

- introduction
- planning problem coding
- representation factoring
- algorithms
- plan decoding

- the first works using **FOL** for plan generation in planning problems date back to the late 60's, using **resolution**.  
Their results couldn't be scale up to real problems.
- 1971: the relative success of **STRIPS** makes this program the paradigm for subsequent works in planning: **systematic search** in a **state space** or **partial plans**.
- then, for more than 20 years, it was believed that **logic was not applicable to practical problems** in planning
- until the work of Kautz & Selman [1992-1999]:  
We believe that the lesson of the 1970s should not have been that planning required specialized algorithms, that **first-order deductive theorem proving does not scale well**.  
By contrast, the past few years have seen dramatic progress in the size of problems that can be handled by **propositional satisfiability** testing programs.

## Applications: SATPLAN

---

- **new probabilistic algorithms based on local search**, introduced at the beginning of the 90's, make it possible to efficiently solve certain hard problems coded in propositional logic
- the size of the tractable problems is **several orders of magnitude** bigger than those with traditional methods
- so it is convenient to code the planning problem as a **propositional satisfiability** problem, instead of thinking it as a **FOL refutation problem**
- the loss in expressive power can be compensated with knowledge representation techniques applied to each particular problem
- let us analyze the difference between propositional satisfiability and FOL refutation coding

### Planning as Deduction

- it is the **traditional** strategy
- several **action axioms** are defined so as to characterized their effect when applied over states that satisfy certain preconditions
- there are also needed **frame axioms** that describe the predicate that are not affected by the actions
- example: a total of 11 axiom schemata are enough to characterize a simple block world using predicates:

`on(X, Y, I)   clear(X, I)   move(X, Y, Z, I)`

- example of action axiom:

$$\begin{aligned} \text{on}(X, Y, I) \wedge \text{clear}(X, I) \wedge \text{clear}(Z, I) \wedge \text{move}(X, Y, Z, I) \\ \Rightarrow \text{on}(X, Z, I + 1) \wedge \text{clear}(Y, I + 1) \end{aligned}$$

- then, planning is the **process of finding a deductive proof** that the initial conditions plus a sequence of action axioms imply the goal conditions
- example: you have to demonstrate that it is possible to infer from the set of axioms the following sentence

$$\begin{aligned} (\exists X_1, Y_1, Z_1, X_2, Y_2, Z_2) (\text{on}(a, b, 1)) \wedge \text{on}(b, \text{table}, 1) \wedge \text{clear}(a, 1) \\ \wedge \text{move}(X_1, Y_1, Z_1, 1) \wedge \text{move}(X_2, Y_2, Z_2, 2) \\ \Rightarrow \text{on}(b, a, 3) \end{aligned}$$

### Planning as Satisfiability (SATPLAN)

- a plan is **not** a theorem
- plans are obtained from **each model of a theory**
- this theory contains formulas that define:
  - the **general properties of actions** (effect axioms, preconditions, frame axioms)
  - the **initial and final conditions** of fluents (properties that change over time)
  - other **static properties**

## Applications: SATPLAN

---

- axiomatizations used in deductive planning **are not adequate** for satisfiability planning: they **admit too many models!**, most of them not intended.
- example: even if the formula

$$\begin{aligned} & (\exists X_1, Y_1, Z_1, X_2, Y_2, Z_2) (\text{on}(a, b, 1) \wedge \text{on}(b, \text{table}, 1) \wedge \text{clear}(a, 1) \\ & \quad \wedge \text{move}(X_1, Y_1, Z_1, 1) \wedge \text{move}(X_2, Y_2, Z_2, 2) \\ & \quad \Rightarrow \text{on}(b, a, 3) \end{aligned}$$

is true in every model of the axioms, **some of these models do not represent a valid plan.**

Like the model:

$$\left\{ \begin{array}{lll} \text{on}(a, b, 1), & \text{on}(b, \text{table}, 1), & \text{clear}(a, 1), \\ \text{on}(b, a, 2), & \text{on}(b, a, 3), & \text{clear}(\text{table}, 1), \\ \text{clear}(\text{table}, 2), & \text{clear}(\text{table}, 3) \end{array} \right\}$$

This means **the world changes but no action is executed!!**

- if we want that each model represent a possible plan, then these **abnormal models** must be eliminated by doing a **more careful specification** of the problem conditions
- another example: axioms only specify **what happens when an action is executed in a state that satisfy the preconditions**
- then it is possible a model like

$$\left\{ \begin{array}{lll} \text{on}(a, b, 1), & \text{on}(b, \text{table}, 1), & \text{clear}(a, 1), \\ \text{move}(b, \text{table}, a, 1), & \text{on}(b, a, 2), & \text{on}(b, a, 3) \end{array} \right\}$$

where preconditions of  $\text{move}(b, \text{table}, a, 1)$  do not hold, but it is consistent that the action occurs so the effect  $\text{on}(b, a, 2)$  still holds

### Advantages of SATPLAN

- it is easy to specify conditions on intermediate state, not only in the initial states
- for example, if block b or block d must not be clear on moment 5, then we add

$$\neg\text{clear}(b, 5) \vee \neg\text{clear}(d, 5)$$

to the specification

- this is much **more difficult** to do in a deductive scheme
- it is possible to express **relations between predicates** like

$$\text{clear}(X, I) \Leftrightarrow \neg(\exists Y)\text{on}(Y, X, I)$$

- SAT theorem provers **scale up much better** than FOL theorem provers

## Applications: SATPLAN

---

- axiomatization used for deductive planning must be modified
- it is safe to use these axioms in deduction: any conclusion from them is true. But it is not correct to state that any model which is consistent with them is a valid plan

### Basic Scheme for SATPLAN

1. **code the problem** in a set of axioms using propositional logic
2. **representation factoring** transforms the coding in order to make it computationally more amenable
3. use a systematic, or stochastic algorithm in order to find **a model** that satisfies the axioms
4. translate the truth-value assignments into a **plan** for the original problem

### 1. Problem Coding

- it is possible to use as **first approximation** the FOL axiomatization
- also similar to STRIPS: **every action** (operator) is characterized by a set of **preconditions** and a set of **postconditions**
- atomic time intervals are assumed, ie we suppose the world to be completely specified in certain **time points** by the fluent and static predicates
- also, we need a **finite number of time points**
- an **omniscient agent** is assumed: there are no probabilities in the world states, which are completely specified
- action only have **deterministic effects**: no probabilities, no nondeterminism on postconditions

- in general, the goal is **conjunctive**
- also, **preconditions are conjunctive**
- it is possible to relax these rules in some way without losing the computational properties
- general restriction on facts and actions in the problem are described as **axiom schemata**, that will be instantiated later
- the **domains** for each variables must be finite. Universal quantification is replaced by conjunctions, and existential quantification by disjunctions.
- the are predicates associated to **fluents** (properties that change with time) must have an additional argument corresponding to the time point
- it is necessary to **bound time intervals** to some maximum  $n$ , fixing then the maximum plan length before starting to look for them

- in case it is not possible to find a plan within the fixed bound, it is possible to **start again** solving the problem with a new bound (binary search of best plan)
- in this way, if an action is executed in time  $t$ , then their preconditions must hold on time  $t$ , and their effect on time  $t + 1$
- in some cases, it is useful to **duplicate time points** in such a way that even time points correspond to stable world states, while odd time points are situations where actions are being executed
- starting time is time 1, where we have a complete specification of initial properties
- a plan must be found before time  $n$  (or  $2n$ ), and the goal specification must contain every intended goal condition
- the absence of an atom in the specification of a state implies its truth value is unknown (**incomplete knowledge**).

- it is convenient to transform the corresponding formulas to CNF
- for coding action  $A$  with precondition  $P$  and consequences  $C$  we use the formula

$$A \Rightarrow P \wedge C$$

- this eliminates abnormal models of  $A \wedge P \Rightarrow C$  where the action is executed without the preconditions being true
- problem: **number of propositions!!** It is possible to find codification in order to keep this number under control

- Then we have
  - $|act|$  : number of **action** predicates
  - $|pred|$  : number of **fluent and static** predicates
  - $|dom|$ : maximum number of **domain constants**
  - $N$  : maximum **plan length** to search
  - $A_p$  : maximum predicate arity
  - $A_o$  : maximum action arity
  - $A = |act| * |dom|^{A_o}$  : upper bound on the number of **ground actions**
- then we need  $N * |act| * |dom|^{A_o}$  **propositions** to represent the execution of each ground action in each time point
- since systematic SAT solvers take exponential time in the number of propositions, and also a big number of propositions make stochastic solvers much slower, then it is desirable to **decrease this number**

- there are basically three ways of representing actions:
  - traditional representation (described so far)
  - simple splitting
  - overloaded splitting (bitwise)
- in **simple splitting** each  $n$ -ary action predicate is replaced during the codification by  $n$  unary fluents

- example:  $\text{move}(A, B, C, T)$  is replaced by

$$(\text{objectMove}(A, T) \wedge \text{sourceMove}(B, T) \wedge \text{destinationMove}(C, T))$$

- example:  $\text{paint}(A, \text{Color}, T)$  is replaced by

$$(\text{objectPaint}(A, T) \wedge \text{colorPaint}(\text{Color}, T))$$

- if we do this for each action, then the number of propositions needed to represent all grounded actions is reduced to  $N * |\text{act}| * |\text{dom}| * A_o$
- also type information can make this bound lower

- in simple splitting, only **instances of the same action** share predicates for each argument
- in **overloaded splitting**, it is allowed to use the same argument predicate for every action, with the addition of a constant representing the action
- example: `move(A, B, C, T)` is transformed into

`action(move, T) ∧ arg1(A, T) ∧ arg2(B, T) ∧ arg3(C, t)`

- example: `paint(A, Color, T)` is coded into

`action(paint, T) ∧ arg1(A, T) ∧ arg2(Color, T)`

- in this way the total number of propositions for ground actions is reduced to

$$N * (|acc| + |dom| * A_o)$$

- the number of action predicates is  $A_o + 1$
- in addition we use **bitwise coding** for this representation
- **every ground action is replaced by  $\lceil \log_2 A \rceil$  unary predicates**, assigning to each ground action a number between 0 and  $A - 1$
- example: if there are four possible ground actions in each  $T$ , then we introduce the predicates **bit1(T)** y **bit2(T)**.

- each of the new predicates represents the **binary encoding** of the ground action number
- then, in order to represent that in time point 1 action 2 is executed, we use the formula

$$(\neg \text{bit1}(1) \wedge \text{bit2}(1))$$

- now we need in total  $N * \lceil \log_2 A \rceil$  propositions
- but we don't have exclusive predicates for each action

- another issue is representing the **frame axioms**
- frame axioms represent the **continuity** of truth values for fluents that are not affected by actions
- we have in this case two possibilities: **classic representation** and **explanatory representation**
- under the classic representation, we have to establish explicitly which fluents **do not change** when an action is executed

$$\text{clear}(D, T) \wedge \text{move}(A, B, C, T) \Rightarrow \text{clear}(D, T + 1)$$

- if we add classical frame axioms for each action and each time point  $T$  to the other axioms, then we almost obtain a valid coding for our planning problem...

- the problem is that if **no action occurs in time  $T$** , then these frame actions cannot infer anything about the fluent truth value in time  $T + 1$
- in this way everything is possible to happen in this time point
- solution: add for each time point an **“at least one executed action”** axiom
- “at least one executed action” is a **disjunction of all ground action** for every time point

$\text{move}(a, a, a, 1) \vee \text{move}(a, a, b, 1) \vee \dots \vee \text{move}(c, c, c, 1)$

...

$\text{move}(a, a, a, 3) \vee \text{move}(a, a, b, 3) \vee \dots \vee \text{move}(c, c, c, 3)$

- in the **explanatory representation** we explicitly represent the actions that **must necessarily be executed** in order to change a state
- example:

$$\text{clear}(d, T) \wedge \neg \text{clear}(d, T + 1) \\ \Rightarrow (\text{move}(a, b, d, T) \vee \text{move}(a, c, d, T) \vee \dots \vee \text{move}(c, \text{table}, d, T))$$

- that is, a change of truth value in a fluent can only happen if some action is executed
- using contrapositive, if **no action is executed then nothing changes**.

- in this way “at least one executed action” axioms **are not needed**
- since explanatory axioms do not force fluents that are unaffected by an action to remain without change, then it is possible **parallel action execution**
- all actions whose preconditions are satisfied at time point  $T$ , and whose effects are not contradictory can be executed in parallel in  $T$
- this modelling of parallelism can create valid plans with **non linear solutions**
- example: action  $A$  has precondition  $X$  and effect  $Y$ ; action  $B$  has precondition  $\neg Y$  and effect  $\neg X$
- while these two actions can be executed in parallel (their consequences are not contradictory), there is no **linear ordering possible**
- therefore, we need **exclusion axioms!**

- plan **linearity** can be guaranteed by restricting the set of actions that can be executed in parallel
- the **exclusion axioms** can be represented with **complete exclusion** or with **conflict exclusion**
- **Complete Exclusion**: for each time point, and each pair of ground actions  $A, B$ , we add the clause  $\neg A \vee \neg B$
- in this way **only one action occurs at each time point**, and the plan can be totally ordered
- **Conflict Exclusion**: only clauses of the form  $\neg A \vee \neg B$  are added for **each pair of conflicting actions** (if preconditions of one are inconsistent with consequences of the other)
- thus we assure plans **with partially ordered actions**. But then any total order obtained from this partial order is a valid plan.
- since this coding is smaller **it is preferable to use conflict exclusion**

### Evaluation

- conflict exclusion cannot be used simultaneously with splitting, since splitting makes each action to be represented by several predicates
- bitwise action representation do not need exclusion axioms, since in each time point we can only represent one binary coding. Thus we have total order, without parallelism
- the biggest problem for any of these methods is the representation of frame axioms, since there is a **combinatoric explosion** in combining actions with fluents

In short:

1. Action Representation
  - (a) Normal
  - (b) Splitting
  - (c) Overloaded Splitting
  - (d) Bitwise Overloaded Splitting
2. Frame Axioms Representation
  - (a) classic with “at least one executed action” axiom
  - (b) explanatory with exclusion axioms:
    - i. complete exclusion
    - ii. conflict exclusion

- there are **eight basic codings** to represent a planning problem in propositional logic, with four action representations, and two frame axioms representations
- unfortunately, the combinations that have a low number of propositions (splitting, bitwise) tend to **exploit in the number of clauses**, or in the **size of clauses** when the formula is translated into CNF

### 2. Representation Factoring

- **Factoring** is a technique that transform the codification of a problem in order to reduce its size
- factoring may **dramatically reduce** both the number of clauses and their sizes, in the case of simple and overloaded splitting
- the basic idea is to use only a **subset of the conjunction that represent an action**, whenever possible
- **partially instantiated actions** are introduced, eliminating redundant propositions. It is understood that each action is represented by the set of all possible total instantiation of the partial representaion.
- **bitwise action representation do not admit easy factoring** since partial conjunction of bit predicates are not useful, unless a special numbering technique is used.

### Factoring action and frame axioms

- example: consider the axiom  $\text{move}(a, b, c, T) \Rightarrow \text{clear}(b, T + 1)$
- after overloaded splitting results in

$$\begin{aligned} &\text{action}(\text{move}, T) \wedge \text{arg1}(a, T) \wedge \text{arg2}(b, T) \\ &\quad \wedge \text{arg3}(c, T) \Rightarrow \text{clear}(b, T + 1) \end{aligned}$$

- a similar axiom is generated from each pair of blocks  $S$  y  $D$  such that  $\text{move}(S, b, D, T)$  is a consistent action

- then  $\text{move}(S, b, D, T)$  is represented using the axiom

$$\text{moveArg2}(b, T) \Rightarrow \text{clear}(b, T + 1)$$

- thus it is not necessary to consider all  $|dom|^2$  values for  $\text{moveArg1}(\cdot, \cdot)$  and  $\text{moveArg3}(\cdot, \cdot)$
- note that the original action name **disappears completely from the vocabulary**

### Factoring exclusion axioms

- example: consider the axiom  $\neg\text{move}(a, b, c, T) \vee \neg\text{move}(a, b, d, T)$
- in this case we relate an action with another one, so the previous technique **is not possible**.
- on the other hand, **each value for the argument is independently excluded** by the action
- we can use the axioms

$$\neg\text{moveArgi}(a, T) \vee \neg\text{moveArgi}(b, T) \quad \forall i : 1 \dots 3, a \neq b$$

in order to make sure that at most one ground instance of  $\text{move}(\cdot, \cdot, \cdot, T)$  is activated at each time point

- to **complete the exclusion**, we need to ensure that at time point  $T$  there is no parallel application of different actions.
- we can include axioms

$$\neg \text{act}X\text{Arg1}(a, T) \vee \neg \text{act}Y\text{Arg1}(b, T) \quad \forall \text{action } X, Y$$

### Factoring “at least one executed action” axioms

- it is possible to use the **disjunction between all possible first arguments** for each action predicate, for each time point
- for example

$$\text{act1Arg1}(a, T) \vee \text{act1Arg1}(b, T) \vee \dots \vee \text{act1Arg1}(d, T)$$
$$\text{act2Arg1}(a, T) \vee \text{act2Arg1}(b, T) \vee \dots \vee \text{act2Arg1}(d, T)$$

- we avoid in this way the exponential number of atoms in the disjunctions due to the combinations of all possible arguments

- however, these factored axioms would allow a **partial representation of each action**
- so it is not possible to let an action being executed unless that action is completely instantiated at that time point
- the following set of axioms are needed in order to fully instantiate a partial action

$$\text{moveArg1}(a, T) \vee \text{moveArg1}(b, T) \vee \dots$$

$$\Leftrightarrow \text{moveArg2}(a, T) \vee \text{moveArg2}(b, T) \vee \dots$$

for each pair of arguments in the original action predicate

### 3. Algorithms

- once the coding of the problem is finished, and the theory is factored and transformed into CNF, we can apply to these formulas **any SAT algorithm**
  - algoritmo GSAT
  - algoritmo WALKSAT
  - algoritmo DPLL con RRR
- Kautz & Selman report in his work in 1996 that for big scale blocks world problems SATPLAN is **some orders of magnitude faster** than systematic algorithms
- Kautz, Levesque, Mitchell, & Selman showed that the critic measure for SAT problems complexity is the **ratio between the number of clauses and the number of propositions**
- for big ratios, the answer **NO** is easy; for small ratios, the answer **YES** is also easy
- there is a **thin boundary** between these two kind of instances in which we find the more difficult problems (ratio 4.3)

### 4. Plan Decoding

- in case of employing complex techniques for problem representation (splitting, factoring, etc), getting the model is not enough for determining the plan
- it is possible to use [automatic tools](#) in order to find which axioms are satisfied in each step, and determine their correspondence with the original axioms
- most of these tools can also help in the coding step

### SATPLAN: conclusions

- 8 basic strategies are possible for coding the problem
- there is a trade-off between the number of propositions and the number of clauses
- there is a trade-off between the size of the clauses and the number of the clauses
- it is possible to optimize the problem representation with factoring.
- the smallest coding is obtained using normal representation or simple splitting with explicit frame axioms
- explicit frame axioms are always better than classic ones

- parallel plans with normal coding have a shorter length, because less repetition of periodic axioms.
- the biggest coding is obtained using normal representation with classical frame axioms
- splitting can provide more flexibility, allowing reasoning about part of the actions
- splitting reduce considerably the number of propositions in normal representation
- bitwise representation gets the least number of proposition, but this extreme optimization does not have much effect on the general result

## References

---

- Logic for Computer Science: Foundations of Automatic Theorem Proving J.H. Gallier, Harper & Row, (1986)
- The Essence of Logic John Kelly, Prentice Hall, (1996)
- Symbolic Logic and Mechanical Theorem Proving Chin-Lian Chang, Richard Lee, Academic Press, (1973)
- Foundations of Logic Programming J. W. Lloyd, Springer Verlag, (1998)
- A Brief History of Logic, Moshe Vardi.
- Knowledge representation, reasoning and declarative problem solving. Chitta Baral. Cambridge University Press, (2003).
- Computational Logic: Memories of the Past and Challenges for the Future J. A. Robinson, in J. W. Lloyd (ed.) Computational Logic 2000, Springer.
- Henry A. Kautz, Bart Selman: Planning as Satisfiability. ECAI 1992: 359-363.
- B. Selman, D. Mitchell and H. Levesque. Generating hard satisfiability problems. Artificial Intelligence. Volume 81, Issue 1-2, 1995.