CS 323, Fall 1999
Problem Set #2: Situation Calculus and the Frame Problem
Handout #15

Due in class on Thursday, November 18. (Two weeks from today.)

The purpose of this problem set is twofold. One is to introduce you to situation calculus and the frame problem. The other is to give a description of a theorem prover called Otter, which we will use to write and prove logical theories. There are seven problems, and one bonus question.

The material in this handout was largely pilfered from [Sha97], so if you are interested/want more detail about the stuff here, we highly recommend this book!

1 Situation Calculus

1.1 The Basics

Situation Calculus was introduced in [MH69] as an ontology of time used to represent actions and change. There are lots of flavors of situation calculus, as well as other ontologies of time. We will present two flavors of sitcalc in this section. Later on in the course we might cover other formalisms for representing change, which include an interval-based formalism, event calculus, and histories.

Situation Calculus has three sorts or types: situations, actions, and fluents.[1] A situation is "the complete state of the universe" [MH69]. Actions are obvious, and fluents are like propositions. There is one predicate $Holds(f, s)$ which asserts that fluent f is true in situation s. So to say it is raining at situation $S0$, we write $Holds(Raining, S0)$.[2] $Result(a, s)$ denotes the situation resulting from performing the action a in situation s. Thus $Result(Toggle(Switch1), S0)$ is the situation resulting from toggling the switch $Switch1$ in situation $S0$.

The structure of situations can be thought of as a tree, with $Result(a, s)$ branching off new situations, one for each action a from a situation s, and $Holds(f, s)$ holding at each of the nodes. There are two ways of interpreting these situations further. One is where each situation is uniquely determined by its place in the tree of situations (the *arboreal* interpretation), and the other is where each situation is uniquely determined by what fluents are true in it (the *situation state* interpretation).

1.1.1 Arboreal Interpretation

In this notion, as mentioned above, each situation is defined by its place in the situation tree. To formalize this notion, we need to add the following two axioms:

---

[1] Technically, there is also a fourth sort of objects, which are passed as arguments to fluents. So for example A and B are objects while $On(A, B)$ is a fluent.

[2] In [MH69], fluents were defined as predicates that took a situation as an argument. Instead of $Holds(Raining, S0)$ we would say $Raining(S0)$. There are pluses and minuses to each representation: using $Holds$ allows one to quantify over fluents in first-order logic. (We can say things like $\forall f. Holds(f, s)$.) The minus is a philosophical objection, where the reification of propositions is denounced as "ad hoc."[Gal91] The process of changing the predicate $Raining(s)$ into an object $Raining$ which is passed as an argument to a predicate is known as *reification*.

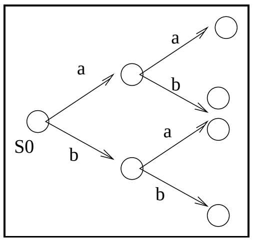

Figure 1: The Tree of Situations.

$$\forall a_1, a_2, s_1, s_2 . Result(a_1, s_1) = Result(a_2, s_2) \rightarrow (a_1 = a_2 \wedge s_1 = s_2) \tag{1}$$

$$\forall a, s . \neg Result(a, s) = S0 \tag{2}$$

These two axioms are known as the *axioms of arboreality.* Each situation s is unique, and determined by the chain of actions leading from $S0$ to s. These are very similar to uniqueness of names axioms. Equation 1 in the converse direction says that if either two situations or actions are different, then the situation resulting from each action is also different. Equation 2 says that there is no action and situation which can lead to $S0$.

Here is some more intuition about the formulas above. Equation 1 ensures that the graph of situations is really a tree – no branch is allowed to loop back to a situation upstream. (Try formulating a counter-example to see how this sentence prevents looping back.) Equation 2 prevents any situation from generating the special situation $S0$, or in other words, letting the entire tree loop back to its root. $S0$ is the special situation denoting the "first" situation. John calls it the "big bang" treatment of situations, as everything begins from $S0$.

Sometimes the induction axiom is also mentioned:

$$\forall \phi . [(\phi(S0) \wedge \forall a, s . \phi(s) \rightarrow \phi(Result(a, s))) \rightarrow \forall s . \phi(s)] \tag{3}$$

This axiom is used to prevent infinite chains in our tree of situations, where the chain doesn't "stem" from $S0$ as depicted below:

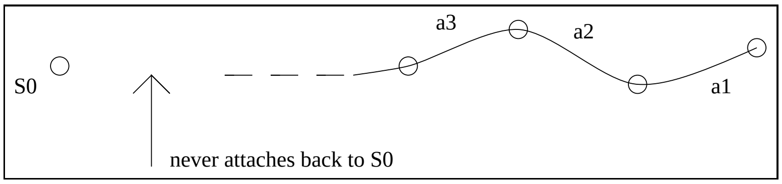

Figure 2: An Example of an Infinite Chain.

The induction axiom says that all situations are generated by some sequence of actions from $S0$. (If we prove that any predicate Φ is true of $S0$, and is preserved from one situation to the next over any action a, then we've shown it true for every situation.) Another way to interpret it is that the set of situations possible is the smallest set S such that $S0 \in S$ and if $s \in S$ then so is $Result(a, s) \in S$.[3]

1.1.2 Situation State

In the arboreal interpretation, situations are defined by the chain of actions which generate it from $S0$. An alternate interpretation is where a situation is determined by the fluents that are true in it. This is formalized by the following axiom:

$$\forall s_1, s_2 . s_1 = s_2 \iff [\forall f.(Holds(f, s_1) \iff Holds(f, s_2))] \tag{4}$$

Here we assert that if two situations have all the same fluents true (and false) in them, then they must be the same situation. In this case our tree of situations could become a graph. For example, if we had one action $Toggle$ which toggles a switch $Switch1$ on or off, then we would have the following depiction of situations:

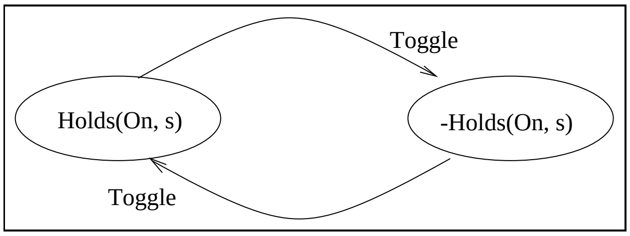

Figure 3: Situation States Collapse to a Cycle for Toggle.

Compare this to the arboreal treatment:

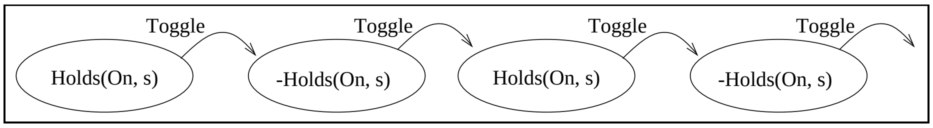

Figure 4: Arboreal Situations Do Not Collapse for Toggle.

Note that we would have to do special things to the situation-state version to account for time – we don't want time to be the same if we toggle a light switch on, off and then on again!

The general terminology is that states are objects which are uniquely determined by a set of fluents, while situations are the arboreal things that are defined by its actions.[4]

Problem 1: Prove that arboreal situation calculus defined by axioms 1 and 2 has no loops.

[3] We could rewrite Equation 3 as $\neg\exists\phi.(\phi(S0) \wedge \forall a, s.\phi(s) \rightarrow \phi(Result(a, s)) \wedge \exists s.\neg\phi(s))$.

[4] Sometimes other connotations are given to states, in that they are determined by a fixed set of fluents, and thus can be represented as a vector of values. Situations then by contrast are these rich objects, where one can never fully describe all properties. In this mind-set situations are definitely more elaboration tolerant than states.

Problem 2: Say we have A actions and F fluents. How many unique situations are possible in the arboreal interpretation? What about the situation state interpretation? (Remember, situations can be incompletely specified!) (We may not have either $Holds(f, s)$ or $\neg Holds(f, s)$)

1.2 Limitations of Situation Calculus?

This discrete, tree-like representation for reasoning about actions and change seems nice. Some skeptics might point out it is also very limiting. Some objections to situation calculus are that it cannot represent concurrent actions, continuous change, or actions where the order is unknown. It can't talk about divisible actions, or delayed effects of actions. It turns out that with a little extra machinery, we can represent each of these, as shown in [GLR91].

Problem 3: Pick one of the solutions in [GLR91] and explain it in your own words. You can get this paper either at the Math and Computer Science library, or from the TA. (Just let me know.)

1.3 The Frame Problem

The *frame problem* arises when formalizing change in a domain. For each action a, one has to write an axiom describing under what preconditions it succeeds, along with what is changed. A sample axiom is below:

$$\begin{aligned}&\forall x, y, s. Holds(Clear(x), s) \wedge Holds(Clear(y), s) \wedge x \neq y \wedge x \neq Table \rightarrow \\ &\quad Holds(On(x, y), Result(Move(x, y), s))\end{aligned} \tag{5}$$

The antecedent of the implication are the preconditions for the action $Move(x, y)$ to succeed, and in the resulting situation $Result(Move(x, y), s)$, we know that x is on top of y. Formulas like this are known as *effect axioms*, for obvious reasons.

It seems easy – to formalize a domain, just take each action, decide what its preconditions and effects are, and then just write down a set of effect axioms. The problem is, the fact that the other fluents not affected by the action do not change also has to be formalized. For example, moving block x to y doesn't change the color of any of the blocks. Also the location of some other block z (different from x and y) is still the same. We could go on and on. To be able to derive that the color of any block isn't affected by the move, we would have to write something like:

$$\forall x, y, z, c, s. Holds(Color(z, c), s) \rightarrow Holds(Color(z, c), Result(Move(x, y), s)) \tag{6}$$

To say that other blocks don't move requires an axiom like:

$$\forall x, y, v, w, s. Holds(On(v, w), s) \wedge x \neq v \rightarrow Holds(On(v, w), Result(Move(x, y), s)) \tag{7}$$

We have to write similar axioms for every fluent that is not affected by an action. If there are n fluents and m actions, then we will have to write something on the order of $m \times n$ frame axioms!

This is the essence of the frame problem. It's not that we can't represent change; we just can't represent it compactly.

It also seems non-intuitive that the fact that the fluents don't change must be represented. A person doesn't need to re-compute all unaffected fluents after an action takes place. A robot, if it is going to do efficient reasoning, shouldn't have to either.

One last note: the frame problem is not limited to situation calculus – it pops up in any logical representation which formalizes actions and change. The next few sections talk about possible solutions to the frame problem, and how they fail in one way or another.

1.3.1 Formalizing the Common Sense Law of Inertia

The common sense law of inertia states: "Normally, given any action (or event type) and any fluent, the action doesn't affect the fluent." We can say more formally: unless an action/fluent pair is abnormal in some way, in a situation, the fluent will have the same value both before and after the action takes place. In logic this becomes:

$$\forall a, f, s. \neg Ab(a, f, s) \rightarrow (Holds(f, s) \equiv Holds(f, Result(a, s))) \tag{8}$$

The equivalence above preserves both negative and positive information – if f was false in situation s, it will remain false in the situation $Result(a, s)$, assuming that a, f, and s aren't together abnormal in some way.[5]

The idea of this approach is to combine the initial observations, like $Holds(On(A, Table), S0)$, with the effect axioms for each action. Then append equation 8 (instead of writing $m \times n$ frame axioms) to get a logical theory Σ. Circumscription can do the work of the $m \times n$ frame axioms, by minimizing the extent of Ab in the theory Σ, and letting $Holds$ vary.[6] (This is written $Circ[\Sigma; Ab; Holds]$.) Hopefully this theory $Circ[\Sigma; Ab; Holds]$ will have a first order equivalent, which will also dictate that only the fluents we want will persist.

Intuitively, this should work – by circumscribing Ab we guarantee that as many fluents' values as possible will persist ($Holds(f, s) \equiv Holds(f, Result(a, s))$) when the theory allows them to. By default, most fluents persist, which is the notion behind the common sense rule of inertia.

1.3.2 The Yale Shooting Problem

Unfortunately, in 1986, Hanks and McDermott found a counterexample which got unintuitive results. The counterexample is known as the "Yale Shooting Problem" [HM86], and is now the benchmark used for testing other solutions to the frame problem. The story is that there are three actions, *Load*, *Wait*, and *Shoot*, and a gun and a person, named Fred. There are two fluents: *Alive* and *Loaded*. The gun is initially unloaded and Fred is alive. The object is to predict that Fred will be dead after loading the gun, waiting, and shooting.

In logic we can formalize the entire theory Σ as:

---

[5] We can rewrite $Holds(f, s) \equiv Holds(f, Result(a, s))$ in the equivalent form $Holds(f, s) \rightarrow Holds(f, Result(a, s)) \wedge \neg Holds(f, s) \rightarrow \neg Holds(f, Result(a, s))$.

[6] If you wonder why $Holds$ must vary, think about the minimization model-theoretically. When comparing models for Ab, you don't want to have to restrict yourself only to models which have the same extension for $Holds$. If anything you want to make your ordering relation $\leq_{Ab}$ as strong as possible to get fewer minimal models.

$$
\begin{aligned}
&Holds(Alive, S0)\\
&\neg Holds(Loaded, S0)\\
\\
&\forall s.Holds(Loaded, Result(Load, s))\\
&\forall s.Holds(Loaded, s) \rightarrow \neg Holds(Alive, Result(Shoot, s))\\
&\forall s.\neg Holds(Loaded, Result(Shoot, s))\\
\\
&\forall a, f, s.\neg Ab(a, f, s) \rightarrow (Holds(f, s) \equiv Holds(f, Result(a, s)))
\end{aligned} \tag{9}
$$

The first two axioms above are the observation sentences. The next three are our effect axioms, and the last is the common sense law of inertia. What we would like to have is that:

$$Circ[\Sigma; Ab; Holds] \models \neg Holds(Alive, Result(Shoot, Result(Wait, Result(Load, S0)))) \tag{10}$$

Unfortunately this doesn't hold, because we get unintended models of $Circ[\Sigma; Ab; Holds]$.

Problem 4: In order to get the result of the circumscription above, we could do some fancy syntactic manipulations. This in general is messy and not so gratifying. Another way is to enumerate and inspect the models of $Circ[\Sigma; Ab; Holds]$. It turns out each model can be represented as a graph with the fluents *Loaded* and *Alive* as boolean functions of time, so not only are the models easy to determine, but simple to interpret.

Below is a template graph – we know that *Loaded* is false in $S0$, and *Alive* is true. After the *Load* action, we know that the gun must be *Loaded*, according to the third axiom in 9. Finally, we also know that the gun is not *Loaded* after the *Shoot* action. These set value are blacked in. This leave us four intervals (dashed) over which the fluents can be true or false, yielding $2^4 = 16$ possible combinations.

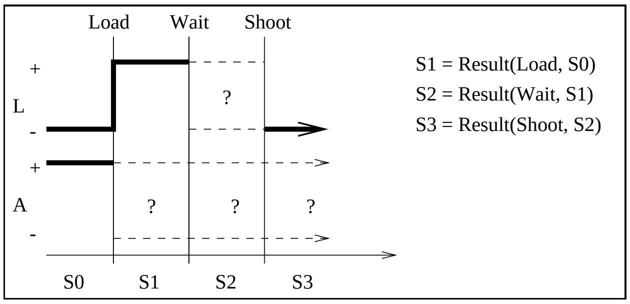

Figure 5: A Template Graph to Represent the Yale Shooting Problem.

Here is a graph corresponding to the intended model – the gun stays loaded after the $Wait$ action, and Fred dies at the end of the *Shoot* action:

This model has three abnormalities, one for each discontinuity in the graph. Logically the *Ab*s are: $Ab(Load, Loaded, S0)$, $Ab(Shoot, Loaded, Result(Wait, Result(Load, S0)))$, and $Ab(Shoot, Alive, Result(Wait, Result(Load, S0)))$.

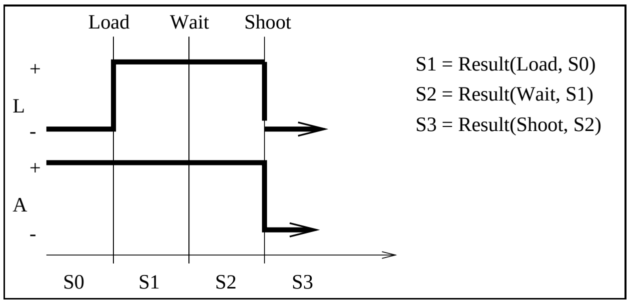

Figure 6: Intended Model of the Yale Shooting Problem.

1. **Enumerate the remaining eleven unintended models of $Circ[\Sigma; Ab; Holds]$, by trying the other fifteen possibilities for the unspecified intervals. (Four of the fifteen won't be legal because they won't obey the fourth axiom in 9.) Draw them all out in the same form as in figure 6. (Yes it is a pain but it is good for you!)**

2. **By inspection, you should be able to draw a Hasse diagram based on the ordering $\leq_{Ab;Holds}$ on all twelve models. [Hint: Since *Holds* is varied, you don't need to worry about the values of *Loaded* and *Alive*. All you have to do is order the models based on subset inclusion of the *Ab*s, which exactly correspond to transitions in the time functions.]**

   [Hint #2: You can split the twelve models into two groups which are mutually incomparable if you look at the *Ab*s from the behavior of *Loaded* – four will have one *Loaded* function while eight will have another, and the *Ab*s in each are not inclusive of the other.]

   [Hint #3: One set will look like a tree and the other a cube.]

3. **We can read off the minimal models by inspection of the Hasse diagram. It turns out there are four *Ab*-minimal models. For each, note how many *Abs* there are, and give a brief English description of each.**

1.3.3 Ensuring Fred Dies

The Yale Shooting Problem demonstrates that the naive approach for minimizing change does not work. John mentioned a few solutions in class:

1. **Chronological minimization:** The unintended models for the Yale Shooting Problem all have one thing in common – change happens too soon. The idea behind chronological minimization is to prefer models which delay change as much as possible.

   For the Yale Shooting Problem, this approach will deliver the intended model where Fred dies. Unfortunately, there is no reason to prefer change to happen later rather than sooner. The "Stolen Car Scenario" [Kau86] is an example which highlights the arbitrariness of chronological minimization: Say a person parks his car. He waits one hour, and then another hour. When he returns to the parking space at 2pm he finds the car missing – it has been stolen! The car must have been stolen either during the first wait (from noon to

1pm) or the second wait (from 1pm to 2pm). Common sense dictates that either of these times are equally likely, but chronological minimization will declare the car was stolen between 1 and 2pm.

2. **Causal Minimization:** This is a more elegant approach, and also more intuitive. First two new predicates are added to the language, $Causes(a, f, v)$ (a causes f to take value v), and $Precond(f, v, a)$ (f has to have value v for a to succeed. (v can be either "true" or "false".) There are also two auxiliary predicates, $Success(a, s)$ and $Affects(a, f, s)$, which are defined in terms of $Causes$ and $Precond$. Instead of using the common sense law of inertia with Ab, we write axioms (not listed here) which say that

   (a) If action a causes fluent f to take on value v (true or false), and the action succeeds, then f will be v in the situation $Result(a, s)$.

   (b) If action a doesn't affect fluent f in s, then the value of f stays the same in $Result(a, s)$.

   (c) (definition of $Success$) a succeeds in s iff every fluent f which needs to have the value v in order for a to happen does so.

   (d) (definition of $Affects$) a affects f in situation s if a succeeds in s and a causes f to take on some value (either true or false).

   We write a theory now in terms of the new predicates $Causes$ and $Precond$, and then take the theory $Circ[\Sigma; Causes, Precond; Holds]$. Circumscribing $Precond$ directly addresses the *qualification problem.*[7] Minimizing $Causes$ ensures that a fluent will change only if there is a cause for it to do so.

   This works for the Yale Shooting Problem, and for all problems which are of the type where one predicts the future. The theory of causal minimization is well understood, and built on some nice principles which determine that it won't work for explanation problems, which are problems where we are given the beginning and end situations, and asked to find out what happened in between. It also cannot represent context-dependent effects (Toggling a switch is either "On" or "Off", depending on the previous state) and theories which contain domain constraints. (At least not without some tinkering.)

3. **Explanation Closure:** [Haa87] noted that axioms can be hand-crafted in a more compact form than frame axioms, so that we can remain in the realm of classical logic. For each fluent f, two axioms are written:

$$\begin{array}{l}
\forall a, s.(Holds(f, s) \wedge \neg Holds(f, Result(a, s))) \rightarrow \\
\quad ((a = action_1 \wedge \text{(whatever needs to } Hold \text{ in } s \text{ for } action_1 \text{ to happen)}) \\
\quad \vee \ldots \vee \\
\quad (a = action_n \wedge \text{(whatever needs to } Hold \text{ in } s \text{ for } action_n \text{ to happen)}) \\
\\
\forall a, s.(\neg Holds(f, s) \wedge Holds(f, Result(a, s))) \rightarrow \\
\quad ((a = action_1 \wedge \text{(whatever needs to } Hold \text{ in } s \text{ for } action_1 \text{ to happen)}) \\
\quad \vee \ldots \vee \\
\quad (a = action_n \wedge \text{(whatever needs to } Hold \text{ in } s \text{ for } action_n \text{ to happen)})
\end{array} \tag{11}$$

[7] The qualification problem is the fact that there are "an impractical and implausible number of qualifications" [McC80] that must be included to get all the preconditions for success of an action. Minimizing the extent of $Prec$ is the same as asserting that the only preconditions that hold for an action a are those we explicitly stated or that follow from a theory.

These explanation closure axioms give the *only* reason why a fluent changes from true to false, or false to true. These axioms, along with the effect axioms and observation sentences, form the desired theory.

Here are the explanation closure axioms for the Yale Shooting Problem:

$$\begin{aligned}
&\forall a, s.(Holds(Loaded, s) \land \neg Holds(Loaded, Result(a, s))) \rightarrow (a = Shoot) \\
&\forall a, s.(\neg Holds(Loaded, s) \land Holds(Loaded, Result(a, s))) \rightarrow (a = Load) \\
&\forall a, s.(Holds(Alive, s) \land \neg Holds(Alive, Result(a, s))) \rightarrow (a = Shoot \land Holds(Loaded, s)) \\
&\forall a, s.(\neg Holds(Alive, s) \land Holds(Alive, Result(a, s))) \rightarrow \perp
\end{aligned} \tag{12}$$

The fourth explanation closure looks strange - -all it says is that there are no conditions ($\perp$) under which Fred is not *Alive* in a situation and then *Alive* in a resulting situation. This statement is equivalent to:

$$\forall a, s.\neg(\neg Holds(Alive, s) \land Holds(Alive, Result(a, s)))$$

or

$$\forall a, s.Holds(Alive, Result(a, s)) \rightarrow Holds(Alive, s)$$

4. **Successor State (a variation):** These axioms are also hand-crafted, and also do not require non-monotonic machinery. The basic form of these axioms is:

$$\begin{aligned}
&\forall a, f, s.Holds(f, Result(a, s)) \equiv \\
&\quad \lor_i \{a_i = \text{<action which makes f true>} \land \text{<all preconditions of } a_i \text{ hold>}\} \\
&\quad \lor \\
&\quad Holds(f, s) \land \neg \lor_j \{a_j = \text{<action which makes f false>} \land \text{<all preconditions of } a_j \text{ hold>}\}
\end{aligned} \tag{13}$$

The intuition behind the logic is trivial: A fluent will be true after action a in exactly two cases. Either one of the actions which made it true succeeds, or it was true before and none of the actions that can falsify f successfully occur.

Note that there is one explanation closure axiom for every fluent f.

Bonus: Write the four axioms needed for causal minimization in first-order logic, using only the predicates $Affects$, $Causes$, $Holds$, $Success$, $Precond$, and $=$.

2 Otter

Otter is a theorem prover written by William McCune. It's a resolution-style theorem proving program that uses first-order logic with equality. It has been used to prove some complex

theorems, including the fact that Robbins algebras are Boolean.
(See `http://www-unix.mcs.anl.gov/m̃ccune/papers/robbins/` .
Otter comes with many bells and whistles, which are not necessary for this problem set. This section just shows you how to run Otter, and write logic in a form Otter can understand. The manual is available on the Leland system at `/usr/class/cs323/otter-3.0.4/document/`. There's also a URL, over at
`http://www-unix.mcs.anl.gov/AR/otter`. You can download a free version for your PC if you end up enjoying theorem proving.

2.1 Commands

To start, first log onto a saga (I've tested it on these machines). From your home directory, try this command:

```
/usr/class/cs323/bin/otter < /usr/class/cs323/examples/socrates.in
```

The output you get (try redirecting it to a file via >) should be the same as in `/usr/class/cs323/examples/socrates.out`. Make sure not to leave out the "<"!! Otter works by piping in files of logical axioms.

2.2 Format of Input File

The format of an otter input file for this problem set is simple. The first statement is `set(auto)`, which tells the program to automatically choose a theorem proving strategy based on the structure of the theory. Then comes `formula_list(usable).`, which begins the lists of logical axioms. The list is ended by `end_of_list.`. Note that every command (and axiom) is punctuated with a period.

The axiom `all _x (_x = _x).` must be in your `formula_list`, otherwise equality won't be handled correctly. After that, you are free to write your own axioms. Here is a table of relevant logical operators:

negation	-
disjunction	|
conjunction	&
implication	->
equivalence	<->
existential quantification	exists
universal quantification	all

Predicates, constants, and functions may begin with either upper or lower case variables. Otter has some flaky rules on what is a variable, so it is recommended that all variables be explicitly preceded with a "_". (The underscore signals that it is definitely a variable!) If your theorems do not seem to work, please check that all variables have been pre-under-scored!

The logical symbols `- | & -> <->` are used as usual. Precedence should be the same as well, though you should use parentheses if you are not sure. The form of axioms with quantification is different from convention, as shown in these examples:

```
all _a _b _c (P(_a) <-> (Q(_b, _c) & R(_c))).
exists _x (P(_x)).
```

There are no commas between quantified variables, and no period except at the end of the sentence. Also, the quantified part of the formula is encapsulated in parentheses.

The conclusion, the formula you want to prove, also goes in the `formula_list`, except that it is negated. Otter works by resolving clauses until it hits $\perp$.

2.3 Format of Output File

The output file has eight sections:

1. A listing of the flags. (Ignore these!)
2. Copy of your formulas.
3. Your formulas converted to CNF.
4. A description of the clauses along with the strategy to be used.
5. Further processing of clauses
6. Resolution "Search"
7. Proof
8. Random Statistics

The only section that is important is the (7) – the proof. (3) is a good place to check your axioms.

Problem 5: "Everybody loves my baby, but my baby only loves me." Formalize this in first-order logic, and then use Otter to show that $Me = Baby$. Use the sample file at `/usr/class/cs323/examples/ps2-4.in` as an initial template.

Problem 6: Write the explanation closure axioms for the Yale Shooting Problem in otter style. Combine with the initial observations and effect axioms (everything but the last axiom in 9). Use otter to prove that

$\neg Holds(Alive, Result(Shoot, Result(Wait, Result(Load, S0))))$. Use the sample file at `/usr/class/cs323/examples/ps2-5.in` as an initial template.

Problem 7: Write the successor state axioms for the Yale Shooting Problem in otter style. Combine with the initial observations. Use otter to prove that $\neg Holds(Alive, Result(Shoot, Result(Wait, Result(Load, S0))))$. Use the sample file at `/usr/class/cs323/examples/ps2-6.in` as an initial template.

References

[Gal91] Antony Galton. **reified temporal theories and how to unreify them**[8]. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI'91)*, pages 1177–1182, 1991.

[8] http://www.dcs.ex.ac.uk/ antony/abstracts/ijcai91.html

[GLR91] Michael Gelfond, Vladimir Lifschitz, and Arkady Rabinov. What are the limitations of the situation calculus? In Robert Boyer, editor, *Automated Reasoning: Essays in Honor of Woody Bledsoe*, pages 167–179. Kluwer Academic, Dordrecht, 1991.

[Haa87] A. R. Haas. The case for domain-specific frame axioms. *Proceedings of the 1987 Workshop on the Frame Problem*, pages 343–348, 1987.

[HM86] S. Hanks and D. McDermott. Default reasoning, nonmonotonic logics and frame problem. In *Proceedings of AAAI-86*, pages 328–333. Morgan Kaufmann, 1986.

[Kau86] H. Kautz. The logic of persistence. *Proceedings of the AAAI 86*, pages 401–405, 1986.

[McC80] John McCarthy. Circumscription—A Form of Non-Monotonic Reasoning[9]. *Artificial Intelligence*, 13:27–39, 1980. Reprinted in [McC90].

[McC90] John McCarthy. *Formalizing Common Sense: Papers by John McCarthy.* Ablex Publishing Corporation, 355 Chestnut Street, Norwood, NJ 07648, 1990.

[MH69] John McCarthy and Patrick J. Hayes. Some Philosophical Problems from the Standpoint of Artificial Intelligence[10]. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, 1969.

[Sha97] Murray Shanahan. *Solving the Frame Problem, a mathematical investigation of the common sense law of inertia.* M.I.T. Press, 1997.

[9] http://www-formal.stanford.edu/jmc/circumscription.html
[10] http://www-formal.stanford.edu/jmc/mcchay69.html