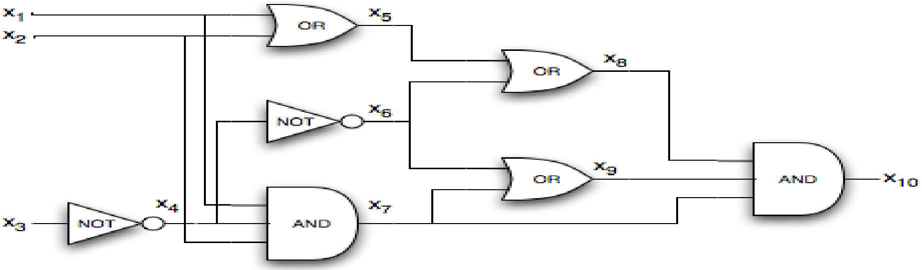
Practical SAT Solving

Lecture 2

Carsten Sinz, Tomáš Balyo | Apr 24, 2018

INSTITUTE FOR THEORETICAL COMPUTER SCIENCE



Encodings: Motivation

- A wide variety of problems can be encoded as SAT!
  - (Finite) arithmetic
  - Mathematical / practical combinatorial problems
  - Hardware / software verification problems
  - Planning problems
- Chosen encoding highly influences runtime of SAT solver
  - A lot of research on good encodings
  - . . . but still more an art than a science

Finite-Domain Variables

- Common in combinatorial problems: finite domain variables, e.g.: $x \in \{v_1, \ldots, v_n\}$
- Relationships between them expressed as equality-formulas, e.g.: $x = v_3 \Rightarrow y \neq v_2$.

Finite-Domain Variables

- Common in combinatorial problems: finite domain variables, e.g.: $x \in \{v_1, \dots, v_n\}$
- Relationships between them expressed as equality-formulas, e.g.: $x = v_3 \Rightarrow y \neq v_2$.
- Direct encoding / "one-hot-encoding":
  - Boolean variables x_v: "x takes value v"
  - Must encode that each variable takes exactly one value from its domain (using at-least-one/at-most-one constraints)
  - Encoding of variables' constraints simple

Finite-Domain Variables

- Common in combinatorial problems: finite domain variables, e.g.: $x \in \{v_1, \ldots, v_n\}$
- Relationships between them expressed as equality-formulas, e.g.: $x = v_3 \Rightarrow y \neq v_2$.
- Direct encoding / “one-hot-encoding”:
  - Boolean variables x_v: “x takes value v”
  - Must encode that each variable takes exactly one value from its domain (using at-least-one/at-most-one constraints)
  - Encoding of variables’ constraints simple
- Log-encoding / binary encoding:
  - Boolean variables b_i^x for $0 \leq i < \lceil \log_2 n \rceil$
  - Each value gets assigned a binary number, e.g. $v_1 \rightarrow 00, v_2 \rightarrow 01, v_3 \rightarrow 10$
  - Inadmissible values must be excluded, e.g.: $x \in \{v_1, v_2, v_3\}$ requires $(\overline{b_0^x} \vee \overline{b_1^x})$
  - Encoding of constraints can become complicated

Comparing Encodings

- Size: number of variables, number of clauses
- Propagation properties

At-Most-One Constraints

Definition

AtMostOne($x_1, \ldots, x_n$) is the constraint that no more than 1 variable / literal out of $x_1, \ldots, x_n$ is set to True.

At-Most-One Constraints

Definition

AtMostOne$(x_1, \ldots, x_n)$ is the constraint that no more than 1 variable / literal out of $x_1, \ldots, x_n$ is set to True.

- Alternative notations: $\leq 1\,(x_1, \ldots, x_n), \quad x_1 + \cdots + x_n \leq 1$

At-Most-One Constraints

Definition

AtMostOne$(x_1, \ldots, x_n)$ is the constraint that no more than 1 variable / literal out of $x_1, \ldots, x_n$ is set to True.

- Alternative notations: $\leq 1\,(x_1, \ldots, x_n), \quad x_1 + \cdots + x_n \leq 1$
- Naive (pairwise) encoding: add clauses $(\overline{x_i} \vee \overline{x_j})$ for $1 \leq i < j \leq n$
  - Results in $\binom{n}{2} = \frac{n \cdot (n-1)}{2}$ clauses

At-Most-One Constraints

Karlsruhe Institute of Technology

Definition

AtMostOne$(x_1, \ldots, x_n)$ is the constraint that no more than 1 variable / literal out of $x_1, \ldots, x_n$ is set to True.

- Alternative notations: $\leq 1\,(x_1, \ldots, x_n)$, $\quad x_1 + \cdots + x_n \leq 1$
- Naive (pairwise) encoding: add clauses $(\overline{x_i} \vee \overline{x_j})$ for $1 \leq i < j \leq n$
  - Results in $\binom{n}{2} = \frac{n \cdot (n-1)}{2}$ clauses
- Can we do better?

At-Most-One Constraints

Karlsruhe Institute of Technology

Definition

AtMostOne$(x_1, \ldots, x_n)$ is the constraint that no more than 1 variable / literal out of $x_1, \ldots, x_n$ is set to True.

- Alternative notations: $\leq 1\,(x_1, \ldots, x_n)$, $\quad x_1 + \cdots + x_n \leq 1$
- Naive (pairwise) encoding: add clauses $(\overline{x_i} \vee \overline{x_j})$ for $1 \leq i < j \leq n$
  - Results in $\binom{n}{2} = \frac{n \cdot (n-1)}{2}$ clauses
- Can we do better? Yes!

At-Most-One Constraints

Definition

AtMostOne$(x_1, \ldots, x_n)$ is the constraint that no more than 1 variable / literal out of $x_1, \ldots, x_n$ is set to True.

- Alternative notations: $\leq 1\,(x_1, \ldots, x_n)$, $\quad x_1 + \cdots + x_n \leq 1$
- Naive (pairwise) encoding: add clauses $(\overline{x_i} \vee \overline{x_j})$ for $1 \leq i < j \leq n$
  - Results in $\binom{n}{2} = \frac{n\cdot(n-1)}{2}$ clauses
- Can we do better? Yes!
- Encodings and their complexity (in number of clauses):
  - Pairwise encoding: $\mathcal{O}(n^2)$ (without extra variables)
  - Ladder encoding: $\mathcal{O}(n)$ (with n extra variables)
  - Tree encoding: $\mathcal{O}(n \log n)$ (with $\log n$ extra variables)

What is Planning

Informal Definition

Planning is the process of finding a plan, i.e., a sequence of actions that changes the state of the world from some initial state to a desired (goal) state.

Examples

- Delivering some packages
- Building a submarine
- Robot motion planning
- Fulfilling a scientific goal by an autonomous space probe

Trucking Example

- Initial State
  - There is a truck and a package in city A
  - There is a package in city B
- Goal
  - There are two packages in city C
- Possible Actions
  - (Un)loading packages from/on the truck, driving between cities

Formalizing Planning

Planning Problem Definition – SAS+ formalism

A planning problem instance Π is a tuple $(\mathcal{X}, \mathcal{A}, s_I, s_G)$ where

- $\mathcal{X}$ is a set of multivalued variables with finite domains.
  - each variable $x \in \mathcal{X}$ has a finite possible set of values $dom(x)$
- $\mathcal{A}$ is a set actions. Each action $a \in \mathcal{A}$ is a tuple $(pre(a), eff(a))$
  - $pre(a)$ is a set of preconditions of action a
  - $eff(a)$ is a set of effects of action a
  - both are sets of equalities of the form $x = v$ where $x \in \mathcal{X}$ and $v \in dom(x)$
- s_I is the initial state, it is a **full** assignment of the variables in $\mathcal{X}$
- s_G is the set of goal conditions, it is a set of equalities (same as $pre(a)$ and $eff(a)$)

Formalizing Planning II

World State

A state is full assignment of the variables in $\mathcal{X}$ (each variable $x \in \mathcal{X}$ has exactly one value assigned from its domain $dom(x)$. A state can be represented as a set of equalities.

The initial state s_I is a state. A state s is a goal state if $s_G \subseteq s$

Applicable Actions

An action $a \in \mathcal{A}$ is applicable in the state s if $pre(a) \subseteq s$

Applying an Action

When an action $a \in \mathcal{A}$ is applied in the state s it changes to the state s' such that $eff(a) \subseteq s'$ and the difference between s and s' is minimal (only variables used in $eff(a)$ are changed).

Formalizing Planning III

A Plan

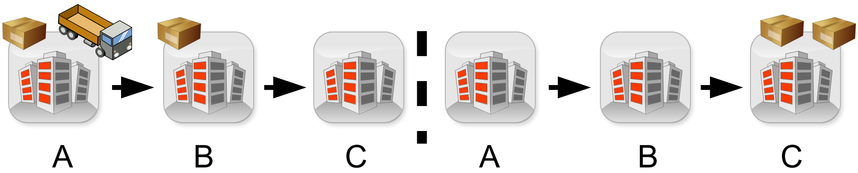
A plan for P for a planning problem $\Pi = (\mathcal{X}, \mathcal{A}, s_I, s_G)$ is sequence of actions $a_1, a_2, \ldots a_n$ such that

- $\forall i \; a_i \in \mathcal{A}$
- let $s_1 = s_I$ and $s_{i+1} = apply(s_i, a_i)$
- a_i is applicable in s_i
- $s_G \subseteq s_{n+1}$

If $P = \{a_1, a_2, \ldots a_n\}$ then n is the lenght of the plan P.

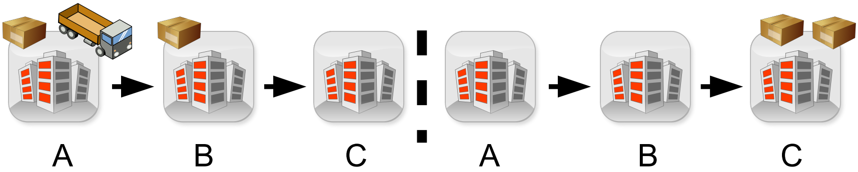
An optimal plan is a plan of shortest length.

Trucking Example



- variables: Truck Location T, $dom(T) = \{A, B, C\}$, Package Locations P_1 and P_2, $dom(P_1) = dom(P_2) = \{A, B, C, T\}$
- Initial state: $\{T = A, P_1 = A, P_2 = B\}$
- Goal: $\{P_1 = C, P_2 = C\}$
- Actions: $load(P_i, L) = (\{T = L, P_i = L\}, \{P_i = T\})$
  $unload(P_i, L) = (\{T = L, P_i = T\}, \{P_i = L\})$
  $drive(L_1, L_2) = (\{T = L_1\}, \{T = L_2\})$ where $i \in \{1, 2\}$ and $L, L_1, L_2 \in \{A, B, C\}$

Trucking Example



World State

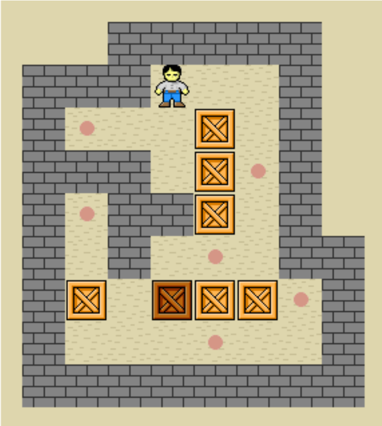
- $T = A, P_1 = A, P_2 = B$
- $T = A, P_1 = T, P_2 = B$
- $T = B, P_1 = T, P_2 = B$
- $T = B, P_1 = T, P_2 = T$
- $T = C, P_1 = T, P_2 = T$
- $T = C, P_1 = C, P_2 = C$

The Plan

- $load(P_1, A)$
- $drive(A, B)$
- $load(P_2, B)$
- $drive(B, C)$
- $unload(P_1, C), unload(P_1, C)$

Sokoban Example

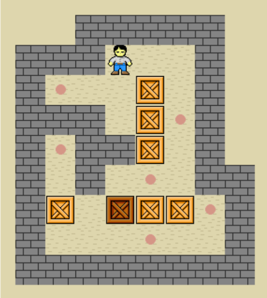
- Initial State
  - There is a worker and a bunch of boxes
- Goal
  - All the boxes must be in goal positions
- Possible Actions
  - moving with the worker
  - pushing a box
- Forbidden
  - to pull boxes
  - move through walls or boxes



http://wki.pe/Sokoban

Encoding Sokoban

- Variables – For each location we have variable, the domain is WORKER, BOX, EMPTY
- Initial State – assign values based on the picture
- Goal – goal position variables have value BOX



- Actions – move and push for each possible location
- $push(L_1, L_2, L_3) = (\{L_1 = W, L_2 = B, L_3 = E\}, \{L_1 = E, L_2 = W, L_3 = B\})$
- $move(L_1, L_2) = (\{L_1 = W, L_2 = E\}, \{L_1 = E, L_2 = W\})$

Encoding Planning into CNF

Is that even possible?

Encoding Planning into CNF

- We cannot encode the existence of a plan in general
- But we can encode the existence of plan up to some length

Encoding Planning into CNF

- We cannot encode the existence of a plan in general
- But we can encode the existence of plan up to some length

SATPLAN Algorithm

- INPUT: a planning problem Π
- OUTPUT: a plan P

for $m := 1, 2, \ldots$ **do**
  $F = \text{encodePlanExists}(\Pi, m)$
  if solver.isSat(F) **then**
    return extractPlan(Π, m, solver.solution)

Encoding Planning into CNF

The Task

Given a planning problem instance $\Pi = (\mathcal{X}, \mathcal{A}, s_I, s_G)$ and $k \in \mathbb{N}$ construct a CNF formula F such that F is satisfiable if and only if there is plan of length k for Π.

Encoding Planning into CNF

The Task

Given a planning problem instance $\Pi = (\mathcal{X}, \mathcal{A}, s_I, s_G)$ and $k \in \mathbb{N}$ construct a CNF formula F such that F is satisfiable if and only if there is plan of length k for Π.

We will need two kinds of variables

- Variables to encode the actions:
  a_i^t for each $t \in \{1, \ldots, k\}$ and $a_i \in \mathcal{A}$
- Variables to encode the states:
  $b_{x=v}^t$ for each $t \in \{1, \ldots, k+1\}$, $x \in \mathcal{X}$ and $v \in dom(x)$

In total we have $k|\mathcal{A}| + (k+1)\sum_{x \in \mathcal{X}} dom(x)$ variables

Encoding Planning into CNF

We will need 8 kinds of clauses

- The first state is the initial state
- The goal conditions are satisfied in the end
- Each state variable has at least one value
- Each state variable has at most one value
- If an action is applied it must be applicable
- If an action is applied its effects are applied in the next step
- State variables cannot change without an action between steps
- At most one action is used in each step

Encoding Planning into CNF

The first state is the initial state

$$\begin{aligned}&(b^1_{x=v})\\ &\forall(x=v)\in s_I\end{aligned} \tag{1}$$

The goal conditions are satisfied in the end

$$\begin{aligned}&(b^{n+1}_{x=v})\\ &\forall(x=v)\in s_G\end{aligned} \tag{2}$$

Encoding Planning into CNF

Each state variable has at least one value

$$(b^t_{x=v_1} \vee b^t_{x=v_2} \vee \cdots \vee b^t_{x=v_d})$$
$$\forall x \in X,\ \mathrm{dom}(x) = \{v_1, v_2, \ldots, v_d\},\ \forall t \in \{1, \ldots, k+1\} \tag{3}$$

Each state variable has at most one value

$$(\neg b^t_{x=v_i} \vee \neg b^t_{x=v_j})$$
$$\forall x \in X,\ v_i \neq v_j, \{v_i, v_j\} \subseteq \mathrm{dom}(x),\ \forall t \in \{1, \ldots, k+1\} \tag{4}$$

Encoding Planning into CNF

If an action is applied it must be applicable

$$(\neg a^t \vee b^t_{x=v})$$
$$\forall a \in \mathcal{A},\ \forall (x = v) \in \mathrm{pre}(a),\ \forall t \in \{1, \ldots, k\} \tag{5}$$

If an action is applied its effects are applied in the next step

$$(\neg a^t \vee b^{t+1}_{x=v})$$
$$\forall a \in \mathcal{A},\ \forall (x = v) \in \mathrm{eff}(a),\ \forall t \in \{1, \ldots, k\} \tag{6}$$

Encoding Planning into CNF

State variables cannot change without an action between steps

$$(\neg b^{t+1}_{x=v} \vee b^{t}_{x=v} \vee a^{t}_{s_1} \vee \cdots \vee a^{t}_{s_j})$$

$$\forall x \in X,\ \forall v \in \mathrm{dom}(x),\ \mathrm{support}(x = v) = \{a_{s_1}, \ldots, a_{s_j}\},\ \forall t \in \{1, \ldots, k\} \tag{7}$$

By support$(x = v) \subseteq \mathcal{A}$ we mean the set of *supporting actions* of the assignment $x = v$, i.e., the set of actions that have $x = v$ as one of their effects.

Encoding Planning into CNF

At most one action is used in each step

$$
\begin{gathered}
(\neg a_i^t \vee \neg a_j^t) \\
\forall \{a_i, a_j\} \subseteq \mathcal{A},\, a_i \neq a_j \; \forall t \in \{1, \ldots, k\}
\end{gathered} \tag{8}
$$

Encoding Planning into CNF

The Task Solved

Given a planning problem instance $\Pi = (\mathcal{X}, \mathcal{A}, s_I, s_G)$ and $k \in \mathbb{N}$ a CNF formula F, which is a conjunction of all the above described clauses is satisfiable if and only if there is plan of length k for Π.

Optimizations

- Better encoding of at-most-one
- Allowing several actions in each step
- Encoding variable transitions instead of variable values

SAT is NP-Hard – proof sketch

- Let M be a non-deterministic Turing machine that accepts an input x in $P(|x|)$ time, where P is a polynomial function.
  - M on x will use at most $P(|x|)$ tape entries
- M on input x as a SAS+ planning problem Π
  - State variables are the state of the TM and the $P(|x|)$ tape entries
  - The transition function table is encoded as actions
  - Initial state: tape contains input, TM state is initial state
  - Goal state: TM state is an accepting state
- Encode Π for plan lenght $k = P(|x|)$ into a CNF formula F_k
- F_k is SAT if and only if M accepts x in $P(|x|)$ time
- F_k has polynomial size w.r.t. to M and x

Planning with incremental SAT

- we are solving a sequence of similar formulas
- how do they differ?
- how to use an incremental solver in this case?

Planning with incremental SAT

- The formula F_k is the subset of F_{k+1} except for the goal clauses.
- The goal clauses will be added as removable (in this case, since they are unit, we can just assume them)

Incremental SATPLAN Algorithm

- INPUT: a planning problem Π
- OUTPUT: a plan P

$S = \text{initSolver}()$
addInitialStateClauses(S)
for $m := 1, 2, \ldots$ **do**
    addClausesForStep(m, S)
    assumeGoalConditionsAtStep(m, S)
    if satisfiable(S) **then return** extractPlan(Π, m, getValues(S))

The DIMSPEC format

- Many other (than planning) problems have a similar structure
  - for example bounded model checking
- They can be specified using the DIMSPEC format
- DIMSPEC is four cnf formulas, where the `"p cnf <n> <m>"` line is replaced by:
  - `i cnf <n> <m>` for the initial state specification (n variables)
  - `g cnf <n> <m>` for the goal state specification (n variables)
  - `u cnf <n> <m>` for the universal state specification (n variables)
  - `t cnf <n> <m>` for the specification of the transition (between two neigbhoring states) ($2n$ variables)

The DIMSPEC format example

```
c this is an example of a dimspec file
i cnf 5 3
-1 2 0
2 3 -5 0
4 0
g cnf 5 1
5 0
u cnf 5 2
-1 2 3 0
-3 4 5 0
t cnf 10 2
-2 7 8 0
-4 9 10 0
```

Planning as DIMSPEC

- Initial state specificaion clauses: $(b_{x=v})$ added $\forall (x = v) \in S_I$
- Goal state specificaion clauses: $(b_{x=v})$ added $\forall (x = v) \in S_G$
- Universal state specification clauses:
  - $(b_{x=v_1} \vee b_{x=v_2} \vee \cdots \vee b_{x=v_d})$ added $\forall x \in X$ where $\mathrm{dom}(x) = \{v_1, v_2, \ldots, v_d\}$ – at least one value
  - $(\overline{b_{x=i}} \vee \overline{b_{x=j}})$ added $\forall x \in X$ $i \neq j \in \mathrm{dom}(x)$ – at most one value
  - $(\overline{a} \vee b_{x=v})$ added $\forall a \in \mathcal{A},\ \forall (x = v) \in \mathrm{pre}(a)$ – action preconditions
  - $(\overline{a_i} \vee \overline{a_j})$ added $\forall i \neq j$ – at most one action
- Transition specification clauses
  - $(\overline{a} \vee b'_{x=v})$ added $\forall a \in \mathcal{A},\ \forall (x = v) \in \mathrm{eff}(a)$ – action effects
  - $(\overline{b'_{x=v}} \vee b_{x=v} \vee a_{s_1} \vee \cdots \vee a_{s_j})$ added $\forall x \in X,\ \forall v \in \mathrm{dom}(x)$ where $\mathrm{support}(x = v) = \{a_{s_1}, \ldots, a_{s_j}\}$ – values cannot change without a reason

Solving DIMSPEC

- Same as solving planning with incremental SAT

The Basic DISMPEC Solving Algorithm

- INPUT: a DIMSPEC problem
- OUTPUT: a truth assignment

$S = \text{initSolver}()$
addInitialStateClauses(S)
for $m := 1, 2, \ldots$ **do**
    addUniversalConditionsWithRenaming(m, S)
    if $m > 1$ **then** addTransitionalConditionsWithRenaming(m, S)
    assumeGoalConditionsWithRenaming(m, S)
    if satisfiable(S) **then return** getValues(S)