Motivation for a resolution system for a class of prefixed formulas

Adrian Williams, Jim Cunningham
{aw3,rjc}@doc.ic.ac.uk
Deparment of Computing, Imperial College

1 Motivation

There have been a number of attempts to devise useful temporal and modal logic programming languages ([3, 2]). Approaches range from extending horn clauses with a couple of basic operators (Chronolog) to providing a complete modal framework in which a modal logic can be *instantiated* (Molog). However, there has yet to be a modal logic programming language which has done for modal logic what Prolog did for classical logic. As an explanation for this, we would suggest that existing systems suffer from one or more of the following:

- Not expressive enough -
  Some languages do not add enough when compared to Prolog to be useful. In fact extra logical features such as cut are usually lost.
- Too slow -
  The majority of existing systems are too slow to be off any use in large-scale applications
- No existing implementation -
  Some proposed languages don't appear to have a useable implementation or rely on a translation into another system for execution.
- Not Prolog-like enough -
  Prolog's syntax and execution strategy are readily understood by anyone with a background in programming and some understanding of logic. Some proposed modal programming languages have complex syntax and/or operational semantics and are only understandable by people with a strong (modal) logic background, hence reducing their appeal.

In our work with prefixed resolution we are attempting to address all four of the above concerns

2 Prefixes

The idea of using prefixes with modal formula goes back to Fitting and his prefixed tableaux ([1]). There a prefix is defined as a *finite* sequence of positive integers. In our system we have two types of prefix, basic prefixes and general prefixes. Basic prefixes are essentially the same as Fitting's prefixes except that they can be infinite. General prefixes are finite basic prefixes extended with additional constant symbols c,$C^\bullet$,C^+, and C^* as well as four types of variable V, $V^\bullet$,V^+ and V^* [1]. Our prefix notation makes use of general prefixes whilst our semantics are based on basic prefixes. In order to apply our semantics to our proof system we have an operator $\mathcal{G}$ which translates general prefixes into sets of basic prefixes.

It is our intention that our prefix notation should capture the essence of a number of modal logics whilst providing a mechanism for efficient computation. We do this by encapsulating the meaning of our modal operators with our constant symbols (c, $C^\bullet$, etc) which represent (infinite) sets of basic prefixes. For example, the general prefix $\langle 0, c\rangle$ represents all basic paths of the form $\langle 0, i\rangle$ where i is an integer. If we consider the formula $\Box X$ in the logic K, the usual meaning in a Kripke semantics is that X holds at all worlds that are accessible from the world at which $\Box X$ holds. If we state that $\Box X$ holds at the world $\langle 0\rangle$ and that all worlds of the form $\langle 0, i\rangle$ are accessible [2] from $\langle 0\rangle$, then we can see that $\langle 0, c\rangle : X$ represents all the information provided by $\langle 0\rangle : \Box X$. The other constant symbols allow us to express the meaning of $\Box$ in other normal logics. E.g. the general prefix $\langle 0, C^+\rangle$ represents all basic paths of the form $\langle 0, ...\rangle$ which have a length of at least 2 and captures the nature of a transitive accessibility relation.

[1] General prefixes are essentially a reworking of the notation used in [5]
[2] In our semantics the accessibility relation is defined as the set $R = \{(\sigma, \tau) | \tau = \sigma; \langle i\rangle\}$

3 Prefix resolution

Using the above prefix notation and its associated semantics we can construct a resolution system which we can show to be sound and complete. The system depends on our ability to determine whether two general prefixes represent overlapping sets of worlds. For this purpose we have a unification operator, υ, which has the properties $\mathcal{G}(\upsilon(\sigma,\tau)) = \mathcal{G}(\sigma) \cap \mathcal{G}\tau$

The (simplified) core rule of our system is:

$$\frac{\begin{array}{c}\sigma : A \vee X \\ \neg\tau : A \vee Y\end{array}}{X \vee Y} \text{ if } \upsilon(\sigma,\tau) \neq \{\}$$

This rule applies provided we can show that the unifier represents at least one possible world in the model to be constructed. For example, consider the two modal clauses $\Box A$ and $\Box\neg A$. These would be translated to the prefixed clauses, $\langle 0, c\rangle : A$ and $\langle 0, c\rangle : \neg A$. According to the above rule these would resolve to give $\{\}$ as the prefixes unify with the unifier $\langle 0, c\rangle$. However, the resolvent should not be derivable unless there exists at least one world accessible from $\langle 0\rangle$, e.g. if we have the clause $\Diamond B$. The rest of our resolution rules deal with how we can determine whether the set of worlds represented by the unifier contains a *possible* world.

4 Modal Prolog

Given the above prefixed resolution rules and some rules for translating modal formulas into prefixed formulas, it is an obvious step to try to combine them to form a modal prolog. The implementation is still a work in progress but below we present some code that will already run on our system. The example was inspired by [4] and so is obviously temporal in nature.

```
global first fib(0).
global first (next fib(1)).
global next (next fib(Z)) :- fib(X), next fib(Y), Z is X + Y.
```

This is translated to the prefixed program below. Note that we haven't changed the structure of the original program in anyway. This prefixed program can then be executed in exactly the way you would expect if you are familiar with Prolog.

$\langle X^*, 0\rangle : \mathrm{fib}(0).$
$\langle X^*, 0, 1\rangle : \mathrm{fib}(1).$
$\langle X^*, 1, 1\rangle : \mathrm{fib}(Z) : -\langle X^*\rangle : \mathrm{fib}(X), \langle X^*, 1\rangle : \mathrm{fib}(Y), \langle X^*\rangle : Z \text{ is } X + Y.$

References

[1] M. Fitting. *Proof Methods For Modal And Intuitionistic Logics.* Reidel, 1983.

[2] M. Gergatsoulis. Temporal and modal logic programming languages.

[3] M. A. Orgun and W. Ma. An overview of temporal and modal logic programming. In D. M. Gabbay and H. J. Ohlbach, editors, *Proceedings of ICTL'94: The 1st International Conference on Temporal Logic*, pages 445–479, Berlin Heidelberg, 1994. Springer-Verlag.

[4] M. A. Orgun and W. W. Wadge. Theory and practice of temporal logic programming. In L. F. del Cerro and M. Penttonen, editors, *Intensional Logics for Programming*, pages 23–50. Oxford University Press, 1992.

[5] J. Pitt and J.Cunningham. Distributed modal theorem proving with KE. In P. Miglioli, U. Moscato, D. Mundici, and M. Ornaghi, editors, *Proceedings 5th Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, volume 1071 of *LNAI*, pages 160–176. Springer-Verlag, 1996.