Chapter 6

Experiments in mutilated problem solving

6.1 Introduction and Motivation

Theorem proving, constraint solving and machine learning provide powerful techniques for solving AI problems. In all these approaches, background knowledge needs to be provided, from which the system will infer new knowledge. Often, however, the background information may be obscure or incomplete, and is usually presented in a form suitable for only one type of problem solver, such as a first order theorem prover. In real world scenarios, there may not be enough background information for any single solver to solve the problem, and we are interested in cases where it may be possible to combine a machine learner, theorem prover and constraint solver in order to best use their incomplete background knowledge to solve the problem.

This chapter describes our experiments designed to test the feasibility of the combined reasoning approach to a whodunnit problem. We demonstrate the Aunt Agatha problem and discuss about how we used machine learning, theorem proving and constraint solving in order to solve it. We then introduce a way to mutilate the problem and present the results of solving techniques given the mutilated Aunt Agatha. We then employ HR [21] to recover the missing axiom. The steps which have been carried out to built an automated combined reasoning system is presented. Such system will have the ability of automatically translating the problem into the syntax of solving techniques being used.

6.2 The Aunt Agatha Problem

We concentrate on the scenario of a police investigation of a murder case. In such a scenario, in addition to axiomatic background knowledge, there may be previous solved cases which bear resemblance to the current case. Given that the previous cases were solved, one can imagine employing a machine learning system to learn a set of rules which can classify suspects in a case as either guilty or not guilty. The rule set could then be applied to the current case. If only one person was classified as guilty, this would solve the problem. While this reasoning may not be sound, it would at least help to identify a prime suspect. In addition, in the current case, there may be information describing the particulars of the case, arising from physical evidence, motives, alibis, general knowledge, etc. If so, it may be possible to define a set of constraints that the guilty suspect must
satisfy, and then use a constraint solver to rule out suspects. If only one suspect satisfies all the constraints, again the problem is solved. Alternatively, the same information about the case may be used as axioms in a theorem proving setting. In such a setting, one could attempt to prove a set of conjectures, each one stating that a particular suspect is guilty/not guilty. If only one suspect is proved to be guilty (or alternatively, it is possible to prove that all but one suspects are not guilty), then the problem is once again solved [22].

To show the feasibility of using three different type of solvers to attack the same problem, we looked at the “Who Killed Aunt Agatha” problem from the TPTP library PUZ001 [23]. The background knowledge for this problem is stated in English as follows: “Someone who lives in Dreadbury Mansion killed Aunt Agatha. Agatha, Butler and Charles live in Dreadbury Mansion and are the only people who live therein. A killer always hates the victim and is never richer than the victim. Charles hates no one that Aunt Agatha hates. Agatha hates everyone except the Butler. The Butler hates everyone not richer than Agatha. The Butler hates everyone Aunt Agatha hates. No one hates everyone and Agatha is not the butler”.

This problem is usually posed as a logic puzzle for theorem provers, where the aim is to prove that Aunt Agatha killed herself. However, in a more general setting, like real world scenarios, the answer would not be given, therefore, we would be asked to find out who killed Aunt Agatha. With this tweak, we can make it amenable to the three different solving approaches as described above.

6.2.1 Machine Learning Experiments

To show that, in principle, such problems are amenable to a machine learning approach, we initially invented some data which embodies the axioms of the problem. In particular we invented the data for five case studies where each case contains three people, one of whom had been murdered. We specified who was richer than who, who hated who, who was killed and in all cases, we specified the killer and the victim as the same according to the problem, therefore, the invented cases had two characteristics: there was a direct mapping from Agatha, Charles and Butler to one of the people in the case study. Importantly, all axioms from the problem statement were satisfied by the cases. In the first instance, the data reflected the fact that the murderer and the victim were always the same person: the Agatha character.

The invented data was produced in the syntax of the Progol machine learning system [19]. Based on the machine learning methodology explained in 4.2 and 4.3.1, we needed to describe the relations (predicates) between objects of given types which can be used in the head or the body of hypothesized clauses. Note that hypothesized clauses are the clauses that the system is going to learn. In this case, we want Progol to learn the concept of “killer”, hence the head of the clause should imply the “killer”. Negative examples were included as non-killers. Parts of the input for one case is shown below and the complete code will be in the appendix:

```prolog
:- modeh(1, killer(+person))?
:- modeb(1, victim(+person))?
:- modeb(1, hates(+person, #person))?
:- modeb(1, richer(+person, #person))?
```
% Case 1
% People involved
  person(a).
  person(b).
  person(c).
% Relations between people
  hates(a, a). richer(a, b).
  hates(a, c). richer(c, b).
  hates(b, a). richer(a, c).
  hates(b, c).
  :- hates(c, a). :- hates(c, b).
% Victim
  victim(a).
% Positive example
  killer(a).
% Negative examples
  :- killer(b).
  :- killer(c).

Our modeh declaration states that the head atom has predicate symbol killer and has 1 variable of type person as argument. The '+' sign indicates that the argument is an input variable. In modeh and modeb declarations, number 1 is the recall which is used to bound the number of alternative solutions for instantiating the atom. For instance, killer predicate should give a unique answer when given an input argument (+person).

modeb declarations notify the atoms that can be used to the body of the hypotheses and '#' sign indicates that a constant should be placed at that position.

We ran the Progol using all five cases and it hypothesized the rule that:

\[
\text{killed}(A) : \neg \text{victim}(A).
\]

The system correctly hypothesized that the killer is the same as the victim. We also changed modeh declaration in order to make the predicate symbol killer with 2 variables of type person as arguments:

\[
:- \text{modeh}(1, \text{killed}(+\text{person}, -\text{person}))?
\]

The '-' sign represents the output variable. The declaration allows killed to be used in the head to find only one output variable for a given input. Respectively, the syntax of positive and negative examples for killer should also be changed, for instance, killed(person1, person1). In this case Progol hypothesized the rule:

\[
\text{killed}(A, A).
\]

To make matters more interesting, in the second instance, we generated the data to still satisfy the axioms, but we varied the murderer/victim combination. In the other words, the victim and the murderer are different people while holding the relations of the murdered and the victim. In this instance, Progol hypothesised the following rule:

\[
\text{killed}(A, B) : \neg \text{hates}(A, B), \setminus + \text{richer}(A, B).
\]
We can re-write the above Progol syntax in FOL as:

\[
killed(A, B) \leftarrow \text{hates}(A, B), \neg \text{richer}(A, B).
\]

For both instances, when we applied the generated output to the current case, only Aunt Agatha fitted the problem.

**Progol being used for Axiom Formation:**

Based on the above results, we can conclude that Progol is able to find the underlying axioms from an automated theorem proving problem. This sparked an idea that, it is possible to use Progol to form the axioms that are not explicit in an ATP problem. In a mutilated problem which we will discuss in this chapter, the main problem is to recover the missing axioms. Progol can be used for such tasks. To do so, we can initially employ an artificial data generator to produce different sets of examples based on the problem axioms and the data will be then imported to the Progol. This idea will be considered as future tasks.

### 6.2.2 Constraint Solving Experiments

To show that such problems are amenable to a constraint solving approach, we wrote a constraint satisfaction problem (CSP) in the syntax of the Sicstus Prolog CLPFD module [24]. We defined the problem as a CSP with one variable which can take one of three values representing Agatha, Butler and Charles. Therefore, the domain of the variable comprises these three people. The domain can be referred to as *suspects*. As for the constraints, two of the problem axioms, *killer always hates the victim* and *killer is never richer than the victim* were instantiated. The rest of the axioms were regarded as background information. According to the problem, only one person can be the killer, therefore, in principle, only one of the suspects can satisfy the constraints for being the killer and the rest of the suspects should be ruled out. Parts of the code written in Sicstus CLPFD syntax is shown below: (the entire code is appended)

```
pseudonym(1, 'AuntAgatha').
pseudonym(2, 'Butler').
pseudonym(3, 'Charles').

'pseudonym' makes '1', '2' and '3' adopting one of the suspects' names. This is required as CLPFD only allows integer domains.

killed_agatha(X): -
    domain([X], 1, 3),
    killed(X, 1),
    labeling([], [X]).

Note that we post a constraint by killed(X, 1), forcing the X to be chosen from the suspects who satisfy this constraint.

killed(X,Y): -
    hates(X,Y),
```
Following statements depict the output of the CSP solver:

- Aunt Agatha has not been ruled out.
- Charles can not be the killer.
- Butler can not be the killer.

In a CSP problem, a variable that satisfies all the constraints is not necessarily the correct answer. We can only decide confidently about the variables that do not satisfy the constraints. For the Aunt Agatha problem, Butler and Charles can not satisfy the constraints, therefore, they can not be the killer in any case and should be rule out and excluded from being a killer. However, Agatha satisfying the constraints for being a killer, can not be ruled out as a “non-killer”.

### 6.2.3 Aunt Agatha - Theorem Proving

Finally, we defined the problem statement in Otter theorem prover[14] syntax, parts of the code is shown below: (the entire code is appended)

\[
\begin{align*}
&\forall x (\neg (\text{richer}(x, \text{agatha})) \implies \text{hates}(\text{butler}, x)). \\
&\forall x (\text{hates}(\text{agatha}, x) \implies \text{hates}(\text{butler}, x)). \\
&\forall x (\text{hates}(\text{agatha}, x) \implies \neg (\text{hates}(\text{charles}, x))). \\
\end{align*}
\]

We specified six conjectures. The conjectures were respectively: Agatha killed/didn’t kill Agatha; Butler killed/didn’t kill Agatha; Charles killed/didn’t kill Agatha. For instance, the conjecture Agatha killed Agatha is as follow:

\[-\text{killed}(\text{agatha}, \text{agatha}).\]

Note that in Otter syntax, we need to negate the statement we want to prove, because Otter uses proof by refutation which means the negated theorem statement is used as the problem premise along with the problem axioms.

Otter successfully proved that Agatha killed Agatha and that Butler and Charles didn’t kill Agatha. It failed to prove any other conjectures. This shows that such who-dunnit problems are amenable to solution by theorem provers.

### 6.3 Mutilated Aunt Agatha problem

As we mentioned in 6.1, in real world scenarios, there may not be enough background information for any single solver to solve the problem and we are interested in cases where it may be possible to combine a machine learner, theorem prover and constraint solver in order to best use their incomplete background knowledge to solve the problem. In order to make our analysis closer to real world problems, we mutilated the Aunt Agatha problem by removing parts of its information from each of the three problem statements in such a way that neither Progol, Sicstus nor Otter could solve the problem. Subsequently.

We investigated methods for combining the mentioned reasoning systems in such a way that the solution can still be found. We plan to investigate many different ways to mutilate the problem, yet still solve it via a combination of systems. We investigated
an opportunity for combining different reasoning systems. In particular, from the theorem proving and CSP problems, we removed the axiom that “no-one hates everyone”, which can be written in FOL as below:

$$\forall X(person(X) \rightarrow \exists Y(person(Y) \land \neg hates(X,Y))$$.

or

$$\neg (\exists X(person(X) \rightarrow \forall Y(person(Y) \land hates(X,Y)))$$.

This is crucial to solving the problem, because without it, Sicstus cannot rule out Butler as the killer, and Otter can similarly prove that both Butler and Agatha killed Agatha. We investigated whether the data from the machine learning approach could be used to recover the missing axioms. In particular, we employed the HR automated theory formation system [21] to form a theory about the previous case studies. In the other words, we input the previous case studies to HR. Using HR’s forall, exists, negate and compose production rules, HR made the conjecture that in all case studies:

$$\forall s.t. person(x) \land (\forall y, (person(y) \rightarrow hates(x,y)))$$.

This states that, in all cases, there is no person who hates everyone. Hence we see that HR has recovered the missing axiom, which could be used by the constraint solver or prover to solve the problem.

### 6.4 Translation Suite

We are building a system which is able to take a general problem statement, such as a whodunnit problem and translate it to the syntax of various solvers such as Sicstus, Otter and Progol. Moreover, in situations where none of the solvers are initially successful, the system will be able to take the partial solutions from each solver and see whether these can be used together to fully solve the problem. To employ AI techniques to simulate a combined reasoning approach, its essential to translate the problem to syntaxes of the AI systems involved. To do so, it is required to implement a translation suite which can automatically translate the problem statements, given in FOL or any language close to FOL, to the required syntaxes. For our case, such translation suite should take problem specifications expressed in Otter syntax and turn them into input for the Sicstus CLPFD and Progol. We chose to express the problem in Otter syntax because of two reasons: firstly, Otter syntax is very similar to FOL and secondly, it will reduce the work of translating to Otter itself in case the problem was stated in a different language. In addition, we translate the Otter syntax into Sicstus CLPFD and Progol using Prolog, therefore, Prolog is the medium of translation. The translation suite is still not complete and requires more work.

#### 6.4.1 Whodunnit Problems - Background Knowledge

Considering that we have focused on whodunnit problems, we initially analyze the structure of the problem which should be translated, bearing in mind that the main language for expressing the problem is Otter. The problem statement comprises:
• Axioms about the problem with no variables involved like: *Agatha hates Butler*

• Constraints; all other axioms are treated as constraints, like: *killer is never richer than the victim*

• Past cases along with the current case, say:

\[
\begin{align*}
\text{hates}(a, a) &. \text{ richer}(a, b). \\
: \neg \text{hates}(a, c) &. \neg \text{richer}(c, b).
\end{align*}
\]

• Question which should explicitly mention:
  
  – Suspects \((S_1, S_2, \ldots, S_n)\)
  
  – Victim(s)
  
  – The question predicate, say \(killed_agatha(X)\).

This information should be preprocessed and fed to Otter, Sicstus CLPFD and Progol.

### 6.4.2 Algorithm - Expressing Whodunnit Problems for Automated Theorem Proving

Preprocessing for Otter is to discard the past cases and to keep the rest of the information (Axioms and constraints). In addition, the question containing the suspects and victim should be also fed to Otter. As the problem is stated in Otter, there will be no translation involved. The algorithm is as follows:

• Keep the axioms and constraints, discard the past cases

• For \(\langle \text{all suspects}(S_1, S_2, \ldots, S_n) \rangle\) Do:
  
  – Attempt to prove that killer is \(S_x\): \(\text{killer}(S_x)\).
    
    if it fails, prove that \(S_x\) is not the killer: \(\neg \text{killer}(S_x)\).

• If \(\exists ! S_x \text{ s.t } \text{killer}(S_x)\) is proved, output \(S_x\) as definite killer

• Else
  
  – if \(\exists ! S_x \text{ s.t } \neg \text{killer}(S_x)\) is not proved, then output \(S_x\) as the killer.
  
  – else output all \(S_x\) that \(\text{killer}(S_x)\) is proved or \(\neg \text{killer}(S_x)\) is not proved as : “\(S_x\) can be the killer”, therefore, narrowing the search space

### 6.4.3 Algorithm: Expressing Whodunnit Problems as CSP - Automatic Generation of Constraints

This section describes the preprocessing and translation from Otter to Sicstus CLPFD.

• keep the axioms and constraints, discard the past cases.

• specify the variable(s) in the “question predicate” as the constraint variable

• suspects defined in the problem should be instantiated as the variable Domain
• extract the axioms with no propositional connectives and quantifiers and import directly to Sicstus
  – in case of such axioms having only negation symbol “−”, rewrite it with “: −” and import it to Sicstus.

• extract the problem question and add to as the head of the main constraint clause, for instance: killed\_agatha(X) :
  – add the domain as the body literal: domain(X,S_1,S_n)

• for each of the remaining axioms, post a constraint in the body of constraint clause with all the variables in the axiom and the domain. Go to “Posting\_Constraint” (below subsection)

• add the search method: labeling([], [X])

• add , between all literals in the constraint body

• end of the code

• the program should be run

• Ask the problem question
  – find all suspects that satisfy the main constraint predicate and output them as the suspects that cannot be ruled out
  – output the suspects that do not satisfy the constraint predicate as the excluded suspects

**Posting\_Constraint:**

*For the axioms with universal quantifiers:*

1. remove the implication and re-write the statement in CNF: for instance: $A \rightarrow B$ will be $\neg A \lor B$

2. find all the combinations of the constants and the variables by replacing the variables with the domain values.

3. for each combination, instantiate the axiom as a constraint.

Item 2 and 3 can be done by posting three more predicates. The first predicate creates a list containing all possible combinations of the constant and the domain variables. The second predicate recurses through the list and calls the third predicate. The third predicate instantiate the axiom as constraint and impose it to the combinations. For instance, let us translate one of the axioms:

$$\forall X(hates(\text{agatha}, X) \rightarrow hates(\text{butler}, X)).$$

post\_constraint(X, Suspects) :
  findall(X, hates(\text{agatha}, X), List),
  post\_constraint2(list).
% all predicates should be parsed before being posted, i.e., the correct syntax should be different.

The result of the list will be \([\text{agatha, agatha, agatha, charles, agatha, butler}]\)

\[
\text{post\_constraint}_2([\text{agatha, X}|L]) : - \\
\quad \text{post\_constraint}_3(\text{agatha, X}), \\
\quad \text{post\_constraint}_2(L).
\]

\[
\text{post\_constraint}_3(\text{agatha, X}) : -, \\
\quad \text{hates}(\text{agatha, X}) \# \Rightarrow \text{hates}(\text{butler, X}).
\]

### 6.5 Front end of whodunnit combined reasoning system

The development of the whodunnit combined reasoning system is still at the preliminary stages. The front end is shown below:

![Figure 6.1: Front end of Whodunnit Combined Reasoning System](image)

As figure 6.1 depicts, problem can be expressed in problem description section. There are console buttons enabling the user to employ any of the techniques to solve the problem as well as combined button which is to perform the combination of reasoning systems.
Translation to any of the syntaxes and the results of any applied technique can be shown in the related sections.