Automated Reasoning I

- Reasoning the ability to draw inferences or conclusions
- Formal logic defines a syntax and a calculus
- Logical reasoning avoids unjustified assumptions and confines inferences to that which are infallible
- Highlights necessary relationships *between* facts
Automated Reasoning II

- Automated theorem prover (ATP) for Modal Logic
- Direct applications to linguistics, system verification, agent-based systems and process analysis
- $\Gamma = \text{system assumptions}$
- $\varphi = \text{desired property}$
- Want to prove $\varphi$ is logical consequence of $\Gamma$
- Decidable with PSPACE-complete complexity
Research Motivation

- Growing size and complexity of designed systems
- Efficient APTs are required for verification
- Seek to build a scalable method for proving formulas in modal logic K
- Leverage the performance of an existing state-of-the-art SAT solver
Modal Logic I

- Extension of classical propositional logic

- Modal propositions broadly allow for necessity and possibility
  
  Example: Proposition P is not possible iff P is not necessary

- Non-truth functional, therefore decision problem non-trivial
Modal Logic II

- Atomic Formulae: $p ::= p_0 \mid p_1 \mid p_2 \mid \ldots$

- Formulae: $\varphi ::= p \mid \neg \varphi \mid \langle > \varphi \mid [\square] \varphi \mid \varphi \land \varphi \mid \varphi \lor \varphi \mid \varphi \rightarrow \varphi$

- Kripke Frame: directed graph $<W, R>$ where $W$ is a non-empty set of vertices and $R \subseteq W \times W$ is a binary relation over $W$

- Valuation: on a Kripke frame $<W, R>$ is a map $\vartheta : W \times \text{Atm} \rightarrow \{t, f\}$

- Kripke Model: $<W, R, \vartheta>$ where $\vartheta$ is a valuation on a Kripke frame $<W, R>$
Modal Logic III

Consequence relations:

<table>
<thead>
<tr>
<th>Forces</th>
<th>We Say</th>
<th>We Write</th>
<th>When</th>
<th>(\not\models \varphi)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in a world</td>
<td>w forces (\varphi)</td>
<td>(w \models \varphi)</td>
<td>(v(w, \varphi) = t)</td>
<td>(v(w, \varphi) = f)</td>
</tr>
<tr>
<td>in a model</td>
<td>M forces (\varphi)</td>
<td>(M \models \varphi)</td>
<td>(\forall w \in W. w \models \varphi)</td>
<td>(\exists w \in W. w \not\models \varphi)</td>
</tr>
</tbody>
</table>

- **Logical consequence:**

  \(\Gamma \models \varphi\) iff \(\forall M \in M \models \Gamma \Rightarrow M \models \varphi\)

- **Satisfiability:**

  K-satisfiable iff \(\exists M = \langle W, R, \upsilon \rangle \in K. w \in W. w \models \varphi\)

- **Want to show validity of \(\varphi\) - always satisfied in any \(M\).**
Tableau Calculi I

- Tableau calculus is a deductive system to show satisfiability, and thus logical consequence, in a modal logic (L)
- A set of rules, where each rule is interpreted as "if the numerator is L-satisfiable, then some denominator is L-satisfiable"
- Proof by refutation: show that $\sim \varphi$ is unsatisfiable, hence $\varphi$ is valid
Tableau Calculi II

- Tableau procedure is represented as a tree for a given formula set
- Each child branch obtained from parent nodes by instantiating a tableau rule:

  The tableau rules for modal logic K are as follows:
  X, Y, Z are possibly empty multi-sets of formulae
  Static rules:

  \[(\text{id}): \frac{p; \neg p; X}{\Box p; X}\]

  \[(\text{\&}): \frac{\varphi \land \psi; X}{\varphi; X \mid \psi; X}\]

  \[(\lor): \frac{\varphi \lor \psi; X}{\varphi; X \mid \psi; X}\]

  Transitional rule:

  \[(\Diamond K): \frac{\Diamond \varphi; \Box X; Z}{\varphi; X} \quad \forall \psi. \Box \psi \not\in Z \text{ and } \Box X = \{\psi \mid \psi \in X\}\]
Tableau Calculi III

- Branch in tableau is closed if leaf is an instance of (id) - contradiction
- Tableau is closed if all branches are closed
- An open branch shows a satisfying model
- Therefore, to show $\varphi$ is valid, must show the tableau for $\sim\varphi$ is closed
A SAT-Based Method I

- Implementation in Python
- Key insight: modal logic is an extension of propositional logic, therefore attempt to leverage performance of highly optimised SAT-solver (Z3 by Microsoft)
- Recursive depth first search
A SAT-Based Method II

- Pre-processing of input formula $\varphi$:
  1) Negate input formula
  2) Lexing and parsing
  3) Transform to negation normal form
  4) Simplification of formula
  5) Transform to modal clausal form (mcf)

- Per Gore and Nguyen (2009) mcf improve space bounds and decision procedures

- The negated formula in mcf is then passed to the top level prover
A SAT-Based Method III

- Saturation phase:

```python
...
M: Dictionary representation of modal clausal form per above.
Key: modal depth (i.e. length of modal context)
Value: [A, IB, ID, D]
X: Dictionary of active modal literals from previous world
V: Dictionary of valuations at each modal depth.
W: Modal depth (initially 0)
...

def saturation_function(M, X, V, w):
    if M[w] is Empty and X is Empty:
        return satisfiable
    else:
        A_w, IB_w, ID_w, D_w <- M[w]
        sat_constraints = set()
        sat_constraints <- add {X[w], A_w}

        valuation <- call_sat_solver(sat_constraints)
        if valuation is satisfiable:
            V[w] <- valuation
            return transition_function(M, X, V, w)
        else:
            return unsatisfiable
```
A SAT-Based Method IV

Transition phase:

```python
def transition_function(M, X, V, w):
    w1 = w + 1
    A_w, IB_w, ID_w, D_w <- M[w]
    
    ''' get_active returns set of active modalities based
    on whether each clause in ID_w or IB_w is (or can be)
    satisfied by V[w] '''
    active_box_literals <- get_active(IB_w, V[w])
    active_diamond_literals <- get_active(ID_w, V[w]), D_w
    
    if active_diamond_literals is None: return satisifiable
    
    for diamond in active_diamond_literals:
        X[w] <- add(diamond, active_box_literals)
        branch = saturation_function(M, X, V, w1)
        if branch == unsatisfiable:
            sat_constraints <- add {X[w], A_w}
            ''' get new valuation for w: OR-branch '''
            valuation = call_sat_solver(sat_constraints)
            if valuation is satisifiable:
                V[w] <- valuation
                return transition_function(M,X,V,w)
            else: return unsatisfiable
    
    return satisifiable
```

A SAT-Based Method V

Example:

- Let $\phi = \Box(p_0 \rightarrow p_1) \rightarrow (\Box p_0 \rightarrow \Box p_1)$
- $MCF(\neg \phi) = \{<>\neg b\}, [\{\neg a, b\}, \{a\}]
- Dictionary representation F:
  - key: 0 ; value: D: $\{\Diamond \neg b\}$
  - key: 1 ; value: A: $\{\{\neg a, b\}, \{a\}\}$

- Call prover (F)
  1. Modal depth = 0, saturation function returns valuation = $\{\}$
  2. Modal depth = 0, transition function returns $<>\neg b$ branch
  3. Modal depth = 1, saturation function returns unsat given (id) instance
  4. No other satisfying valuation at modal depth = 0, return unsat
  5. $\neg \phi$ is unsat, therefore $\phi$ is valid
A SAT-Based Method VI

- To find an open tableau, simply find first open tableau branch - i.e. $\phi$ is not valid
- To conclude tableau is closed, expand all branches and show that each leaf is an instance of (id) - i.e. $\phi$ is valid
- Complexity depends on branching
- Want to prioritise the expansion of branches with a higher likelihood of remaining open and quickly fail on branches with potential contradictions
- Difficult in application
Evaluation and results

- Benchmarks designed precisely to compare various methods
- Comparison to existing state-of-the-art methods:

<table>
<thead>
<tr>
<th>Subclass</th>
<th>SAT-based</th>
<th>BDDtab</th>
<th>InKreSAT</th>
<th>FaCT++</th>
</tr>
</thead>
<tbody>
<tr>
<td>branch_n</td>
<td>9</td>
<td>16</td>
<td>13</td>
<td>10</td>
</tr>
<tr>
<td>branch_p</td>
<td>16</td>
<td>21</td>
<td>16</td>
<td>9</td>
</tr>
<tr>
<td>d4_n</td>
<td>2</td>
<td>21</td>
<td>21</td>
<td>21</td>
</tr>
<tr>
<td>d4_p</td>
<td>3</td>
<td>21</td>
<td>21</td>
<td>21</td>
</tr>
<tr>
<td>dum_n</td>
<td>10</td>
<td>21</td>
<td>21</td>
<td>21</td>
</tr>
<tr>
<td>dum_p</td>
<td>0</td>
<td>21</td>
<td>21</td>
<td>21</td>
</tr>
<tr>
<td>grz_n</td>
<td>0</td>
<td>21</td>
<td>21</td>
<td>21</td>
</tr>
<tr>
<td>grz_p</td>
<td>0</td>
<td>21</td>
<td>21</td>
<td>21</td>
</tr>
<tr>
<td>lin_n</td>
<td>14</td>
<td>21</td>
<td>21</td>
<td>21</td>
</tr>
<tr>
<td>lin_p</td>
<td>1</td>
<td>21</td>
<td>21</td>
<td>21</td>
</tr>
<tr>
<td>path_n</td>
<td>9</td>
<td>21</td>
<td>10</td>
<td>21</td>
</tr>
<tr>
<td>path_p</td>
<td>12</td>
<td>21</td>
<td>10</td>
<td>21</td>
</tr>
<tr>
<td>ph_n</td>
<td>4</td>
<td>9</td>
<td>21</td>
<td>13</td>
</tr>
<tr>
<td>ph_p</td>
<td>3</td>
<td>9</td>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td>poly_n</td>
<td>21</td>
<td>21</td>
<td>21</td>
<td>21</td>
</tr>
<tr>
<td>poly_p</td>
<td>21</td>
<td>21</td>
<td>21</td>
<td>21</td>
</tr>
<tr>
<td>t4p_n</td>
<td>1</td>
<td>21</td>
<td>21</td>
<td>21</td>
</tr>
<tr>
<td>t4p_p</td>
<td>0</td>
<td>21</td>
<td>21</td>
<td>21</td>
</tr>
</tbody>
</table>
Conclusions and further work

- Achieved a sound and complete SAT-based theorem prover for modal logic K
- However performance of naïve prover is poor
- Further work:
  - Extend to multimodal logics of belief
  - Extend to modal logic S4
  - Re-implement in a lower level programming language