Propositional Logic Intro, Syntax

CPSC 322 – Logic 1

Textbook §5.0 – 5.2

Lecture Overview

1. Recap
2. Logic Intro
3. Propositional Definite Clause Logic: Syntax

Planning as a CSP

- We don't have to worry about searching forwards if we set up a planning problem as a CSP
- To do this, we need to "unroll" the plan for a fixed number of steps
  - this is called the horizon
- To do this with a horizon of k:
  - construct a variable for each feature at each time step from 0 to k
  - construct a boolean variable for each action at each time step from 0 to $k-1$.

CSP Planning: Constraints

As usual, we have to express the preconditions and effects of actions:

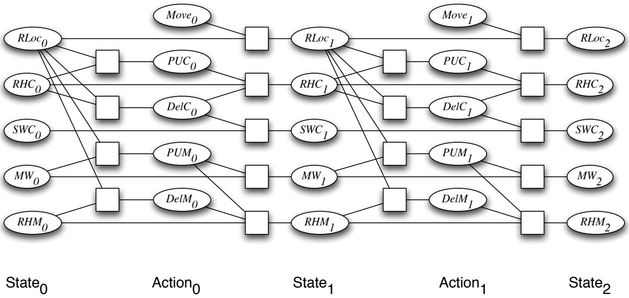
- precondition constraints
  - hold between state variables at time t and action variables at time t
  - specify when actions may be taken
- effect constraints
  - between state variables at time t, action variables at time t and state variables at time $t+1$
  - explain how state variables at time $t+1$ are affected by the action taken at time t
  - this includes both causal and frame axioms
    - basically, it goes back to the feature-centric representation the book discusses before STRIPS
    - of course, solving the problem this way doesn't mean we can't *encode* the problem using STRIPS

CSP Planning: Constraints

Other constraints we must/may have:

- initial state constraints constrain the state variables at time 0
- goal constraints constrain the state variables at time k
- action constraints
  - specify which actions cannot occur simultaneously
    - note that without these constraints, there's nothing to stop the planner from deciding to take several actions simultaneously
    - when the order between several actions doesn't matter, this is a good thing
  - these are sometimes called mutual exclusion (mutex) constraints
- state constraints
  - hold between variables at the same time step
  - they can capture physical constraints of the system
  - they can encode maintenance goals

CSP Planning: Robot Example



The constraints shown represent the preconditions of actions and the effects of actions.

Lecture Overview

1. Recap
2. Logic Intro
3. Propositional Definite Clause Logic: Syntax

Logic: A more general framework for reasoning

- Let's now think about how to represent a world about which we have only partial (but certain) information
- Our tool: propositional logic
- General problem:
  - tell the computer how the world works
  - tell the computer some facts about the world
  - ask a yes/no question about whether other facts must be true

Why Propositions?

We'll be looking at problems that could still be represented using CSPs. Why use propositional logic?

- Specifying logical formulae is often more natural than constructing arbitrary constraints
- It is easier to check and debug formulae than constraints
- We can exploit the Boolean nature for efficient reasoning
- We need a language for asking queries that may be more complicated than asking for the value of one variable
- It is easy to incrementally add formulae
- Logic can be extended to infinitely many variables (using logical quantification)
- This is a starting point for more complex logics (e.g., first-order logic) that do go beyond CSPs.

Representation and Reasoning System

Definition (RSS)

A Representation and Reasoning System (RRS) is made up of:

- syntax: specifies the symbols used, and how they can be combined to form legal sentences
- semantics: specifies the meaning of the symbols
- reasoning theory or proof procedure: a (possibly nondeterministic) specification of how an answer can be produced.

Using an RRS

1. Begin with a task domain.
2. Distinguish those things you want to talk about (the ontology).
3. Choose symbols in the computer to denote propositions
4. Tell the system knowledge about the domain.
5. Ask the system whether new statements about the domain are true or false.

Propositional Definite Clauses

- Propositional Definite Clauses: our first representation and reasoning system.
- Two kinds of statements:
  - that a proposition is true
  - that a proposition is true if one or more other propositions are true
- To define this RSS, we'll need to specify:
  - syntax
  - semantics
  - proof procedure

Lecture Overview

1. Recap
2. Logic Intro
3. Propositional Definite Clause Logic: Syntax

Propositional Definite Clauses: Syntax

Definition (atom)

An atom is a symbol starting with a lower case letter

Propositional Definite Clauses: Syntax

Definition (atom)

An atom is a symbol starting with a lower case letter

Definition (body)

A body is an atom or is of the form $b_1 \wedge b_2$ where b_1 and b_2 are bodies.

Propositional Definite Clauses: Syntax

Definition (atom)

An atom is a symbol starting with a lower case letter

Definition (body)

A body is an atom or is of the form $b_1 \wedge b_2$ where b_1 and b_2 are bodies.

Definition (definite clause)

A definite clause is an atom or is a rule of the form $h \leftarrow b$ where h is an atom and b is a body. (Read this as "h if b.")

Propositional Definite Clauses: Syntax

Definition (atom)

An atom is a symbol starting with a lower case letter

Definition (body)

A body is an atom or is of the form $b_1 \wedge b_2$ where b_1 and b_2 are bodies.

Definition (definite clause)

A definite clause is an atom or is a rule of the form $h \leftarrow b$ where h is an atom and b is a body. (Read this as "h if b.")

Definition (knowledge base)

A knowledge base is a set of definite clauses

Syntax: Example

The following are syntactically correct statements in our language:

- ai_is_fun

Syntax: Example

The following are syntactically correct statements in our language:

- ai_is_fun
- $ai\_is\_fun \leftarrow get\_good\_grade$

Syntax: Example

The following are syntactically correct statements in our language:

- ai_is_fun
- $ai\_is\_fun \leftarrow get\_good\_grade$
- $ai\_is\_fun \leftarrow get\_good\_grade \wedge not\_too\_much\_work$

Syntax: Example

The following are syntactically correct statements in our language:

- ai_is_fun
- $ai\_is\_fun \leftarrow get\_good\_grade$
- $ai\_is\_fun \leftarrow get\_good\_grade \wedge not\_too\_much\_work$
- $ai\_is\_fun \leftarrow$
  $get\_good\_grade \wedge not\_too\_much\_work \wedge remain\_awake$

Syntax: Example

The following are syntactically correct statements in our language:

- ai_is_fun
- $ai\_is\_fun \leftarrow get\_good\_grade$
- $ai\_is\_fun \leftarrow get\_good\_grade \wedge not\_too\_much\_work$
- $ai\_is\_fun \leftarrow$
  $get\_good\_grade \wedge not\_too\_much\_work \wedge remain\_awake$

The following statements are syntactically incorrect:

- $ai\_is\_fun \vee ai\_is\_boring$

Syntax: Example

The following are syntactically correct statements in our language:

- ai_is_fun
- $ai\_is\_fun \leftarrow get\_good\_grade$
- $ai\_is\_fun \leftarrow get\_good\_grade \land not\_too\_much\_work$
- $ai\_is\_fun \leftarrow get\_good\_grade \land not\_too\_much\_work \land remain\_awake$

The following statements are syntactically incorrect:

- $ai\_is\_fun \lor ai\_is\_boring$
- $ai\_is\_fun \land relaxing\_term \leftarrow get\_good\_grade \land not\_too\_much\_work$

Syntax: Example

The following are syntactically correct statements in our language:

- ai_is_fun
- $ai\_is\_fun \leftarrow get\_good\_grade$
- $ai\_is\_fun \leftarrow get\_good\_grade \wedge not\_too\_much\_work$
- $ai\_is\_fun \leftarrow$
  $get\_good\_grade \wedge not\_too\_much\_work \wedge remain\_awake$

The following statements are syntactically incorrect:

- $ai\_is\_fun \vee ai\_is\_boring$
- $ai\_is\_fun \wedge relaxing\_term \leftarrow$
  $get\_good\_grade \wedge not\_too\_much\_work$

Do any of these statements *mean* anything? Syntax doesn't answer this question.