CLAR paper

SMU CCLAW

July 5, 2021

Abstract

Your abstract.

1 Introduction

Let the tuple $Config = (R, F, M, I)$ denote a *configuration* of legal rules. The set R denotes a set of rules of the form $pre\_con(r) \rightarrow concl(r)$. These are 'naive' rules with no information pertaining to any of the other rules in R. F is a set of literals and predicates that describe facts of the legal scenario we wish to consider. M is a set of the binary predicates *despite*, *subject_to* and *strong_subject_to*. I is a collection of minimal inconsistent sets of positive atoms/predicates. Henceforth for a rule r, we may write C_r for it's conclusion $Concl(r)$

The binary predicate $legally\_valid(R, C)$ intuitively means that the rule R enforces conclusion C. The unary predicate $is\_legal(C)$ intuitively means that C legally holds. The predicates *despite*, *subject_to* and *strong_subject_to* all cause some rules to override others. Their precise semantics will be given next.

1.1 Semantics

A set S of *is_legal* and *legally_valid* predicates is called a *model* of $Config = (R, F, M, I)$, if and only if

A1) $\forall f \in F$ $is\_legal(f) \in S$.

A2) $\forall r \in R$, if $legally\_valid(r, C_r) \in S$. then $S \models is\_legal(pre\_con(r))$ and $S \models is\_legal(C_r)$ (this is a slight abuse of notation. Explain later?)

A3) $\forall c$, if $is\_legal(c) \in S$, then either $c \in F$ or there exists $r \in R$ such that $legally\_valid(r, C_r) \in S$ and $c = C_r$.

B1) $\forall r_1, r_2 \in R$, if $despite(r_1, r_2) \in M$ and $S \models is\_legal(pre\_con(r_2))$, then $legally\_valid(r_1, C_{r_1}) \notin S$

B2) $\forall r_1, r_2 \in R$, if $strong\_subject\_to(r_1, r_2) \in M$ and $legally\_valid(r_1, C_{r_1}) \in S$, then $legally\_valid(r_2, C_{r_2}) \notin S$

B3) $\forall r_1, r_2 \in R$ if $subject\_to(r_1, r_2) \in M$, and $legally\_valid(r_1, C_{r_1}) \in S$ and there exists a minimal conflicting set $i \in I$ such that $is\_legal(C_{r_1}) \in i$, $i \not\subseteq S$, and $i \subseteq S \cup \{is\_legal(C_{r_2})\}$, then $legally\_valid(r_2, C_{r_2}) \notin S$.

B4) $\forall r \in R$, if $S \models pre\_con(r)$, but $legally\_valid(r, C\_r) \notin S$, then at-least one of B1, B2, B3 hold.

2 ASP encoding

Here is an ASP encoding scheme given a configuration $Config = (R, F, M, I)$ of legal rules.

```
% For any f in F, we have:
is_legal(f).

% Any rule r in R is encoded using the general schema:
according_to(r,C_r):-is_legal(pre_con(r)).

% Say {a,b,c} is a minimal inconsistent set in I, then this would get encoded as:
opposes(a,b):-is_legal(c)
opposes(a,c):-is_legal(b).
opposes(b,c):-is_legal(a).
% The above is done for every minimal inconsistent set

opposes(X,Y):-opposes(Y,X).



% Encoding for 'despite'
defeated(R,C):-according_to(R,C),according_to(R1,C1),despite(R,R1).

% Encoding for 'subject_to'
defeated(R,C):-according_to(R,C),legally_valid(R1,C1),opposes(C,C1),subject_to(C1,C).

% Encoding for 'strong_subject_to'
defeated(R,C):-according_to(R,C),legally_valid(R1,C1),subject_to(C1,C).
```

```
legally_valid(R,C):-according_to(R,C),not defeated(R,C).

is_legal(C):-legally_valid(R,C).
```

2.1 Main claim

For a configuration $Config = (R, F, M, I)$, let the above encoding be the program ASP_{Config}. Then the sets of *is_legal* and *legally_valid* predicates from answer sets of ASP_{Config} correspond exactly to the *models* of $Config$.

2.2 Comments/Thoughts

It is not entirely clear to me how to unify this approach with Martin's approach to the semantics of L4/where this stuff fits in exactly. However I think it can be done? It seems that our notions of what a rule is match well. Class hierarchies can be incorporated using rules like $is\_truck(X) \rightarrow is\_vehicle(X)$. In this set-up this information along with minimal inconsistent sets can all be stored in the parameter I from the tuple. So perhaps we could rename it T for background theory?

The minimal inconsistent sets in I seem to somewhat loosely correspond to 'assertions' in Martin's set up although they seem to mean different things. In this set up, they are meant to encode basic domain knowledge about things that contradict each other. But I think they can also be used as a tool to check properties of the program.

Our notions of 'subject to' are different but 'strong subject to' here seems close to Martin's 'subject to'.

The intuition for the three modifiers here is that with despite, once the precondition of the higher priority rule is true, it invalidates the lower priority rule regardless of whether the higher priority rule actually comes into effect. With 'strong subject to', once the higher priority rule is in effect, then it invalidates the lower priority rule. With 'subject to', The higher priority rule has to be in effect and it has to contradict the lower priority rule for the lower priority rule to be invalidated.

2.3 (Slightly worrying) Issues

1. I haven't really thought about how to handle disjunctions/conjunctions in rule conclusions.

2. I think the main claim 2.1 is correct but I haven't properly proved it yet...

3. Although this builds on Jason's work,this framework still needs to be tried out on proper test cases.

References