Proof sketch of R1

- R1: Given D, deciding whether Φ_D exists or not is polynomial.
- A polynomial algorithm:
  - For each pair of effect axioms of the form
    a **causes** f **if** $p_1, \ldots, p_m$ and
    a **causes** $\neg f$ **if** $q_1, \ldots, q_r$
    we have to make sure that there does not exist a state where both $p_1, \ldots, p_m$ and $q_1, \ldots, q_r$ hold.
    This is the case if there exists a fluent literal f such that $f \in \{p_1, \ldots, p_m\}$ and $g \in \{q_1, \ldots, q_r\}$, where g is the complement of f.
    i.e., $\{p_1, \ldots, p_m\} \cap \{\bar{q}_1, \ldots, \bar{q}_r\} \neq \emptyset$, where $\bar{\neg f} = f$ and $\bar{f} = \neg f$, for any fluent f.
  - If D has n effect propositions at the worst case we have to check $O(n^2)$ pairs of effect propositions.

Proof sketch of R2

- R2: Given a consistent D and a set of fluents σ_0, deciding whether σ_0 is an initial state with respect to (D, O), is polynomial.
- A polynomial algorithm:
  - Lemma 1 (L1): Given a consistent D, Computing $\Phi_D(a, \sigma)$ is polynomial time.
    - Recall that $\Phi_D(a, \sigma) = \sigma \cup E_D^+(a, \sigma) \setminus E_D^-(a, \sigma)$
    - Computing $E_D^+(a, \sigma)$ means inspecting each rule in D once. Similarly for $E_D^-(a, \sigma)$.
  - To evaluate if $M = (\sigma_0, \Phi_D)$ is a model of (D, O) we need to consider each observation of the form f **after** $a_1, \ldots a_m$ and check if f holds with respect to $\Phi_D(a_m, \Phi_D(a_{m-1}, \ldots \Phi_D(a_1, \sigma_0) \ldots))$.
  - Note: we don't need to explicitly compute Φ_D for all pairs of actions and states (that would be exponential) but just for the cases we need in the previous step.

A quick review of NP-completeness

- P: The class of problems that can be solved in polynomial time.
- NP: The class of problems that can be solved in *non-deterministic* polynomial time. (Can be solved by a non-deterministic Turing machine in polynomial time.)
  - P $\subseteq$ NP. (All problems in P are also in NP.)
  - A problem is solvable in non-deterministic polynomial time if we can guess a solution and verify it in (deterministic) polynomial time. (For decision problems we guess an instance of the solution that supports the decision.)

    i.e., If some one gives us a potential solution we have a program (C, C++, Java, LISP etc.) that runs in polynomial time and that can tell if the input (potential solution) is indeed a solution or not.
  - Finding k-colors ($k > 2$) that color a graph so that no vertices connected by an edge have the same color is in NP.

    Because if someone gives us a color assignment (potential solution) with k-colors by inspecting each edge of the graph in $O(|E|)$ times we can tell if the input is indeed

- Propositional satisfiability is in NP. (3SAT is in NP).
  $(f_{11} \vee f_{12} \vee f_{13}) \wedge \ldots \wedge (f_{n1} \vee f_{n2} \vee f_{n3})$

- coNP – A problem is in coNP if the complement of the problem is in NP.
  - Unstatisfiability of propositional formulas is in coNP.

- NP Complete: A problem is said to be (in the class) NP complete if
  1. it is in NP, and
  2. all problems in NP can be reduced to that problem in polynomial time.

- Propositional statisfiability (SAT) is showed to be NP complete by encoding computations in a non-deterministic Turing machine as a propositional formula.

- 3SAT is shown to be NP-Complete by (i) showing 3SAT is in NP; and (ii) polynomially reducing propositional statisfiability to 3SAT.
  - Polynomial reduction: Given a propositional formula F we can create a 3SAT formula F' in polynomial time such that F is satisfiable *iff* F' is satisfiable.
  - Since all NP problems can be polynomially reduced to SAT, and SAT can be polynomially reduced to 3SAT it means all NP problems can be polynomially reduced to 3SAT.

- In general to show a problem is NP Complete we need to
  1. Show that the problem is in NP, and
  2. Show that a known NP complete problem (such as SAT or 3SAT) is polynomially reducable to this problem.