Scalability of Prolog: Real-world logic applications

Rishi Sharma

CS302: Paradigms of Programming

May 4, 2021

Another Puzzle Time

Can Prolog Survive This?

Remember the **Magic Square Puzzle**?

- Square size *nxn*.
- Fill distinct numbers from 1 to n^2.
- Sum of all rows, columns and diagonals should be equal.

Propositional Logic

- **Proposition**: A statement that can either be `True` or `False`.
- **Atomic Propositions**: Boolean Variables.
- **Propositional Logic**: Deals with Propositions.
- **Propositional Formulae**:

$$\phi := a \mid (\neg\phi) \mid (\phi \land \phi) \tag{1}$$

- **Example**: $(a \land b \land \neg c) \land (\neg a \land b)$
- **Precedence**: $\neg > \land > \lor > \rightarrow > \leftrightarrow$

Assignments

- **Assignment**: A function (α) that maps variables to `True` or `False`.
- **Question**: If there are V variables, how many Full Assignments are possible?
- **Question**: Given an assignment of variables in a Propositional Formula, you want to check if the Propositional Formula evaluates to `True`. What will be the complexity of this evaluation?

Finding the Model

For a Prop. Form. ϕ and set of all assignments **Assign**:

Satisfiability

Sat: $\exists \alpha \in Assign,\ Eval(\phi, \alpha) = True$.
Unsat: $\forall \alpha \in Assign,\ Eval(\phi, \alpha) = False$.

Validity

Valid: $\forall \alpha \in Assign,\ Eval(\phi, \alpha) = True$.

The α which satisfies ϕ, is known as the **model** for ϕ.

Finding the Model

- $a \lor \neg a$ Valid.
- $(m \land t) \rightarrow (m \lor t)$
  $\equiv \neg(m \land t) \lor (m \lor t)$
  $\equiv \neg m \lor \neg t \lor m \lor t$ Valid.
- $(r \land \neg s) \lor (r \land s)$ Sat.
- $a \land \neg a$ Unsat.
- $\neg((m \land t) \rightarrow (m \lor t))$ Unsat.

The Satisfiability Problem: Propositional Logic

- For a Prop. Form. ϕ and set of all assignments **Assign**, Find $\alpha \in Assign, Eval(\phi, \alpha) = True$.
- Decidable, but **NP Complete**.

```
Model SAT(phi){
    while(true) {
        if there are unassigned variables {
            choose an unassigned variable x;
            choose v from {true, false};
        } else {
            if phi is Satisfied, return SAT;
            else {
                if (!Backtrack()) return UNSAT;
            }
        }
    }
}
```

The Satisfiability Problem: Prolog

Recall the Hostel Allocation Puzzle from previous class.

```
rooms([room(_,5),room(_,4),room(_,3),room(_,2),room(_,1)]).
hostel(Rooms) :- rooms(Rooms),
    member(room(akash, A), Rooms), A \= 5,
    member(room(kairav, K), Rooms), K \= 1,
    member(room(milind, M), Rooms), M \= 1, M \= 5,
    member(room(piyush, P), Rooms),
    not(adjacent(M, P)), not(adjacent(M, K)),
    member(room(nites, N), Rooms), N > K,
    print_rooms(Rooms).
```

- **Observation**: Prolog is also solving the Satisfiability Problem.
- How is it different from Satisfiability on Propositional Logic?
- Range of the assignments is not limited to {`True`, `False`}.

Normal Forms

A **literal** is a boolean variable or its negation.
A **term** is a conjunction of literals.
Example: $(a \land b \land \neg c)$

Disjunctive Normal Form

Propositional Formula that is a disjunction of terms.
Example: $(a \land \neg b) \lor (\neg b \land \neg a \land c) \lor (a \land c)$.
What is the complexity of identifying:

- Satisfiability?
- Validity?
- Conversion of any formula to DNF?

The Normal Form for SAT Solvers

A **literal** is a boolean variable or its negation.
A **clause** is a disjunction of literals.
Example: $(\neg a \vee \neg b \vee c)$

Conjunctive Normal Form

Propositional Formula that is a conjunction of clauses.
Example: $(a \vee \neg b) \wedge (\neg b \vee \neg a \vee c) \wedge (a \vee c)$.
What is the complexity of identifying:

- Satisfiability?
- Validity?

Conversion of any formula to CNF can be done in linear time using **Tseitin's Encoding**.

Applications of Logic

- **Verification of Systems**:
  - A compiler optimizes a code; verify that the optimized code works the same as the original one.
  - An engineer comes up with a new circuit design of the processor; verify that it works as intended.
  - Assert that a particular erroneous state does not arise in a system.
- **Scheduling**: Can we schedule an additional train in a railway network without affecting other trains?
- **Solving Puzzles**: Magic Squares, Sudoku, Mastermind etc.

What makes SAT Solvers Fast?

Unit Propagation

- Remember they work on CNFs?
- Say a prop. form. has a clause: $(x1 \vee x2 \vee x3)$, and partial assignment $\alpha = \{x1 : False, x2 : False\}$.
- For the formula to be `SAT`, this clause must evaluate to `True`. Hence at this stage, the assignment $x3 = False$ can be skipped. Pruned search space.

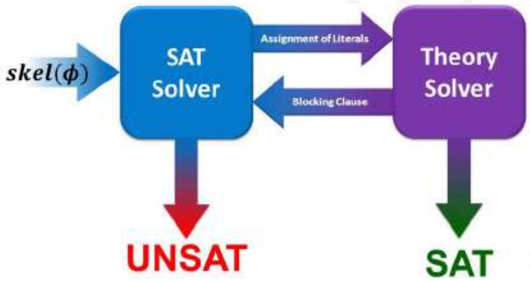
What makes SAT Solvers Faster?

Reason for Conflict: CDCL

- $\phi := (y\ l\ m) \wedge (x\ y\ \neg k) \wedge (k\ r\ x)$.
- $\alpha_1 = \{\neg y, \neg r, \neg l\}$.
- $\alpha_{1u} = \{\neg y, \neg r, \neg l, m\}$.
- $\alpha_2 = \{\neg y, \neg r, \neg l, m, \neg x\}$.
- $\alpha_{2u} = \{\neg y, \neg r, \neg l, m, \neg x, \neg k\}$.
- Conflict!
- **Reason for Conflict**: $\{\neg x, \neg y, \neg r\}$. First UIP is unique.
- $\phi_c := (y\ l\ m) \wedge (x\ y\ \neg k) \wedge (k\ r\ x) \wedge (x\ y\ r)$.
- Conflict Resolution: For some assignment α, SAT solvers can identify a much smaller partial assignment α_0, which would still conflict. Never try or extend that partial assignment, Pruned search space.

SMT Solvers

- Moving ahead of Propositional Logic. We want to do **arithmetic**, **equality**, **functions**, etc.
- Theory formula:
  $((a > 25) \vee (a + b = 5)) \wedge ((a < -5) \vee (b^2 = 16))$.
- Boolean Abstraction: $(p1 \vee p2) \wedge (p3 \vee p4)$.



Z3 Solver

Z3Py is a Python Wrapper for the Microsoft Z3 Theorem Prover.
A guide to Z3Py:
https://ericpony.github.io/z3py-tutorial/guide-examples.htm.

Thank you!