1. Theoretical Foundations

1.1 Introduction

Goal

learn about:
- different levels of correctness guarantees
- unsolvability certificates for classical planning
- how to make your planner certifying
Target Audience

- familiar with classical planning
- optional: planner developer

Certifying Algorithms

Motivation

- ISP wants to build antenna towers for 5G.
- antenna supplier:
  - “You need at least \( x \) towers”
  - shows calculations with a tool, if using less than \( x \) towers tool says “unsolvable”
- ISP’s options:
  - blindly trust the tool
  - demand some form of correctness guarantee for their tool
Levels of Verification

How can we verify the correctness of an algorithm?

- **Theoretical**: correctness proof in papers
- **Implementation**:
  - Verification method
    - Unit test
    - Certifying algorithms
    - Theorem provers
  - Verified inputs
    - Some designated inputs
    - Each input when it occurs
    - All possible inputs

▶ Unit tests are easy to do, but it is also easy to miss bugs.
▶ Theorem provers are very expensive but offer highest guarantee (tiny core which is checked very carefully).
▶ Certifying algorithms strike a balance between trust and effort.

Certifying Algorithms [McConnell et al. 2011]

“A certifying algorithm is an algorithm that produces, with each output, a certificate or witness (easy-to-verify proof) that the particular output has not been compromised by a bug.”

→ Algorithm might still contain bug, but current output is correct

**Example**

Is CNF formula $\varphi$ satisfiable?

- Yes $\rightarrow$ provide satisfying assignment $I$
- No $\rightarrow$ provide UNSAT certificate (resolution, DRAT proof...)

Certifying Algorithms in Planning

Guiding Properties

**Soundness & Completeness**
We can create a certificate for task $\Pi$ iff $\Pi$ is unsolvable.

**Efficient Generation**
Certificate creation incurs only polynomial overhead to the planner.

**Efficient Verification**
Certificate verification is at most polynomial in its size.

**Generality**
A wide variety of planning techniques can produce a certificate.
Quis custodiet ipsos custodes?

Certificate must be verified by a verifier. But what if the verifier has bugs?

► Verify the verifier!
► Verify the verifier-verifier!
► ...?
► At some point we need to trust something.

Good news: Verifiers are often simpler than the original algorithm.
→ Verify the verifier with theorem provers.

Example: A Formally Verified Validator for Classical Planning Problems and Solutions [Abdulaziz & Lammich, 2018]

Proof Systems

Natural Deduction

► Proof systems are built on axioms and inference rules.
  ► axioms: tautology ($A \lor \neg A$)
  ► inference rules: conclusion based on premises (if $A \land B$ then $A$)
► Hilbert-style systems try to express as much as possible in axioms.
► Natural deduction in contrast focuses on inference rules.
  ► first proposed by Gerhard Genzen [Genzen 1935]
  ► should more closely reflect our natural way of reasoning

The proof system presented here uses the natural deduction style.

Inference Rules

Inference rule
An inference rule $\Gamma$ takes premises $A_1, \ldots, A_n$ and concludes $B$: 

\[
\frac{A_1 \quad \ldots \quad A_n}{B} \quad \Gamma
\]

► Rules use placeholder variables and are universally true for all instantiations.
► The correctness of rules can be shown in two ways:
  ► within the proof system using existing rules, or
  ► outside of the proof system.
→ Once proven correct, we can use rules purely syntactically.
► Axioms are rules with no premises.
► Rules can also use and discard assumptions: For example, if under assumption $A$ we can prove $B$, we have shown $A \rightarrow B$. 
Example Proof System

Example
Inference Rules:

\[
\begin{array}{c}
* \quad \square \quad A \\
\diamond \quad B \\
\diamond \quad C \\
\square \quad * \quad \diamond \quad D
\end{array}
\]

Show \(\circ\):

<table>
<thead>
<tr>
<th>#</th>
<th>statement</th>
<th>justification</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>(\diamond)</td>
<td>from (B)</td>
</tr>
<tr>
<td>(2)</td>
<td>(*)</td>
<td>from (C) with (1)</td>
</tr>
<tr>
<td>(3)</td>
<td>(\square)</td>
<td>from (A) with (2) and (1)</td>
</tr>
<tr>
<td>(4)</td>
<td>(\circ)</td>
<td>from (D) with (3), (2) and (1)</td>
</tr>
</tbody>
</table>

Unsolvability Proof System

First Certificate Attempt

How can we show that a planning task is unsolvable?

Inductive Certificate [E et al 2017]

An inductive certificate for a STRIPS planning task

\[\Pi = \langle V, A, I, G \rangle\]

is a set of states \(S\) with the following properties:

\[\begin{align*}
&\text{\(I \in S\)} \\
&\text{\(S\) contains no goal state} \\
&\text{\(S[A] \subseteq S\), where \(S[A] = \{s' \mid s[a] = s'\} \text{ for some } a \in A\)}
\end{align*}\]
How Planners Show Unsolvability

- Uninteresting search space areas get pruned incrementally
- Later pruning steps can use knowledge from previous ones.
- Distilling these steps into a singular argument is difficult.

→ Proof systems can capture this type of incremental reasoning.

Proof System Objects

- state sets $S$ represented as
  - BDD
  - (dual)-Horn formula
  - 2CNF formula
  - explicit enumeration
  - ...
- action sets $A$ represented as ID enumeration

Types of Knowledge

Dead State
A state $s$ is dead if no plan traverses $s$, i.e. there is no plan $\pi = \langle a_1, \ldots, a_n \rangle$ and $1 \leq i \leq n$ with $s = I[a_1] \ldots [a_i]$.

→ captures idea of pruned states (in both directions)

statements in the proof system:
- $S$ dead (all $s \in S$ dead)
- $E \subseteq E'$ (where $E$ and $E'$ are sets of states or actions)
- unsolvable
Inference Rules - Showing Deadness

Empty set Dead
\[ \emptyset \text{ dead} \]

Union Dead
\[ S \text{ dead } S' \text{ dead} \]
\[ S \cup S' \text{ dead} \]

Subset Dead
\[ S' \text{ dead} \]
\[ S \subseteq S' \text{ dead} \]

Inference Rules - Showing Unsolvability

Conclusion Initial
\[ \{I\} \text{ dead} \]
\[ \text{unsolvable} \]

Conclusion Goal
\[ S_G \text{ dead} \]
\[ \text{unsolvable} \]

Inference Rules - Set Theory

Union Left
\[ E \subseteq (E' \cup E) \]

Intersection Right
\[ (E \cap E') \subseteq E \]

Subset Union
\[ E \subseteq E'' \]
\[ E' \subseteq E'' \]
\[ (E \cup E') \subseteq E'' \]

Subset Intersection
\[ E \subseteq E' \]
\[ E \subseteq E'' \]
\[ E \subseteq (E' \cap E'') \]

Subset Transitivity
\[ E \subseteq E' \]
\[ E' \subseteq E'' \]
\[ E \subseteq E'' \]
Inference Rules - Progression and Regression

**Action Union**

\[
S[A] \subseteq S' \quad S[A'] \subseteq S' \quad \frac{}{S[A \cup A'] \subseteq S'}
\]

**Progression Transitivity**

\[
S[A] \subseteq S'' \quad S'[A] \subseteq S'' \quad \frac{}{S'[A] \subseteq S''}
\]

**Progression to Regression**

\[
S[A] \subseteq S' \quad \frac{}{[A]S'' \subseteq S''}
\]

... 

Is this enough?

How can we show \( S[A] \subseteq S \) or similar statements?

▶ depends on planning task and contents of \( S \)

▶ requires semantic analysis

▶ set theory rules only syntactical

→ new source of information: basic statements

Soundness and Completeness

Given a STRIPS planning task \( \Pi = (V, A, I, G) \), there is a proof in the proof system for \( \Pi \) if \( \Pi \) is unsolvable.

**Proof**

Soundness: This follows from the correctness of the inference rules and basic statements.

Completeness: Consider \( \mathcal{R}^\Pi \), the set of states reachable from \( I \).

<table>
<thead>
<tr>
<th>#</th>
<th>statement</th>
<th>justification</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>( \emptyset ) dead</td>
<td>ED</td>
</tr>
<tr>
<td>(2)</td>
<td>( \mathcal{R}^\Pi[A] \subseteq \mathcal{R}^\Pi \cup \emptyset )</td>
<td>B2</td>
</tr>
<tr>
<td>(3)</td>
<td>( \mathcal{R}^\Pi \cap S_G \subseteq \emptyset )</td>
<td>B1</td>
</tr>
<tr>
<td>(4)</td>
<td>( \mathcal{R}^\Pi \cap S_G ) dead</td>
<td>SD with (1) and (3)</td>
</tr>
<tr>
<td>(5)</td>
<td>( \mathcal{R}^\Pi ) dead</td>
<td>PG with (2), (1) and (4)</td>
</tr>
<tr>
<td>(6)</td>
<td>( {I} \subseteq \mathcal{R}^\Pi )</td>
<td>B1</td>
</tr>
<tr>
<td>(7)</td>
<td>( {I} ) dead</td>
<td>SD with (5) and (6)</td>
</tr>
<tr>
<td>(8)</td>
<td>unsolvable</td>
<td>CI with (7)</td>
</tr>
</tbody>
</table>

Basic Statements

\( B1 \) \( \cap L_R \subseteq \cup L_R' \)

\( B2 \) \( (\cap X_R[A]) \cap \cap L_R \subseteq \cup L_R' \)

\( B3 \) \( [A](\cap X_R) \cap \cap L_R \subseteq \cup L_R' \)

\( B4 \) \( L_R \subseteq L_R' \)

\( B5 \) \( A \subseteq A' \)

▶ In B1-B3 all sets must be represented by the same formalism.

▶ Unions and intersections are bounded.

▶ We only support pro-/regression for set variables.

▶ B4 enables us to mix formalisms.
1. Theoretical Foundations

Efficient Verification

Verifying Statements

The verifier needs to verify each step of the proof.

- inference rules
  - universally true
  - check if rule is applied correctly (syntax)
    → easy to verify
- basic statements
  - sets must be interpreted (semantic)
    → depends on set representation formalism

Formalisms & Operations

How can we analyze whether basic statements can be verified efficiently?

- check each formalism separately
- check what operations a formalism \( R \) must support
  - **SE** (sentential entailment): Given \( R \)-formulas \( \varphi \) and \( \psi \), test whether \( \varphi \models \psi \).
  - **\( \land \)BC** (bounded conjunction): Given \( R \)-formulas \( \varphi \) and \( \psi \), construct an \( R \)-formula representing \( \varphi \land \psi \).
  - **toCNF** (transform to CNF): Given \( R \)-formula \( \varphi \), construct a CNF formula that is equivalent to \( \varphi \).
  - ... [Darwiche & Marquis 2002]

Efficient Verification of \( B1 \)

To verify \( \bigcap X_R \subseteq \bigcup X_R \) efficiently \( R \) must efficiently support:

| \( |\bigcup X'_R| = 0 \) | \( |\bigcap X_R| = 0 \) | \( |\bigcap X_R| = 1 \) | \( |\bigcap X_R| > 1 \) |
|-----------------|-----------------|-----------------|-----------------|
| \( |\bigcup X'_R| = 1 \) | CO | CO, \( \land \)BC toDNF |
| \( |\bigcup X'_R| = 1 \) | VA | SE |
| \( |\bigcup X'_R| > 1 \) | VA, \( \lor \)BC toCNF | SE, \( \lor \)BC toCNF, CE |

- multiple rows indicate different possible options
- for \( B1 \): move negated literals to the “correct” side
**Efficient Verification of B1 - Example**

| $|\bigcap X_R|$ | $|\bigcup X_R|$ | $|\bigcap X_R|$ | $|\bigcup X_R|$ |
|-----------------|-----------------|-----------------|-----------------|
| 0               | CO              | CO, $\land$ BC  | toDNF           |
| 1               | VA, $\lor$ BC   | SE, $\land$ BC  | IM              |
| > 1             | VA, toCNF      | SE, $\land$ BC  | toDNF, IM, $\lor$ BC |
|                 | toCNF, CE      | SE, $\land$ BC, $\lor$ BC | toDNF, IM, $\lor$ BC |

**Example**

For sets $S_1$ to $S_5$ (all represented with $R$), the statement $S_1 \cap S_2 \subseteq S_3 \cup S_4 \cup S_5$ can be verified efficiently iff $R$ supports

- **SE** (sentential entailment) and **$\lor$ BC** (bounded disjunction), or
- **toCNF** (transform to CNF) and **CE** (clausal entailment).

**Concrete Formalisms**

What do BDDs, (dual-)Horn formulas, 2CNF formulas and explicit enumeration support?

- Basic statements **B1-B3** are fully supported by all formalisms.
- Basic statement **B4** between those formalisms is supported in most cases with the following exceptions:
  - $\varphi_R \models \neg \psi'_R$ where $R$ and $R'$ are a combination of BDD, (dual-)Horn and 2CNF
  - $\varphi_{(\text{dual-Horn/2CNF)}} \models \psi_{\text{BDD}}$