Chapter 10
Control Rules in Planning

Dana S. Nau
University of Maryland
Fall 2009
Motivation

- Often, planning can be done much more efficiently if we have domain-specific information.

- Example:
  - Classical planning is EXPSPACE-complete.
  - Block-stacking can be done in time $O(n^3)$.

- But we don’t want to have to write a new domain-specific planning system for each problem!

- *Domain-configurable* planning algorithm
  - Domain-independent search engine (usually a forward state-space search).
  - Input includes domain-specific information that allows us to avoid a brute-force search.
    - Prevent the planner from visiting unpromising states.
Motivation (Continued)

- If we’re at some state $s$ in a state space, sometimes a domain-specific test can tell us that
  - $s$ doesn’t lead to a solution, or
  - for any solution below $s$, there’s a better solution along some other path

- In such cases we can to prune $s$ immediately

- Rather than writing the domain-dependent test as low-level computer code, we’d prefer to talk directly about the planning domain

- One approach:
  - Write logical formulas giving conditions that states must satisfy; prune states that don’t satisfy the formulas

- Presentation similar to the chapter, but not identical
  - Based partly on TLPlan [Bacchus & Kabanza 2000]
Quick Review of First Order Logic

- First Order Logic (FOL):
  - constant symbols, function symbols, predicate symbols
  - logical connectives ($\lor$, $\land$, $\neg$, $\Rightarrow$, $\Leftrightarrow$), quantifiers ($\forall$, $\exists$), punctuation
  - Syntax for formulas and sentences
    - $\exists x \ on(x,A)$
    - $\forall x \ (ontable(x) \Rightarrow clear(x))$

- 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 to express concepts that would be difficult to express within FOL
- 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., $\bigcirc \text{on}(A,B)$
    $\Diamond f$ - eventually $f$ - $f$ either holds now or in some future state
    $\square f$ - always $f$ - $f$ holds now and in all future states
    $f_1 \bigcup f_2$ - $f_1$ 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 $f(x_1) \land f(x_2) \land \ldots \land f(x_n)$

    $\exists [x:g(x)] \ f(x)$
    
    - means $\exists x \ (g(x) \land f(x))$
    - expands into $f(x_1) \lor f(x_2) \lor \ldots \lor f(x_n)$
Models for LTL

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

- Write \((M, s_i, \nu) \models f\)
  to mean that \(\nu(f)\) is true in \(s_i\)

- Always require that
  \[
  (M, s_i, \nu) \models \text{TRUE} \\
  (M, s_i, \nu) \models \neg \text{FALSE}
  \]
Examples

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

$$(M, s_0, v) \models \Diamond \Diamond \text{on}(A, B)$$  means $A$ is on $B$ in $s_2$

● Abbreviations:

$$(M, s_0) \models \Diamond \Diamond \text{on}(A, B)$$  no free variables, so $v$ is irrelevant:

$M \models \Diamond \Diamond \text{on}(A, B)$  if we omit the state, it defaults to $s_0$

● Equivalently,

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

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

● $M \models \Box \neg \text{holding}(C)$

• in every state in $M$, we aren’t holding $C$

● $M \models \Box (\text{on}(B, C) \Rightarrow (\text{on}(B, C) \cup \text{on}(A, B)))$

• whenever we enter a state in which $B$ is on $C$, $B$ remains on $C$ until $A$ is on $B$.  

Dana Nau: Lecture slides for Automated Planning
Licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License: http://creativecommons.org/licenses/by-nc-sa/2.0/
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 $\Rightarrow$ 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 $\text{GOAL}(f)$ 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 \( \text{goodtower}(x) \) 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:
  - \( \text{goodtower}(x) \iff \text{clear}(x) \land \neg \text{GOAL}(\text{holding}(x)) \land \text{goodtowerbelow}(x) \)
  - \( \text{goodtowerbelow}(x) \iff \)
    \[
    \begin{align*}
    &\text{ontable}(x) \land \neg \exists[y: \text{GOAL}(\text{on}(x,y))] \\
    \lor &\exists[y: \text{on}(x,y)] \{ \neg \text{GOAL}(\text{ontable}(x)) \land \neg \text{GOAL}(\text{holding}(y)) \\
    &\land \neg \text{GOAL}(\text{clear}(y)) \land \forall[z: \text{GOAL}(\text{on}(x,z))] (z = y) \\
    &\land \forall[z: \text{GOAL}(\text{on}(z,y))] (z = x) \land \text{goodtowerbelow}(y) \}
    \end{align*}
    \]
  - \( \text{badtower}(x) \iff \text{clear}(x) \land \neg \text{goodtower}(x) \)
Blocks World Example (continued)

Three different control formulas:

(1) Every goodtower must always remain a goodtower:
\[ \Box \left( \forall [x: \text{clear}(x)] \text{goodtower}(x) \Rightarrow \Diamond (\text{clear}(x) \lor \exists[y: \text{on}(y, x)] \text{goodtower}(y)) \right) \]

(2) Like (1), but also says never to put anything onto a badtower:
\[ \Box \left( \forall [x: \text{clear}(x)] \text{goodtower}(x) \Rightarrow \Diamond (\text{clear}(x) \lor \exists[y: \text{on}(y, x)] \text{goodtower}(y) \right. \\
\left. \land \text{badtower}(x) \Rightarrow \Diamond (\neg \exists[y: \text{on}(y, x)]) \right) \]

(3) Like (2), but also says never to pick up a block from the table unless you can put it onto a goodtower:
\[ \Box \left( \forall [x: \text{clear}(x)] \text{goodtower}(x) \Rightarrow \Diamond (\text{clear}(x) \lor \exists[y: \text{on}(y, x)] \text{goodtower}(y)) \right. \\
\left. \land \text{badtower}(x) \Rightarrow \Diamond (\neg \exists[y: \text{on}(y, x)]) \right. \\
\left. \land (\text{onTable}(x) \land \exists[y: \text{goal}(\text{on}(x, y))] \neg \text{goodtower}(y)) \Rightarrow \Diamond (\neg \text{holding}(x)) \right) \]
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 $\text{Progress}(f; s)$

Case

1. $f$ contains no temporal operators:
   \[ f^+ := \text{TRUE if } s \models f, \text{FALSE otherwise.} \]

2. $f = f_1 \land f_2$:
   \[ f^+ := \text{Progress}(f_1, s) \land \text{Progress}(f_2, s) \]

3. $f = \neg f_1$:
   \[ f^+ := \neg \text{Progress}(f_1, s) \]

4. $f = \diamond f_1$:
   \[ f^+ := f_1 \]

5. $f = f_1 \lor f_2$:
   \[ f^+ := \text{Progress}(f_2, s) \lor (\text{Progress}(f_1, s) \land f) \]

6. $f = \bigcirc f_1$:
   \[ f^+ := \text{Progress}(f_1, s) \lor f \]

7. $f = \Box f_1$:
   \[ f^+ := \text{Progress}(f_1, s) \land f \]

8. $f = \forall [x: \gamma(x)] f_1$:
   \[ f^+ := \land_{i=1,\ldots,n} \text{Progress}(f_i, s) \]

9. $f = \exists [x: \gamma(x)] f_1$:
   \[ f^+ := \lor_{i=1,\ldots,n} \text{Progress}(f_i, s) \]

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. $[\text{FALSE} \land \phi | \phi \land \text{FALSE}] \mapsto \text{FALSE}$,
   3. $\neg \text{TRUE} \mapsto \text{FALSE}$,

2. $[\text{TRUE} \land \phi | \phi \land \text{TRUE}] \mapsto \phi$,
   4. $\neg \text{FALSE} \mapsto \text{TRUE}$.
Examples

- Suppose $f = \square \text{on}(a,b)$
  - $f^+ = \text{Progress}(\text{on}(a,b), s) \land \square \text{on}(a,b)$
  - If $\text{on}(a,b)$ is true in $s$ then
    - $f^+ = \text{TRUE} \land \square \text{on}(a,b)$
    - simplifies to $\square \text{on}(a,b)$
  - If $\text{on}(a,b)$ is false in $s$ then
    - $f^+ = \text{FALSE} \land \square \text{on}(a,b)$
    - simplifies to $\text{FALSE}$

- Summary:
  - $\square$ generates a test on the current state
  - If the test succeeds, $\square$ propagates it to the next state
Examples (continued)

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

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

- $f^+ = \text{Progress}(f,s) \land f$

- $\text{Progress}(f,s)$
  \[
  = \text{Progress}(\forall [x:\text{clear}(x)] \{(\text{ontable}(x) \land \neg \exists [y:\text{GOAL}(\text{on}(x,y))]) \Rightarrow \lozenge \neg \text{holding}(x)\}, s) \\
  = \text{Progress}((\text{ontable}(a) \land \neg \exists [y:\text{GOAL}(\text{on}(a,y))]) \Rightarrow \lozenge \neg \text{holding}(a)\}, s) \\
  \land \text{Progress}((\text{ontable}(b) \land \neg \exists [y:\text{GOAL}(\text{on}(b,y))]) \Rightarrow \lozenge \neg \text{holding}(b)\}, s) \\
  = \neg \text{holding}(a) \land \text{TRUE}
  \]

- $f^+ = \neg \text{holding}(a) \land \text{TRUE} \land f$
  \[
  = \neg \text{holding}(a) \land \\
  \square \forall [x:\text{clear}(x)] \{(\text{ontable}(x) \land \neg \exists [y:\text{GOAL}(\text{on}(x,y))]) \Rightarrow \lozenge \neg \text{holding}(x)\}
  \]
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 $(s, f, g, \pi)$

$$f^+ \leftarrow \text{Progress} \ (f, s)$$

if $f^+ = \text{FALSE}$ then return failure
if $s$ satisfies $g$ then return $\pi$
$A \leftarrow \{\text{actions applicable to } s\}$
if $A = \text{empty}$ then return failure
nondeterministically choose $a \in A$

$s^+ \leftarrow \gamma(s, a)$
return TLPlan $(s^+, f^+, g, \pi.a)$
Blocks-World Results

Control 1 fails on 1 problem of size 11
Blocks-World Results

SatPlan fails on 3 problems of size 10

UCPOP fails on all problems of size 6

BlackBox fails on 1 problem of size 10

IPP fails on 2 problems of size 11 and 12, exceeds 1GB RAM on problems of size 13
Logistics-Domain Results

- IPP fails on problems of size > 9
- SatPlan fails on 2 problems of size 14 and 15
- BlackBox fails on problems of size > 15
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