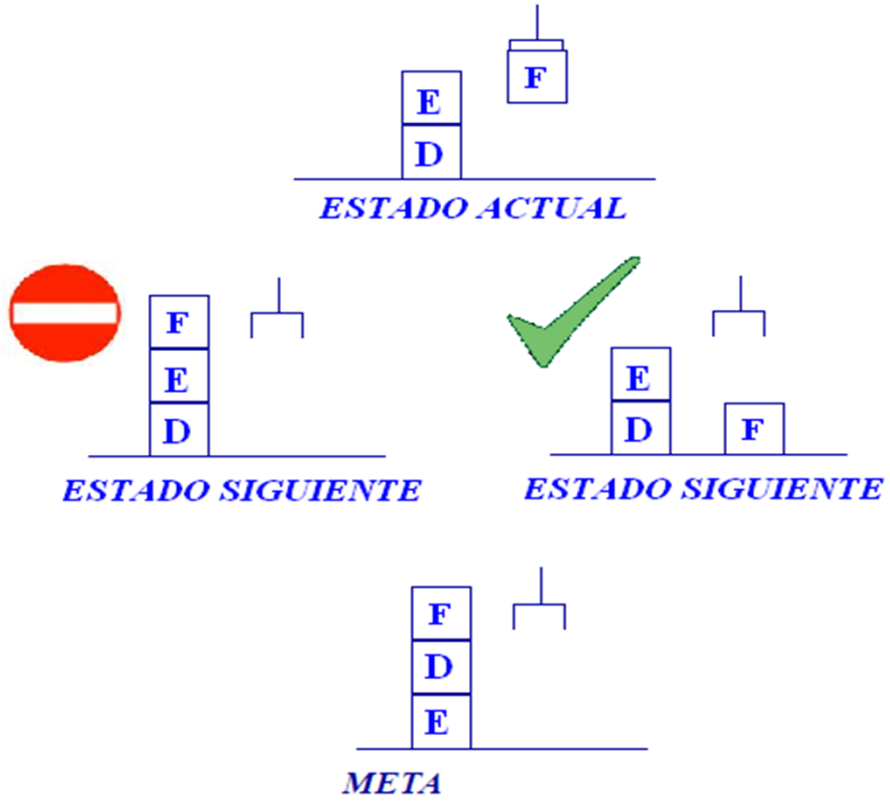
Note that this CR allows stacking a block in a badtower!!



$$CR_1 = \Box\left(\forall\left[x : \text{clear}(x)\right]\text{goodtower}(x) \rightarrow \bigcirc\left(\text{clear}(x) \vee \exists\left[y : \text{on}(y, x)\right]\text{goodtower}(y)\right)\right)$$

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



$$CR_2 = CR_1 \wedge \Box\left(\text{badtower}(x) \rightarrow \bigcirc\left(\neg\exists\left[y : \text{on}(y, x)\right]\right)\right)$$

TLPlan Pseudocode

PROCEDURE TLPlan(s, f, g, π)

1. $f^+ \leftarrow$ Progress(f, s)
2. **IF** f^+ = FALSE **THEN RETURN** FAIL
3. **IF** s satisfies g **THEN RETURN** π
4. A $\leftarrow$ {Actions that can be executed from s}
5. **IF** A = Φ **THEN RETURN** FAIL
6. Choose an action $a \in A$
7. **DO** $s^+ \leftarrow \gamma(s, a)$
8. **DO** $\pi^+ = \pi.a$
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 ($f = f_1 \wedge f_2$):
f^+ := Progress(f_1, s) $\wedge$ Progress (f_2, s)
CASE ($f = \neg f_1$):
f^+ := $\neg$Progress (f_1, s)
CASE ($f = \circ f_1$):
f^+ := f_1
CASE ($f = f_1 \cup f_2$):
f^+ := Progress (f_2, s) $\vee$ (Progress(f_1, s) $\wedge$ f)
CASE ($f = \Diamond f_1$):
f^+ := Progress(f_1, s) $\vee$ f
CASE ($f = \Box f_1$):
f^+ := Progress (f_1, s) $\wedge$ f
CASE ($f = \forall[x{:}g(x)]f_1$):

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

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

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

Progress Example #1

- f = □on(A, B)

- f^+ = Progress(□on(A, B), s)
- f^+ = Progress(on(A, B), s) ∧ □on(A, B)

- Two possibilities:
  - f^+ = TRUE ∧ □on(A, B)
  - f^+ = □on(A, B)

  - f^+ = FALSE ∧ □on(A, B)
  - f^+ = FALSE

- Conclusion:
  - □ checks that the control rule holds in the current state.
  - If the rule holds, then □ 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

- $f = \Box(on(A, B) \rightarrow \bigcirc clear(A))$

- $f^+ = Progress(\Box(on(A, B) \rightarrow \bigcirc clear(A)), s)$
- $f^+ = Progress(on(A, B) \rightarrow \bigcirc clear(A), s) \wedge \Box(on(A, B) \rightarrow \bigcirc clear(A))$

- Two possibilities:
  - $f^+ = Progress(\bigcirc clear(A), s) \wedge \Box(on(A, B) \rightarrow \bigcirc clear(A))$
  - $f^+ = clear(A) \wedge \Box(on(A, B) \rightarrow \bigcirc clear(A))$

  - $f^+ = TRUE \wedge \Box(on(A, B) \rightarrow \bigcirc clear(A))$
  - $f^+ = \Box(on(A, B) \rightarrow \bigcirc clear(A))$

p	q	→
0	0	1
0	1	1
1	0	0
1	1	1

Performance Graphs