Course on Automated Planning: Planning as SAT

Hector Geffner
ICREA & Universitat Pompeu Fabra
Barcelona, Spain

Logics

- Logics come in many forms and shapes, like propositional and predicate logic, modal logics, conditional logics, etc.

- Many uses in CS, AI, and Planning

- Some key dimensions:

  ▷ **Language:** defines the (valid) forms in the language, called **formulas**
  ▷ **Semantics:** defines the **meaning** of a formula as the set of models, and when a formula is **deducible** (follows) from another
  ▷ **Proof Theory:** provides 'local' (syntactic) methods for deriving new formulas from old

- Some key properties:

  ▷ Proof theory is **sound** if derived formulas deducible from old
  ▷ Proof theory is **complete** if **all** deducible formulas are derivable

Propositional Logic: Language

Propositional language **inductively** defined as set of expressions $\mathcal{P}$ such that

- propositional symbols p, q, r, . . . are in $\mathcal{P}$,
- $\neg A$ is in $\mathcal{P}$ if A in $\mathcal{P}$
- $(A\ op\ B)$ in $\mathcal{P}$ if A and B in $\mathcal{P}$, and $op \in \{\vee, \wedge, \supset, \ldots\}$
- (nothing else is in $\mathcal{P}$)

– Expressiones in $\mathcal{P}$ called **formulas**

– Often some parenthesis omitted if no ambiguity; e.g.,

$$p \wedge q \supset \neg r \vee s$$

abbreviates

$$((p \wedge q) \supset (\neg r \vee s))$$

Propositional Logic: Semantics

- States/worlds/truth valuations s are boolean (0/1) assignment over the propositional symbols in $\mathcal{P}$

- The truth value of a propositional symbol $p \in \mathcal{P}$ in s denoted as $s(p) \in \{0, 1\}$ $(0 = \mathbf{false}$, $1 = \mathbf{true})$

- The truth value A^s of arbitrary formulas A defined inductively as:

  - $s(A)$ if A is a propositional symbol,
  - $NEG(B^s)$ if A is of the form $\neg B$
  - $OP(B^s, C^s)$ if A is of the form $B\ op\ C$

  where NEG and $OP \in OR, AND, IMPLIES, \ldots$ are unary and binary functions mapping booleans into booleans as follows (**truth-tables**):

  NEG(0) = 1 , NEG(1)=0
  OR(0,0) = 0, else OR(*,*)=1
  AND(1,1)=1 , else AND(*,*)=0
  IMPLIES(1,0)=0, else IMPLIES(*,*) = 1, ...

Propositional Logic Semantics: Definitions

- A formula A is **satisfiable** if $A^s = 1$ for some state s
- Two formulas A and B are **logically equivalent** if $A^s = B^s$ for all states s
- A formula A is a **tautology** (**contradiction**) if A^s is true (**false**) for **all** states s
- A formula B **deductively follows** from $A_1, \ldots, A_n$, written $A_1, \ldots, A_n \models B$, if for all s, $B^s = 1$ if $A_1^s = \ldots = A_n^s = 1$

Proof Theory

- **Axiomatic Systems:** based on a few axiom schemas and one or two rules of inference (e.g., modus ponens with the form 'if $H \vdash A \supset B$ and $H \vdash A$, then $H \vdash B$). *Derivations often long and not natural.*

- **Natural Deduction:** based on no axioms and a several rules of inference. *Natural derivations can be constructed by hand, but difficult to control automatically*.

- **Resolution** based on no axioms and a **single** (resolution) rule of inference that works on **clauses** only (disjunction of possibly negated atoms, called **literals**).

Resolution

- The resolution rule of inference has the form:

  $$\text{if } p \vee C \text{ and } \neg p \vee C', \text{ then } C \vee C'$$

  where C and C' are (potentially empty) clauses, and clauses are regarded as *sets* of literals.

- The resolution rule used to derive a **contradiction** (empty clause) from the premises and the **negation** of the conclusion (all expressed as a set of clauses).

- Otherwise, resolution is **not complete** (it's **refutation complete**)

- Resolution (refutation) suitable for **automated** theorem proving, and simple to extend to **predicate logic**. Many refinements advanced, and it's at the basis of PROLOG ...

Example

Model the following argument in propositional logic and prove the conclusion semantically and by resolution.

> John killed Louis or Peter did it. If it was John, then Mary must have seen the killing and she must be shocked. Thus, if Mary is not shocked, Peter must have done it.

SAT and SAT Solvers

- Best **computational methods** for **checking validity** in propositional logic rely on SAT
- SAT is the problem of determining whether a set of **clauses** or **CNF formula** is satisfiable
- A clause is disjunction of **literals** where a literal is a **propositional symbol** or its **negation**

$$x \vee \neg y \vee z \vee \neg w$$

- Many problems can be mapped into SAT such as Planning, Scheduling, CSPs, Verification problems etc.
- SAT is an **intractable problem** (exponential in the worst case unless P=NP) yet very large SAT problems can be solved in practice
- Best SAT algorithms not based on either pure **case analysis** (model theory) or **resolution** (proof theory), but a **combination** of both

Davis and Putnam Procedure for SAT

- DP (DPLL) is a sound and complete proof procedure for SAT that uses resolution in a restricted form called **unit resolution**, in which one **parent clause** must be **unit clause**

- Unit resolution is very efficient (poly-time) but **not complete** (Example: $q \vee p$, $\neg q \vee p$, $q \vee \neg p$, $\neg q \vee \neg p$)

- When **unit resolution** gets stuck, DP picks undetermined Var, and **splits** the problem in two: one where Var is true, the other where it is false (**case analysis**)

```
 DP(clauses)
     Unit-resolution(clauses)
     if Contradiction, Return False
     else if all VARS determined, Return True
*    else pick non-determined VAR, and
     Return  DP(clauses + VAR)  OR  DP(clauses + NEG VAR)
```

Currently very large SAT problems can be solved. Criterion for **var selection** is critical, as **learning from conflicts** (not shown).

Planning as SAT (Kautz & Selman)

- Maps planning problem $P = \langle F, O, I, G \rangle$ with horizon n into a **set of clauses** $C(P, n)$, solved by **SAT solver** (satz,chaff,. . .).

- Theory $C(P, n)$ includes vars $p_0, p_1, \ldots, p_n$ and $a_0, a_1, \ldots, a_{n-1}$ for each $p \in F$ and $a \in O$

- $C(P, n)$ satisfiable **iff** there is a parallel plan with length n; in that case, plan extracted from satisfying assignment

- In parallel plan, **non-mutex** actions can be executed in parallel; two actions are **mutex** if one deletes precs/adds of the other (don't commute)

- **Optimal** parallel plans minimize number of time steps; obtained by starting with optimistic horizon n (lower bound), and increasing it by 1 til $C(P, n)$ satisfiable

Theory $C(P, n)$ for Problem $P = \langle A, O, I, G \rangle$

1. **Init:** p_0 for $p \in I$, $\neg q_0$ for $q \not\in I$
2. **Goal:** p_n for $p \in G$
3. **Actions:** For $i = 0, 1, \ldots, n-1$
   $a_i \supset p_i$ for $p \in Prec(a)$
   $a_i \supset p_{i+1}$ for each $p \in Add(a)$
   $a_i \supset \neg p_{i+1}$ for each $p \in Del(a)$
4. **NO-OPs:** For each p, and $i = 0, 1, \ldots, n-1$, 'dummy' **NO-OP**(p) action added, with precondition and add list p and empty delete list.
5. **Frame:** If a^1, . . . , a^m are the actions that add p, then for $i = 0, \ldots, n-1$:

$$\neg a_i^1 \wedge \cdots \wedge \neg a_i^m \supset \neg p_{i+1}$$

6. **Mutex:** If a and a' mutex, $\neg(a_i \wedge a'_i)$

- Current SAT/CSP formulations built on top of planning graph that extracts **implicit mutex relations** between **action pairs**, and between **atom pairs**.

Other variations in Classical Planning

Only if there is time . . .

- Regression Planning
- Graphplan
- Partial Order Causal Link (POCL) Planning

Regression Planning

Search backward from **goal** rather than forward from initial state:

- **initial** state σ_0 is G
- a **applicable** in σ if $Add(a) \cap \sigma \neq \emptyset$ and $Del(a) \cap \sigma = \emptyset$
- resulting state is $\sigma_a = \sigma - Add(a) + Prec(a)$
- terminal states σ if $\sigma \subseteq I$

Advantages/Problems:

\+ Heuristic $h(\sigma)$ for any σ can be computed by simple aggregation (max,sum, . . .) of estimates $g(p, s_0)$ for $p \in \sigma$ computed only **once** from s_0

\- Spurious states σ not reachable from s_0 often generated (e.g., where a block is on two blocks at the same time). A good h should make $h(\sigma) = \infty$ . . .

Variation: Parallel Regression Search

Search backward from goal assuming that **non-mutex actions** can be done in **parallel**

- The regression search is similar, except that **sets** of non-mutex actions A allowed: $Add(A) = \cup_{a \in A} Add(a)$, $Del(A) = \cup_{a \in A} Del(a)$, $Prec(A) = \cup_{a \in A} Prec(a)$.
- Resulting state from regression is $\sigma_A = \sigma - Add(A) + Prec(a)$

Advantages/Problems:

\+ Sometimes easier to compute optimal **parallel** plans than optimal **serial** plans

\+ Some heuristics provide tighter estimates of **parallel cost** than **serial cost** (e.g., $h = h1$)

\- **Branching factor** in parallel search (either forward or backward) can be very large (2^n if n applicable actions).

Parallel Regression Search with NO-OPs

- Assumes 'dummy' operator **NO-OP(p)** for each p with $Prec = Add = \{p\}$ and $Del = \emptyset$
- A set of non-mutex actions A (possibly including NO-OPs) applicable in σ if $\sigma \subseteq Add(A)$ and $Del(A) \cap \sigma = \emptyset$
- Resulting state is $\sigma = Prec(A)$
- Starting state $\sigma_0 = G$ and terminal states $\sigma \subseteq I$

Advantages/Problems:

- More actions to deal with

+ Enables certain compilation techniques as in Graphplan . . .

Graphplan (Blum & Furst): First Version

- Graphplan does an IDA* parallel regression search with NO-OPs over **planning graph** containing **propositional** and **action layers** P_i and A_i, $i = 0, \ldots, n$
  - P_0 contains the atoms true in I
  - A_i contains the actions whose precs are true in P_i
  - P_{i+1} contains the **positive** effects of the actions in A_i

- planning graph built til layer P_n where G appears, then **search** for plans with horizon $n-1$ invoked with $Solve(G, n)$ where
  - $Solve(G, 0)$ succeds if $G \subseteq I$ and fails otherwise, and
  - $Solve(G, n)$ mapped into $Solve(Prec(A), n-1)$, where A **is a set of non-mutex actions in layer** in A_{n-1} that covers G, i.e., $G \subseteq Add(A)$.

- If search fails, n increased by 1, and process is repeated

Graphplan: Real version

- The IDA* search is **implicit**; heuristic $h(\sigma)$ encoded in planning graph as **index of first layer P_i that contains σ**
- This heuristic, as defined above, corresponds to the $\mathbf{hmax} = \mathbf{h1}$ heuristic; Graphplan actually uses a more powerful admissible heuristic akin to h_2 ...
- Basic idea: extend **mutex** relations to **pairs** of actions and propositions in each layer $i > 0$ as follows:
  - p and q **mutex in** P_i if p and q are in P_i and the actions in A_{i-1} that support p and q are **mutex in** A_{i-1};
  - a and a' **mutex in** A_i if a and a' are in A_i, and they are **mutex** or $Prec(a) \cup Prec(a')$ contains a **mutex in** P_i
- The **index of first layer** in planning graph that contains a set of atoms P or actions A **without** a mutex, is a **lower bound**
- Thus, search can be **started** at level in which G appears without a mutex, and $Solve(P, i)$ needs to consider only sets of actions A in A_{i-1} that **do not contain a mutex**.

Partial Order Planning: Regression + Decomposition. Intuition

1. recursively **decompose** regression with goal $p_1, \ldots, p_n$ into n regressions with goals p_i, $i = 1, \ldots, n$;
2. **combine** resulting plans so that they **do not interfere** with each other

E.g.: let $G = \{p, q\}$, $I = \{r\}$, and two actions

$$a1\text{: } Prec(a1) = \{r\},\ Add(a1) = \{p\},\ Del(a1) = \{r\}$$

$$a2\text{: } Prec(a2) = \{r\},\ Add(a2) = \{q\},\ Del(a2) = \{\}$$

– $P1 = \{a1\}$ is a plan for p, and $P2 = \{a2\}$ a plan for q

– Yet $a1$ in $P1$ **deletes** a precondition of $a2$

– This 'threat' can be solved by forcing $a1$ **after** $a2$, i.e., $a2 \prec a1$.

Partial Order Causal Link planning is a formulation of POP that pursues 1 and 2 concurrently

Partial Plans and Causal Links

A **partial plan** P in POCL is a triple $(Steps, \mathcal{O}, CLs)$ where

– $Steps$ is a set of **actions** a_i

– $\mathcal{O}$ is a set of **precedence constraints** $a_i \prec a_j$

– CLs is a set of **causal links** $(a1, p, a2)$ meaning that that precondition p of $a2$ is achieved by action $a1$

- POCL extends partial plans til they become **complete** (to be defined)
- **States** σ in the search are partial plans
- **Initial state** (partial plan) is $P_0 = (\{Start, End\}, \{Start \prec End\}, \{\})$, where $Start$ and End are actions that summarize I and G: $Add(Start) = I$, $Prec(End) = G$

POCL Planning Algorithm

- A partial plan $P = (Steps, \mathcal{O}, CLs)$ is **complete** when ordering $\mathcal{O}$ is **consistent** and there is no **flaw** of the form:

  - ▷ **unsupported precondition:** a precond $p \in Prec(a)$ for $a \in Steps$ s.t. no CL (a', p, a) in CLs
  - ▷ **threatened causal link:** a CL (a', p, a) for $b \in Steps$ s.t. $p \in Del(b)$ and $a' \prec b \prec a$ is consistent with $\mathcal{O}$

- POCL search **starts** with the plan $P = P_0$ above, selecting a flaw in P, and trying each one of the repairs:

  - ▷ **Flaw #1:** fixed by selecting an action a', $p \in Add(a)$, and adding a' to $Steps$, $a' \prec a$ to $\mathcal{O}$, and (a', p, a) to CLs
  - ▷ **Flaw #2:** fixed by adding $b \prec a'$ or $a \prec b$ to $\mathcal{O}$

- The **terminal states** in search are the complete plans (**solutions**) or the inconsistent ones (**dead ends**)

Status of POCL Planning

- POP/POCL dominated planning research for 10-15 years, until Graphplan
- Unlike other approaches, can work with action **schemas**
- In recent years lost favor to Graphplan/SAT/CSP/HSP
- Recent comeback combined with heuristics in RePOP and CPT
- Holds promise as **branching scheme for temporal planning**