TEMPORAL LOGIC & PLANNING

kopriva@agents.felk.cvut.cz

Quick Review of First Order Logic

- First Order Logic (FOL):
  - constant symbols, function symbols, predicate symbols
  - logical connectives ($\vee$, $\wedge$, $\neg$, $\Rightarrow$, $\Leftrightarrow$), quantifiers ($\forall$, $\exists$), punctuation
  - Syntax for formulas and sentences

    $on(A,B) \wedge on(B,C)$

    $\exists x\ on(x,A)$

    $\forall x\ (ontable(x) \Rightarrow clear(x))$
- First Order Theory *T*:
  - "Logical" axioms and inference rules – encode logical reasoning in general
  - Additional "nonlogical" axioms – talk about a particular domain
  - Theorems: produced by applying the axioms and rules of inference
- Model: set of objects, functions, relations that the symbols refer to
  - For our purposes, a model is some state of the world *s*
  - In order for *s* to be a model, all theorems of *T* must be true in *s*
  - $s \models on(A,B)$ read "*s* satisfies *on*(*A*,*B*)" or "*s* models *on*(*A*,*B*)"
    - means that *on*(*A*,*B*) is true in the state *s*

Linear Temporal Logic

- Modal logic: FOL plus *modal operators*
- Linear Temporal Logic (LTL):
  - Purpose: to express a limited notion of time
    - An infinite sequence $\langle 0, 1, 2, \ldots \rangle$ of time instants
    - An infinite sequence $M = \langle s_0, s_1, \ldots \rangle$ of states of the world
  - Modal operators to refer to the states in which formulas are true:

$\bigcirc f$	- *next f*	- *f* holds in the next state, e.g., $\bigcirc$ *on*(*A*,*B*)
$\Diamond f$	- *eventually f*	- *f* either holds now or in some future state
$\Box f$	- *always f*	- *f* holds now and in all future states
$f_1 \cup f_2$	- f_1 *until* f_2	- f_2 either holds now or in some future state, and f_1 holds until then

  - Propositional constant symbols TRUE and FALSE

Linear Temporal Logic (continued)

- Quantifiers cause problems with computability
  - Suppose $f(x)$ is true for infinitely many values of x
  - Problem evaluating truth of $\forall x \; f(x)$ and $\exists x \; f(x)$

- Bounded quantifiers
  - Let $g(x)$ be such that $\{x : g(x)\}$ is finite and easily computed

    $\forall[x{:}g(x)]\; f(x)$
    - means $\forall x \; (g(x) \Rightarrow f(x))$
    - expands into $f(x_1) \wedge f(x_2) \wedge \ldots \wedge f(x_n)$

    $\exists[x{:}g(x)]\; f(x)$
    - means $\exists x \; (g(x) \wedge f(x))$
    - expands into $f(x_1) \vee f(x_2) \vee \ldots \vee f(x_n)$

Models for LTL

- A model is a triple (M, s_i, v)
  - $M = \langle s_0, s_1, \ldots \rangle$ is a sequence of states
  - s_i is the *i*'th state in M,
  - v is a *variable assignment* function
    - a substitution that maps all variables into constants

- Write $(M,s_i,v) \models f$

  to mean that $v(f)$ is true in s_i

- Always require that

  $(M, s_i, v) \models \text{TRUE}$

  $(M, s_i, v) \models \neg\text{FALSE}$

Examples

Suppose $M = \langle s_0, s_1, \ldots \rangle$

$(M, s_0, v) \models \bigcirc\bigcirc\, on(A,B)$ means A is on B in s_2

- Abbreviations:

  $(M, s_0) \models \bigcirc\bigcirc\, on(A,B)$no free variables, so v is irrelevant:

  $M \models \bigcirc\bigcirc\, on(A,B)$ if we omit the state, it defaults to s_0

- Equivalently,

  $(M, s_2, v) \models on(A,B)$ same meaning with no modal operators

  $s_2 \models on(A,B)$ same thing in ordinary FOL

- $M \models \Box\neg holding(C)$
  - in every state in M, we aren't holding C

- $M \models \Box(on(B, C) \Rightarrow (on(B, C) \cup on\ (A, B)))$
  - whenever we enter a state in which B is on C, B remains on C until A is on B.

Where We're Going

- Basic idea:
  - TLPLan does a forward search, using LTL to do pruning tests
  - Input includes a current state *s*, and a **control formula** *f* written in LTL
    - If *f* isn't satisfied, then *s* is unacceptable => backtrack
    - Else keep going

- We'll need to augment LTL to include a way to refer to goal states
  - Include a GOAL operator such that GOAL(*f*) means *f* is true in every goal state
  - $((M,s_i,V),g) \models \text{GOAL}(f)$ iff $(M,s_i,V) \models f$ for every $s_i \in g$

- Next, some examples of control formulas

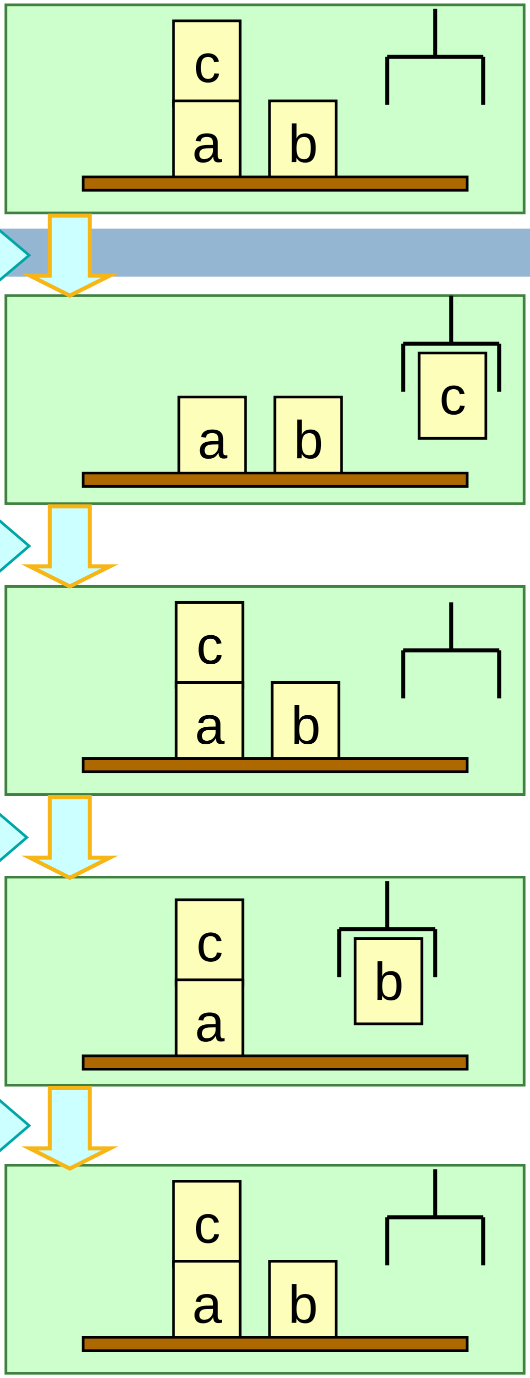
Example: Blocks World

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: ontable(x), clear(x), handempty
- Effects: ¬ontable(x), ¬clear(x), ¬handempty, holding(x)

putdown(x)
- Precond: holding(x)
- Effects: ¬holding(x), ontable(x), clear(x), handempty



Supporting Axioms

- Want to define conditions under which a stack of blocks will never need to be moved
- If x is the top of a stack of blocks, then we want *goodtower*(x) to hold if
  - x doesn't need to be anywhere else
  - None of the blocks below x need to be anywhere else
- Definitions to support this:
  - $\mathit{goodtower}(x) \Leftrightarrow \mathit{clear}(x) \land \neg \mathrm{GOAL}(\mathit{holding}(x)) \land \mathit{goodtowerbelow}(x)$
  - $\mathit{goodtowerbelow}(x) \Leftrightarrow$

$$\begin{aligned} &[\mathit{ontable}(x) \land \neg\exists[y{:}\mathrm{GOAL}(\mathit{on}(x,y))]] \\ &\lor \exists[y{:}\mathit{on}(x,y)]\ \{\neg\mathrm{GOAL}(\mathit{ontable}(x)) \land \neg\mathrm{GOAL}(\mathit{holding}(y)) \\ &\qquad \land \neg\mathrm{GOAL}(\mathit{clear}(y)) \land \forall[z{:}\mathrm{GOAL}(\mathit{on}(x,z))]\ (z = y) \\ &\qquad \land \forall[z{:}\mathrm{GOAL}(\mathit{on}(z,y))]\ (z = x) \land \mathit{goodtowerbelow}(y)\} \end{aligned}$$

  - $\mathit{badtower}(x) \Leftrightarrow \mathit{clear}(x) \land \neg\mathit{goodtower}(x)$

Blocks World Example (continued)

Three different control formulas:

(1) Every goodtower must always remain a goodtower:

$$\Box\Big(\forall[x{:}clear(x)]\, goodtower(x) \Rightarrow \bigcirc(clear(x) \lor \exists[y{:}on(y,x)]\, goodtower(y))\Big)$$

(2) Like (1), but also says never to put anything onto a badtower:

$$\begin{aligned}\Box\Big(&\forall[x{:}clear(x)]\, goodtower(x) \Rightarrow \bigcirc(clear(x) \lor \exists[y{:}on(y,x)]\, goodtower(y))\\ &\land badtower(x) \Rightarrow \bigcirc(\neg\exists[y{:}on(y,x)]\,)\Big)\end{aligned}$$

(3) Like (2), but also says never to pick up a block from the table unless you can put it onto a goodtower:

$$\begin{aligned}\Box\Big(&\forall[x{:}clear(x)]\, goodtower(x) \Rightarrow \bigcirc(clear(x) \lor \exists[y{:}on(y,x)]\, goodtower(y))\\ &\land badtower(x) \Rightarrow \bigcirc(\neg\exists[y{:}on(y,x)]\,)\\ &\land (ontable(x) \land \exists[y{:}\mathrm{GOAL}(on(x,y))]\, \neg goodtower(y))\\ &\qquad \Rightarrow \bigcirc(\neg holding(x))\Big)\end{aligned}$$

Outline of How TLPlan Works

- Recall that TLPLan's input includes a current state *s*, and a control formula *f* written in LTL
  - How can TLPLan determine whether there exists a sequence of states *M* beginning with *s*, such that $M \models f$?

- We can compute a formula f^+ such that for every sequence $M = \langle s, s^+, s^{++}, \ldots \rangle$,
  - $M \models f^+$ iff $M^+ = \langle s^+, s^{++}, \ldots \rangle$ satisfies f^+
  - f^+ is called the **progression** of *f* through *s*

- If $f^+ = \text{FALSE}$ then no M^+ can satisfy f^+
  - Thus no *M* can satisfy *f*, so TLPLan can backtrack
- Otherwise, need to determine whether there is an M^+ that satisfies f^+
  - For every child s^+ of *s*, call TLPLan recursively on s^+ and f^+

- How to compute the progression of *f* through *s*?

```
Procedure Progress($f, s$)
  Case
  1. $f$ contains no temporal operators:
                         $f^+ :=$ TRUE if $s \models f$, FALSE otherwise.
  2. $f = f_1 \wedge f_2$:       $f^+ := \text{Progress}(f_1, s) \wedge \text{Progress}(f_2, s)$
  3. $f = \neg f_1$:            $f^+ := \neg\text{Progress}(f_1, s)$
  4. $f = \bigcirc f_1$:        $f^+ := f_1$
  5. $f = f_1 \,\mathsf{U}\, f_2$:       $f^+ := \text{Progress}(f_2, s) \vee (\text{Progress}(f_1, s) \wedge f)$
  6. $f = \Diamond f_1$:        $f^+ := \text{Progress}(f_1, s) \vee f$
  7. $f = \Box f_1$:            $f^+ := \text{Progress}(f_1, s) \wedge f$
  8. $f = \forall[x{:}\gamma(x)]\, f_1$:  $f^+ := \bigwedge_{i=1,\ldots,n} \text{Progress}(f_i, s)$
  9. $f = \exists[x{:}\gamma(x)]\, f_1$:  $f^+ := \bigvee_{i=1,\ldots,n} \text{Progress}(f_i, s)$
```

where $\{c_1, \ldots, c_n\} = \{x : s \models \gamma(x)\}$, and $f_i = f$ with x replaced by c_i

Boolean simplification rules:

1. $[\text{FALSE} \wedge \phi \mid \phi \wedge \text{FALSE}] \mapsto \text{FALSE}$,
2. $[\text{TRUE} \wedge \phi \mid \phi \wedge \text{TRUE}] \mapsto \phi$,
3. $\neg\text{TRUE} \mapsto \text{FALSE}$,
4. $\neg\text{FALSE} \mapsto \text{TRUE}$.

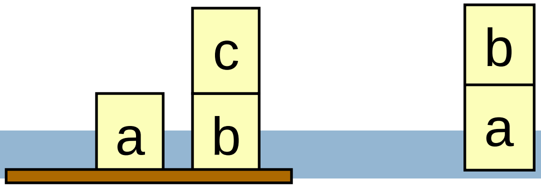
Examples

- Suppose *f* = □ on(a,b)
  - *f* ⁺ = Progress(*on*(a,b), *s*) ∧ □ *on*(a,b)
  - If *on*(a,b) is true in *s* then
    - *f* ⁺ = TRUE ∧ □ *on*(a,b)
    - simplifies to □ *on*(a,b)
  - If *on*(a,b) is false in *s* then
    - *f* ⁺ = FALSE ∧ □ *on*(a,b)
    - simplifies to FALSE
- Summary:
  - □ generates a test on the current state
  - If the test succeeds, □ propagates it to the next state

Examples (continued)

- Suppose $f = \square(\mathit{on}(a,b) \Rightarrow \bigcirc \mathit{clear}(a))$
  - $f^{+} = \text{Progress}[\square(\mathit{on}(a,b) \Rightarrow \bigcirc \mathit{clear}(a)), s]$
  - $\quad = \text{Progress}[\text{on}(a,b) \Rightarrow \bigcirc \mathit{clear}(a), s] \wedge \square(\mathit{on}(a,b) \Rightarrow \bigcirc \mathit{clear}(a))$
  - If *on*(a,b) is true in *s*, then
    - $f^{+} = \text{clear}(a) \wedge \square(\mathit{on}(a,b) \Rightarrow \bigcirc \mathit{clear}(a))$
      - Since *on*(a,b) is true in *s*,
        s^{+} must satisfy *clear*(a)
      - The “always” constraint is propagated to s^{+}
  - If *on*(a,b) is false in *s*, then
    - $f^{+} = \square(\mathit{on}(a,b) \Rightarrow \bigcirc \mathit{clear}(a))$
      - The “always” constraint is propagated to s^{+}

Example



- $s = \{ontable(\mathrm{a}), ontable(\mathrm{b}), clear(\mathrm{a}), clear(\mathrm{c}), on(\mathrm{c,b})\}$
- $g = \{on(\mathrm{b, a})\}$
- $f = \Box\forall[x{:}clear(x)]\ \{(ontable(x) \wedge \neg\exists[y{:}\mathrm{GOAL}(on(x,y))]) \Rightarrow \bigcirc\neg holding(x)\}$
  - never pick up a block *x* if *x* is not required to be on another block *y*

- $f^+ = \mathrm{Progress}(f,s) \wedge f$

- Progress(*f*,*s*)

$$= \mathrm{Progress}(\ \forall[x{:}clear(x)]\ \{(ontable(x) \wedge \neg\exists[y{:}\mathrm{GOAL}(on(x,y))]) \Rightarrow \bigcirc\neg holding(x)\},s)$$

$$= \mathrm{Progress}((ontable(\mathrm{a}) \wedge \neg\exists[y{:}\mathrm{GOAL}(on(\mathrm{a},y))]) \Rightarrow \bigcirc\neg holding(\mathrm{a})\},s)$$

$$\wedge\ \mathrm{Progress}((ontable(\mathrm{b}) \wedge \neg\exists[y{:}\mathrm{GOAL}(on(\mathrm{b},y))]) \Rightarrow \bigcirc\neg holding(\mathrm{b})\},s)$$

$$= \neg holding(\mathrm{a}) \wedge \mathrm{TRUE}$$

- $f^+ = \neg holding(\mathrm{a}) \wedge \mathrm{TRUE} \wedge f$

$$= \neg holding(\mathrm{a}) \wedge \Box\forall[x{:}clear(x)]\ \{(ontable(x) \wedge \neg\exists[y{:}\mathrm{GOAL}(on(x,y))]) \Rightarrow \bigcirc\neg holding(x)\}$$

Pseudocode for TLPlan

- Nondeterministic forward search
  - Input includes a control formula *f* for the current state *s*
  - When we expand a state *s*, we progress its formula *f* through *s*
  - If the progressed formula is false, *s* is a dead-end
  - Otherwise the progressed formula is the control formula for *s*'s children

```
Procedure TLPlan ($s, f, g, \pi$)
        $f^+ \leftarrow$ Progress ($f, s$)
        if $f^+$ = FALSE then return failure
        if $s$ satisfies $g$ then return $\pi$
        $A \leftarrow$ {actions applicable to $s$}
        if $A$ = empty then return failure
        nondeterministically choose $a \in A$
        $s^+ \leftarrow \gamma(s,a)$
        return TLPlan ($s^+, f^+, g, \pi.a$)
```

Discussion

- 2000 International Planning Competition
  - TALplanner: same kind of algorithm, different temporal logic
    - received the top award for a "hand-tailored" (i.e., domain-configurable) planner
- TLPlan won the same award in the 2002 International Planning Competition
- Both of them:
  - Ran several orders of magnitude faster than the "fully automated" (i.e., domain-independent) planners
    - especially on large problems
  - Solved problems on which the domain-independent planners ran out of time/memory