SMT

Severin Buchser

SMT:~$ Satisfiability Modulo Theories

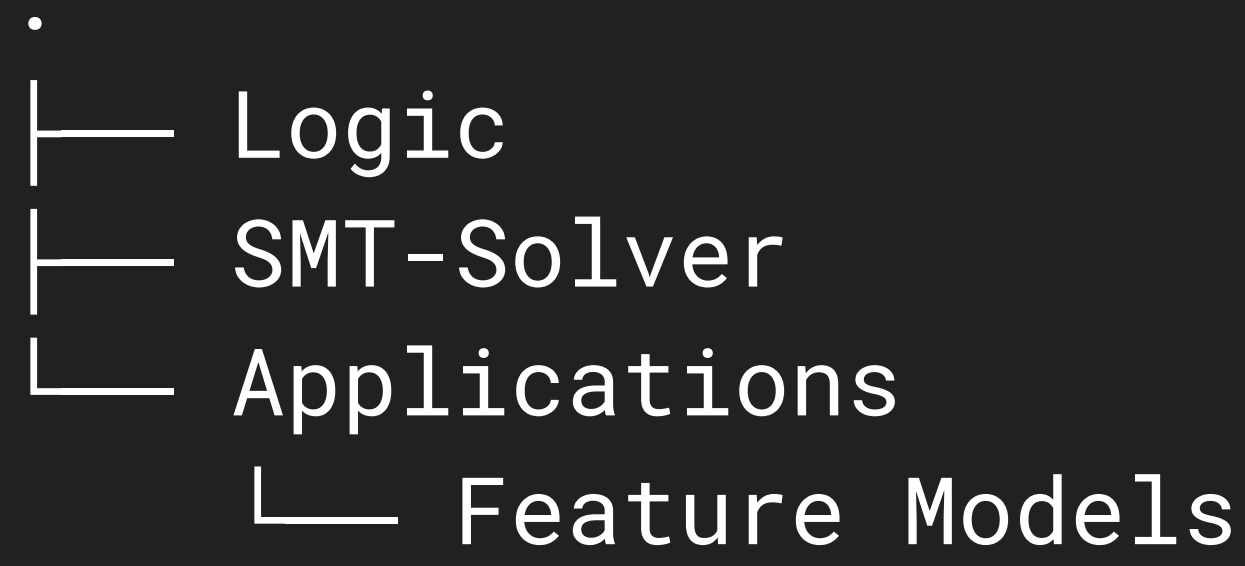
>> Satisfiability problem

Does a Formula have a solution?

$$F = x \land y \text{ (SAT)}$$

$$F = (x + y = y + x) \text{ (SMT)}$$

>> Produces some useful byproducts whilst solving


.
Logic
SMT-Solver
Applications
Feature Models

SMT:~$ Logic

>> SMT works with first-order-logic

>> f.o.l. extends propositional logic

>> there are some solvers which work with higher-order-logic

SMT:~/Logic$ First Order Logic

>> introduces predicates which can be on any domain:

- >> real arithmetic
- >> uninterpreted functions
- >> arrays and list structures
- >> bit vectors
- >> …

SMT:~/Logic$ Formula

A **term** is a variable t or $f(t_1,...,t_2)$ where t_i is a term.
An **atomic formula** is $t_1=t_2$ or $P(t_1,...,t_n)$ where t_i is a term.
A **formula** is:

>> $\neg p$, $(p \wedge q)$ or $(p \vee q)$ if p and q are formulas

>> $\exists x p$ if x is a variable and p is a formula

>> $\forall x p$ if x is a variable and p is a formula

(sidenote: in most SMT-Solvers quantifiers are not included)

SMT:~$ SMT-Solver

Problem:

Is a given formula satisfiable?

SMT:~/SMT-Solver$ Approaches

Two approaches:

>> Eager: Produces propositional logic formula and uses SAT

>> Lazy: DPLL(T) algorithm

Davis–Putnam–Logemann–Loveland

SMT:~/SMT-Solver$ DPLL (without Theories)

$$F = (a \lor \neg b) \land (\neg a \lor \neg b \lor \neg c)$$

1. Bring formula to CNF
2. Extract atomic formulas

1. $F = (a \lor \neg b) \land (\neg a \lor \neg b \lor \neg c)$
2. $\mathcal{A} = \{a, b, c\}$

SMT:~/SMT-Solver/DPLL$ # Partial Interpretation

f = False, t = True

P = {a = False, b = True}

F = (f ∨ ¬t) ∧ (¬f ∨ ¬t ∨ ¬c) = False

$P \Rightarrow$ ¬F

P = {a = False, b = False}

F = (f ∨ ¬f) ∧ (¬f ∨ ¬f ∨ ¬c) = True

$P \Rightarrow$ F

P = {a = True, b = True}

F = (t ∨ ¬t) ∧ (¬t ∨ ¬t ∨ ¬c) = ¬c

undetermined

SMT:~/SMT-Solver/DPLL$ Algorithm

```
def DPLL(F, P):
    if P ⇒ F:
        return True
    elif P ⇒ ¬F:
        return False
    else:
        x_i = x_j | x_j ∈ F ∧ x_j ∉ P
    return DPLL(F, union(P, {x_i = True})) or
           DPLL(F, union(P, {x_i = False}))
```

SMT:~/SMT-Solver/DPLL$ Example

P = {

a

}

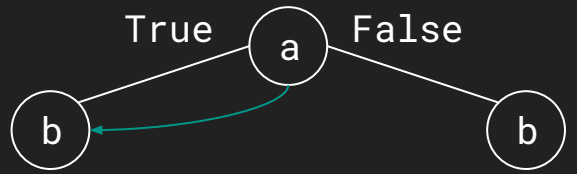
F = (a ∨ ¬b)∧(¬a ∨ ¬b ∨ ¬c)

SMT:~/SMT-Solver/DPLL$ # Example

P = {
  a = True

}

F = ¬b ∨ ¬c

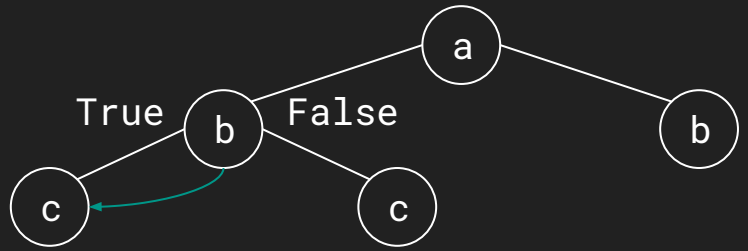


SMT:~/SMT-Solver/DPLL$ # Example

P = {
  a = True
  b = True

}

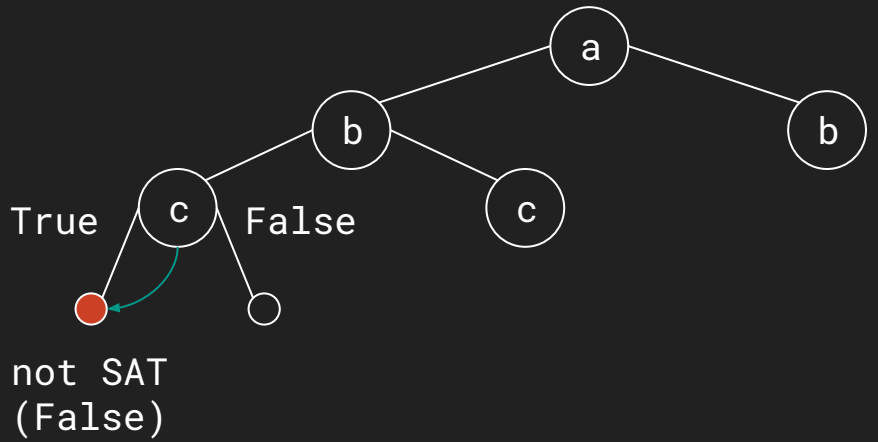
F = ¬c



SMT:~/SMT-Solver/DPLL$ # Example

P = {
  a = True
  b = True
  c = True
}

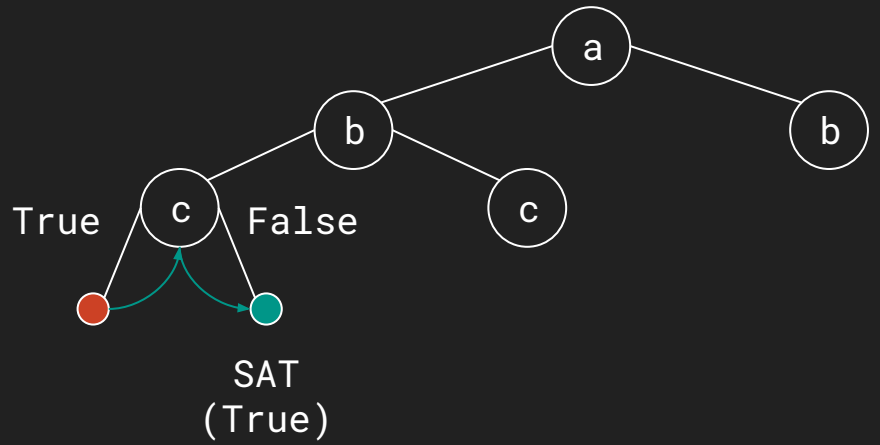
F = False



SMT:~/SMT-Solver/DPLL$ # Example

P = {
  a = True
  b = True
  c = False
}

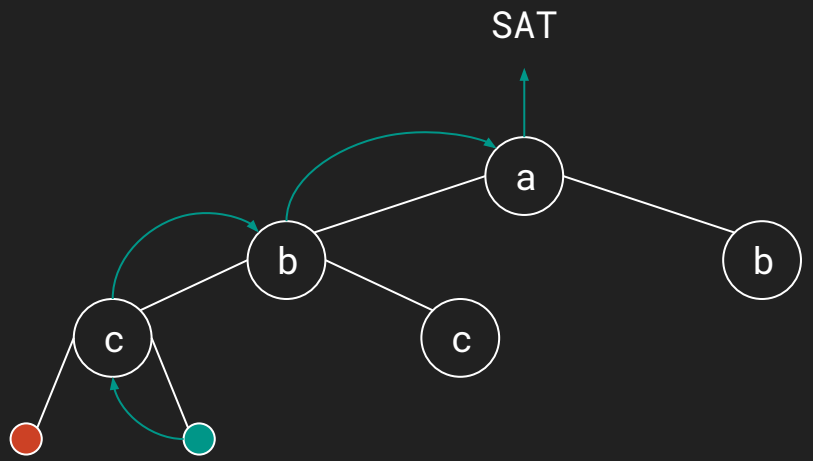
F = True



SMT:~/SMT-Solver/DPLL$ # Example

a = True
b = True
c = False

F = True



SMT:~/SMT-Solver$ DPLL(T)

>> Extend DPLL to DPLL(T)

>> T = Theory, i.e. addition with partial order (+, ≥, ≤)

>> Uses DPLL on the boolean domain and uses T-Solvers for predicates (atomic-formulas)

SMT:~/SMT-Solver/DPLL(T)$ Example

>> $F = (x \geq 0) \land (y \geq 0) \land (x + y \leq -1)$

atomic formulas:

- $f_1 = x \geq 0$
- $f_2 = y \geq 0$
- $f_3 = x + y \leq -1$

>> $F = f_1 \land f_2 \land f_3$

>> $F'_1 = a \land b \land c$ boolean skeleton

SMT:~/SMT-Solver/DPLL(T)$ Example

>> P'_1={a=True, b=True, c=True}

>> P_1={
    f_1 = x ≥ 0 = True
    f_2 = y ≥ 0 = True
    f_3 = x + y ≤ -1 = True
}

>> Ask T-solver if this interpretation is conflicting

SMT:~/SMT-Solver/DPLL(T)$ Example

>> $x \geq 0 \land y \geq 0 \Rightarrow x + y \geq 0$, conflicting with $x + y \leq -1$

>> $f_1 \land f_2 \land f_3$ is not SAT, so $\neg(f_1 \land f_2 \land f_3)$ universally

True

>> Add $\neg(a \land b \land c)$ to F'_1:

$F'_2 = F'_1 \land (\neg(a \land b \land c)) = a \land b \land c \land (\neg(a \land b \land c))$

>> Ask SAT-solver (DLLP) if F'_2 is SAT,

F'_2 is not SAT, so F is not SAT

SMT:~/SMT-Solver/DPLL(T)$ T-Solver

>> T-solvers are complicated

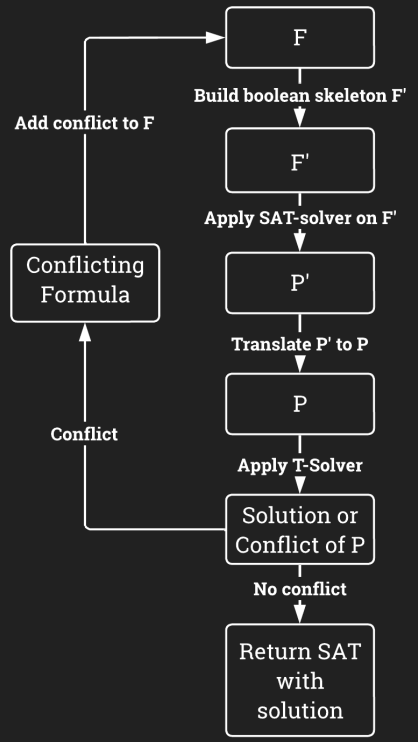
>> assume T-solvers as black box

>> T-solvers tell us if a partial interpretation is conflicting

>> T-solvers tell us a solution to a partial interpretation if not conflicting

SMT:~/SMT-Solver/DPLL(T)$ Algorithm

1. Build boolean skeleton
2. SAT-Solver (DPLL)
3. Translate to original problem
4. Apply T-Solver
5. If conflict add to F else return SAT



SMT:~/SMT-Solver$ Practice

>> Check if SAT

>> Find solution

>> Find all solutions

SMT:~$ Applications

>> Concolic Testing

>> Feature Models

>> …

SMT:~/Applications$ Feature Models

>> Models a domain, i.e. a Car, which has features

>> Features relate with constraints

SMT:~/Applications$ Feature Models

>> Features: Engine, Doors, Radio, Driver Seat

>> Constraints:

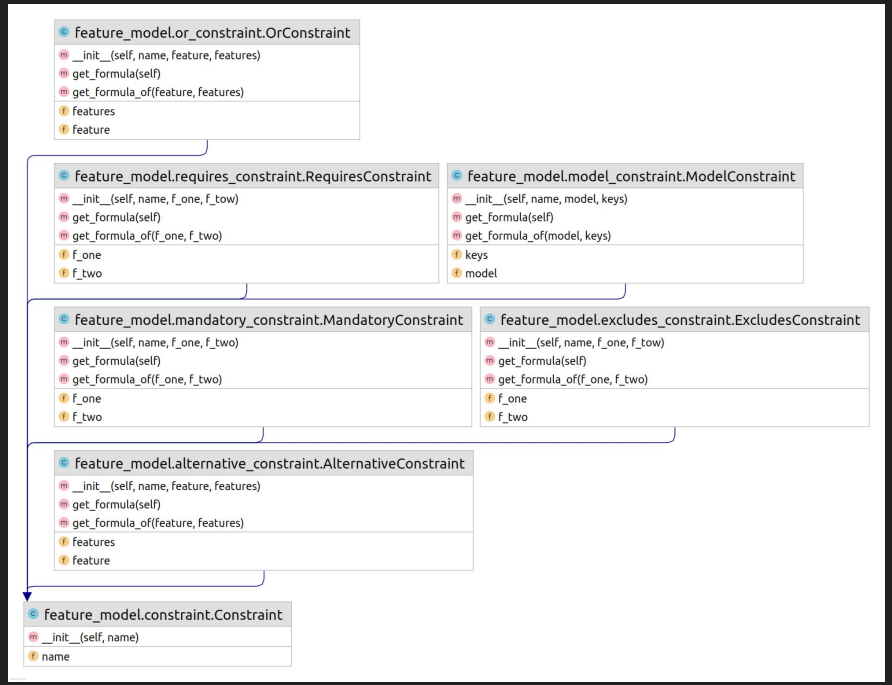
- Optional Sub-Features
- Mandatory Sub-Features
- Alternative Sub-Features (Exactly one allowed)
- Or-Sub-Features (Multiple allowed)
- Cross-Tree-Constraints: Requires and Excludes

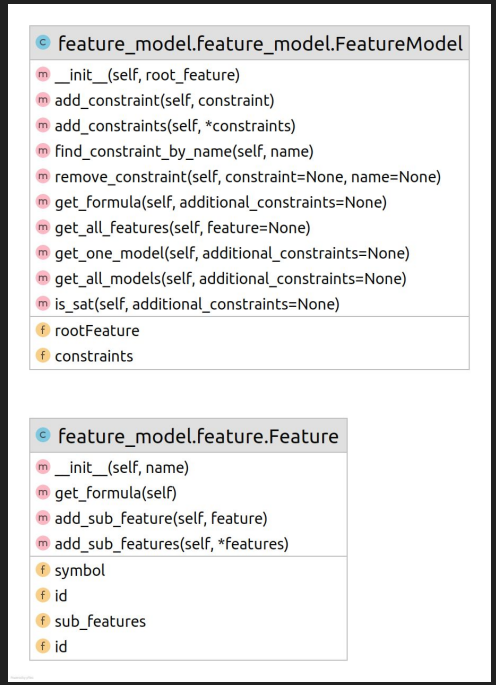
>> Constraints can be expressed as Formulas together with the Features

SMT:~/Applications$ # Feature Models

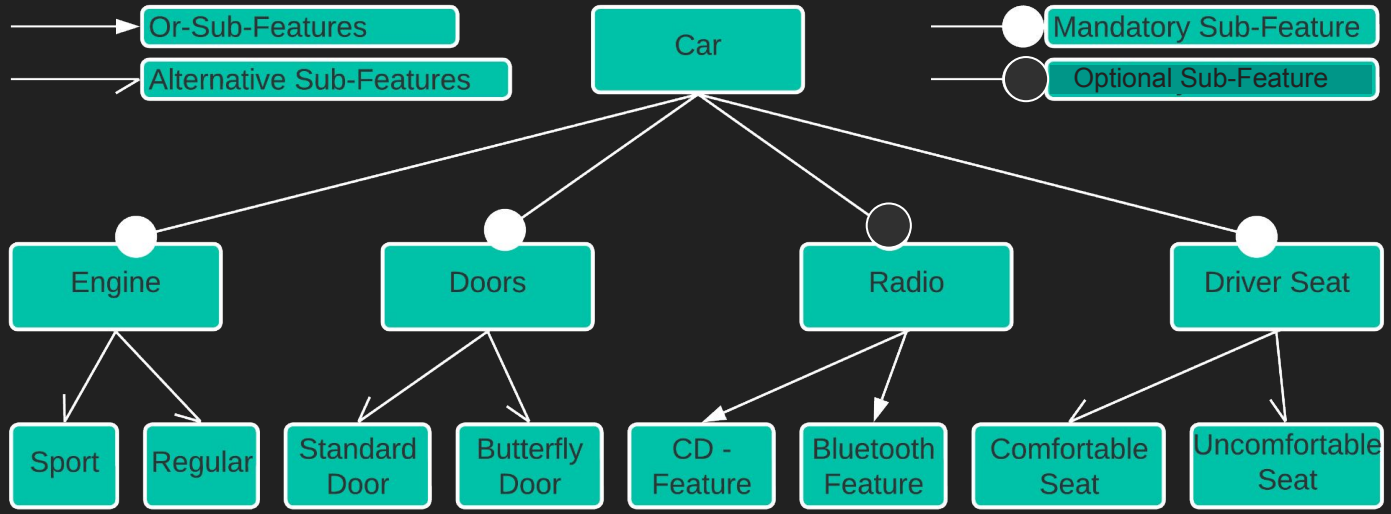
f_1 requires f_2	$f_1 \Rightarrow f_2$
f_1 excludes f_2	$\neg(f_1 \wedge f_2)$
f_s optional sub-feature of f	$f_s \Rightarrow f$
f_s mandatory sub-feature of f	$f_s \Leftrightarrow f$
$f_1,...,f_n$ alt.-sub-feat. of f	$(f_1 \vee ... \vee f_n) \Leftrightarrow f \wedge \bigwedge \neg(f_i \wedge f_j)$
$f_1,...,f_n$ or-sub-features of f	$(f_1 \vee ... \vee f_n) \Leftrightarrow f$

SMT:~/Applications$ Feature Models





SMT:~/Applications/Feature Models$ # Car Example




```
requires(Sport, Butterfly-Door) and
excludes(Comfortable Seat, Standard Door)
```