A Prefix Notation and Unification Algorithm for Encoding Modal Logics

Adrian Williams and Jim Cunningham

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

1 Introduction

When reasoning about modal logics, it is necessary to deal with the various properties of modal operators in different logics. When dealing with logics with well understood frame properties, it is common practice to try to encode (part of) the semantics at a syntactic level. Existing methods include expressing the accessibility relation classically (as relations and/or functions e.g. [6]) or by using prefixes ([1–4, 8]). The exact nature of a prefix varies between authors but they all have the basic premise that a prefix is a string that represents paths of worlds. Each formula is prefixed by one of these strings to indicate at which world(s) it must hold. Beyond this, each prefix system encodes a different amount of semantic information into a prefix as well as differing in terms of presentation.

In this paper we present a prefix notation that encapsulates the nature of a number of different prefix systems. It is our intention to provide a clear consistent notation that not only can be used in different proof systems but which can be reasoned about purely on its own terms without being concerned with the modal logics that are being encoded. Obviously, a complete proof system needs to be concerned about such matters but when it comes to sub-problems such as unification (the task of determining whether two prefixes represent common worlds) we wish to be able to solve this task by a straightforward analysis of the structure of the prefixes. One of the advantages of such a modular approach is that optimisations can be made to unification algorithm without affecting or even needing to understand the rest of the proof system.

2 Prefixes

We begin by presenting a family of prefixes, Σ_{XY}. Each set of prefixes is distinguished by the length of the contained prefixes and the symbols that are used to construct them.

Definition 1. *The set of general constants, $\mathcal{C}$, is the set $\{c, C_\bullet, C_+, C_*\}$*

Definition 2. *The sets $V, V_\bullet, V_+$ and V_* are disjoint countable sets of variable names.*

Definition 3. *$\mathcal{V}$ is the set of all variable names ($\mathcal{V} = V \cup V_\bullet \cup V_+ \cup V_*$).*

We will use variable names of the form $v, v_\bullet, v_+, v_*$ for variables that belong to the sets $V, V_\bullet, V_+$ and V_* respectively.

A prefix will be some sequence of the symbols $\mathbb{N}, \mathcal{V}$ and $\mathcal{C}$.

In order to describe the restrictions on the length of allowed prefixes in a particular prefix set we will use the following symbols:

- $\bullet$ prefixes must have a length of 0 or 1
- 1 prefixes must have a length of 1
- $+$ prefixes can be any non-empty sequence
- $*$ prefixes can be any sequence

Allowed symbols within a particular prefix set will be determined by subscripting the set name with the names of the allowed symbol sets.

For example, the prefix set $\Sigma_{\mathbb{N}\mathcal{V}\bullet}$ is the set of all sequences of the symbols $\mathbb{N} \cup \mathcal{V}$ that have the length 0 or 1.

The most important prefix set is the set $\Sigma_{\mathbb{N}*}$ as it is this set which is used to define the meaning of more *interesting* prefixes. We call $\Sigma_{\mathbb{N}*}$ the set of basic prefixes and will usually use the shorthand Σ.

The other prefix set of particular interest is the set $\Sigma_{\mathbb{N}\mathcal{V}\mathcal{C}*}$ as all other sets are subsets. We call this set the set of general prefixes and will use the shorthand Σ_*.

Definition 4. *A basic prefix is a potentially infinite sequence of natural numbers ($\mathbb{N}$).*

Definition 5. *Σ is the set of all basic prefixes ($= \Sigma_{\mathbb{N}*}$).*

Definition 6. *A general prefix is a finite sequence of the symbols $\mathbb{N} \cup \mathcal{C} \cup \mathcal{V}$.*

Definition 7. *Σ_* is the set of all general prefixes ($= \Sigma_{\mathbb{N}\mathcal{V}\mathcal{C}*}$).*

For reasoning about these prefixes we use a number of common operators.

We define '|' as our 'head and tail' operator, and ';' as our concatenation operator. i.e.

$$\langle h|T\rangle = \langle h\rangle; T$$

$|\sigma|$ is used to indicate the length of the prefix σ. e.g.

$$|\langle 1, 2, 1\rangle| = 3$$

$\sigma(i)$ refers to the value at ith of σ. e.g.

$$\langle 3, 2, 1\rangle(3) = 1$$

3 Unification

As we have already stated these prefixes are used to represent worlds at which formulas are known to hold. It is a requirement of any refutation system that we are able to determine when we have two contradictory statements. In this case, contradictory means that we have to contradictory formula holding at the same world. Therefore, it is necessary to be able to decide when two prefixes represent common worlds. This process is referred to as prefix unification.

3.1 Semantics

Basic Prefixes
Basic prefixes are our core prefix structure and each is unique. As such unification between basic prefixes is trivial. Two basic prefixes unify if they are equal.

$\Sigma_{\mathbb{N}\mathcal{V}*}$ prefixes
In our prefix system variables are typed. Variables from the sets V, $V_\bullet$, V_+, and V_* can be assigned values from the sets $\Sigma_{\mathbb{N}1}$, $\Sigma_{\mathbb{N}\bullet}$, $\Sigma_{\mathbb{N}+}$, and $\Sigma_{\mathbb{N}*}$ respectively. Taking these restrictions into account unification between $\Sigma_{\mathbb{N}\mathcal{V}*}$ prefixes is essentially a variation on string unification.

Definition 8. *A prefix substitution $\theta : \mathcal{V} \to \Sigma_{\mathbb{N}\mathcal{V}*}$ is a function mapping variables to prefixes that satisfies the following constraints:*

$$\begin{aligned} \theta(v) &\in \Sigma_{\mathbb{N}V1} \\ \theta(v_\bullet) &\in \Sigma_{\mathbb{N}V_\bullet\bullet} \\ \theta(v_+) &\in \Sigma_{\mathbb{N}VV_+ +} \\ \theta(v_*) &\in \Sigma_{\mathbb{N}\mathcal{V}*} \end{aligned}$$

A substitution θ can be represented by a set of (variable\prefix) pairs, $\{v\backslash\sigma \mid \theta(v) = \sigma \text{ and } \langle v\rangle \neq \sigma\}$.

The application $\theta(\sigma)$ (usually written $\sigma\theta$) of a substitution θ to a prefix σ leads to the prefix σ' where the variables in σ are replaced by their values under θ.

More formally

$$\begin{aligned} \langle\rangle\theta &= \langle\rangle \\ \langle i|T\rangle\theta &= \langle i\rangle; \langle T\rangle\theta \\ \langle C : \mathcal{C}|T\rangle\theta &= \langle C\rangle; \langle T\rangle\theta \\ \langle v : \mathcal{V}|T\rangle\theta &= \theta(v); \langle T\rangle\theta \end{aligned}$$

Definition 9. *The composition $\gamma(\theta)$ of two substitutions θ and γ yields the substitution $\{v\backslash\gamma(\sigma) \mid v\backslash\sigma \in \theta$ and $\langle v\rangle \neq \gamma(\sigma)\} \cup \{v\backslash\sigma \mid v\backslash\sigma \in \gamma$ and $v\backslash\tau \notin \theta$ for some $\tau\}$*

Definition 10. *A substitution γ is called an instance of a substitution θ (or θ is more general than γ) iff there is a substitution ρ s.t. $\gamma = \rho(\theta)$*

Two $\Sigma_{\mathbb{N}\mathcal{V}*}$ prefixes, σ and τ, unify iff $\exists\theta \,.\, \sigma\theta = \tau\theta$.

$\Sigma_{\mathbb{N}\mathcal{C}*}$ prefixes
In addition to our typed variables we have the special constant symbols, $\mathcal{C}$. Unlike the natural numbers, each symbol in $\mathcal{C}$ represents multiple values and unlike variables we never actually consider them being assigned a particular value. For example, the prefix $\langle 1, c\rangle$ represents all prefixes of the form $\langle 1, n\rangle$ where $n : \mathbb{N}$.

In an alternative presentation we may have chosen to replace each occurrence of our $\mathcal{C}$ symbols with variables of an appropriate type, where each variable is used only once within its scope. However, as we know that these variables are used only once, we also know that for any substitutions we make in order to unify two prefixes we could promptly forget any part that involved these variables. This is because we know that we will never use them again. In order to take advantage of this fact in our unification algorithm, we would need some mechanism that reflects this property. Instead, we decided on using our $\mathcal{C}$ symbols which are never actually assigned a value but can match any value of the correct type.

When defining unification of prefixes containing $\mathcal{C}$ symbols, we don't think in terms of a unifying substitution. Instead, we look at the sets of basic prefixes they represent and determine if they overlap. For this purpose, we use a grounding function, $\mathcal{G} : \Sigma_{\mathbb{N}\mathcal{C}*} \to \Sigma$, that takes a $\Sigma_{\mathbb{N}\mathcal{C}*}$ prefix and returns the set of basic prefixes it represents.

Definition 11. *The operator $\mathcal{G} : \Sigma_{\mathbb{N}\mathcal{C}*} \to \Sigma$ is defined as follows:*

$$\begin{aligned}
\mathcal{G}(\langle\rangle) &= \{\langle\rangle\} \\
\mathcal{G}(\langle i|T\rangle) &= \{\langle i\rangle\}; \mathcal{G}(T) \\
\mathcal{G}(\langle c|T\rangle) &= \Sigma_{\mathbb{N}1}; \mathcal{G}(T) \\
\mathcal{G}(\langle C_\bullet|T\rangle) &= \Sigma_{\mathbb{N}\bullet}; \mathcal{G}(T) \\
\mathcal{G}(\langle C_+|T\rangle) &= \Sigma_{\mathbb{N}+}; \mathcal{G}(T) \\
\mathcal{G}(\langle C_*|T\rangle) &= \Sigma_{\mathbb{N}*}; \mathcal{G}(T)
\end{aligned}$$

where the operator ; is extended over sets of prefixes as follows

$$S; T = \{s; t | s \in S \wedge t \in T\}$$

Definition 12. *A prefix, σ, is an instance of a prefix, τ, if $\sigma \in \mathcal{G}(\tau)$*

For two $\Sigma_{\mathbb{N}\mathcal{C}*}$ prefixes, σ and τ, σ and τ unify iff $\mathcal{G}(\sigma) \cap \mathcal{G}(\tau) \neq \emptyset$.
A unification of σ and τ is a $\Sigma_{\mathbb{N}\mathcal{C}*}$ prefix, π s.t. $\mathcal{G}(\pi) \subseteq \mathcal{G}(\sigma) \cap \mathcal{G}(\tau)$.

General Prefixes
As general prefixes contain both variables and $\mathcal{C}$ symbols, our definition of unification needs to combine both θ-substitution and the grounding function, $\mathcal{G}$. As $\mathcal{G}$ is only defined on prefixes that do not contain variables, we need to be able to reduce a general prefix into a $\Sigma_{\mathbb{N}\mathcal{C}*}$ prefix. For this we define the idea of a prefix mapping.

Definition 13. *A prefix mapping, $m : \mathcal{V} \to \Sigma$ is a mapping between prefix variables and basic prefixes such that*

$$\begin{aligned} m(v) &\in \Sigma_{\mathbb{N}1} \\ m(v_\bullet) &\in \Sigma_{\mathbb{N}\bullet} \\ m(v_+) &\in \Sigma_{\mathbb{N}+} \\ m(v_*) &\in \Sigma_{\mathbb{N}*} \end{aligned}$$

We apply a prefix mapping, m, to a prefix, σ by replacing each occurrence of a variable, v in σ with $m(v)$. We write the result as σ_m. This is analogous to the application of a substitution θ to σ, $\sigma\theta$.

If we are considering two general prefixes, σ and τ, for any prefix mapping, m, we are interested in the set $\mathcal{G}(\sigma_m) \cap \mathcal{G}(\tau_m)$

Two general prefixes, σ and τ, unify iff $\exists m \,.\, \mathcal{G}(\sigma_m) \cap \mathcal{G}(\tau_m) \neq \emptyset$.
π is a unification of σ and τ iff $\exists m, \theta \,.\, \mathcal{G}(\pi) \subseteq \mathcal{G}(\sigma_m) \cap \mathcal{G}(\tau_m)$.

3.2 Unification Algorithm

Below, we give the details of our unification algorithm. $\upsilon : \Sigma_* \times \Sigma_* \to \mathbb{P}\Sigma_*$ takes two general prefixes and returns a set of general prefixes that represent their unification.

If we extend $\mathcal{G}$ over sets of prefixes such that

$$\mathcal{G}(\Pi) = \bigcup_{\pi \in \Pi} \mathcal{G}(\pi)$$

then for any prefix mapping, m,

$$\mathcal{G}(\upsilon(\sigma, \tau)_m) = \mathcal{G}(\sigma_m) \cap \mathcal{G}(\tau_m)$$

For any possible combination or pairs of prefixes $(\langle h_1|\sigma\rangle, \langle h_2|\tau\rangle)$ we only provide the rule for one ordering of the pair. Given the rule for $(\langle h_1|\sigma\rangle, \langle h_2|\tau\rangle)$, the rule for $(\langle h_2|\tau\rangle, \langle h_1|\sigma\rangle)$ is self evident.

Also in this particular presentation we do not retain the implied substitution(s). Instead, any substitutions required are applied during the unification process and we only return the final unifications.

If $\upsilon(\sigma, \tau) \neq \emptyset$ we say that σ and τ unify. For convenience, we will also use υ as a predicate such that $\upsilon(\sigma, \tau)$ iff σ and τ unify.

Definition 14. *We define our unification algorithm,υ as follows:*

$$
\begin{aligned}
\upsilon(\langle\rangle,\langle\rangle) &= \{\langle\rangle\} \\
\upsilon(\langle\rangle,\langle h|\sigma\rangle) &= \upsilon(\langle\rangle,\sigma) && \textit{if } h\in\{C_\bullet,C_*\} \\
&= \upsilon(\langle\rangle,\sigma)\{h\backslash\langle\rangle\} && \textit{if } h\in V_\bullet\cup V_* \\
&= \{\} && \textit{otherwise} \\
\upsilon(\langle h|\sigma\rangle,\langle h|\tau\rangle) &= \{h\};\upsilon(\sigma,\tau)\textit{if } h\in\mathbb{N}\cup\mathcal{V} \\
\upsilon(\langle i|\sigma\rangle,\langle j|\tau\rangle) &= \{\} \\
\upsilon(\langle i|\sigma\rangle,\langle c|\tau\rangle) &= \{\langle i\rangle\};\upsilon(\sigma,\tau) \\
\upsilon(\langle i|\sigma\rangle,\langle C_\bullet|\tau\rangle) &= \upsilon(\langle i|\sigma\rangle,\tau)\cup\langle i\rangle;\upsilon(\sigma,\tau) \\
\upsilon(\langle i|\sigma\rangle,\langle C_+|\tau\rangle) &= \{\langle i\rangle\};\upsilon(\sigma,\langle C_*|\tau\rangle) \\
\upsilon(\langle i|\sigma\rangle,\langle C_*|\tau\rangle) &= \upsilon(\langle i|\sigma\rangle,\tau)\cup\langle i\rangle;\upsilon(\sigma,\langle C_*|\tau\rangle) \\
\upsilon(\langle i|\sigma\rangle,\langle v|\tau\rangle) &= \{\langle i\rangle\};\upsilon(\sigma,\tau)\{v\backslash\langle i\rangle\} \\
\upsilon(\langle i|\sigma\rangle,\langle v_\bullet|\tau\rangle) &= \upsilon(\langle i|\sigma\rangle,\tau)\{v_\bullet\backslash\langle\rangle\}\cup \\
&\quad\ \langle i\rangle;\upsilon(\sigma,\tau)\{v_\bullet\backslash\langle i\rangle\} \\
\upsilon(\langle i|\sigma\rangle,\langle v_+|\tau\rangle) &= \{\langle i\rangle\};\upsilon(\sigma,\langle u_*|\tau\rangle)\{v_+\backslash\langle i,u_*\rangle\} \\
\upsilon(\langle i|\sigma\rangle,\langle v_*|\tau\rangle) &= \upsilon(\langle i|\sigma\rangle,\tau)\{v_*\backslash\langle\rangle\}\cup \\
&\quad\ \{\langle i\rangle\};\upsilon(\sigma,\langle u_*|\tau\rangle)\{v_*\backslash\langle i,u_*\rangle\} \\
\upsilon(\langle c|\sigma\rangle,\langle c|\tau\rangle) &= \{\langle c\rangle\};\upsilon(\sigma,\tau) \\
\upsilon(\langle c|\sigma\rangle,\langle C_\bullet|\tau\rangle) &= \upsilon(\langle c|\sigma\rangle,\tau)\cup\langle c\rangle;\upsilon(\sigma,\tau) \\
\upsilon(\langle c|\sigma\rangle,\langle C_+|\tau\rangle) &= \{\langle c\rangle\};\upsilon(\sigma,\langle C_*|\tau\rangle) \\
\upsilon(\langle c|\sigma\rangle,\langle C_*|\tau\rangle) &= \upsilon(\langle c|\sigma\rangle,\tau)\cup\{\langle c\rangle\};\upsilon(\sigma,\langle C_*|\tau\rangle) \\
\upsilon(\langle c|\sigma\rangle,\langle v|\tau\rangle) &= \{\langle v\rangle\};\upsilon(\sigma,\tau) \\
\upsilon(\langle c|\sigma\rangle,\langle v_\bullet|\tau\rangle) &= \upsilon(\langle c|\sigma\rangle,\tau)\{v_\bullet\backslash\langle\rangle\}\cup\langle u\rangle;\upsilon(\sigma,\tau)\{v_\bullet\backslash\langle u\rangle\} \\
\upsilon(\langle c|\sigma\rangle,\langle v_+|\tau\rangle) &= \{\langle u\rangle\};\upsilon(\sigma,\langle u_*|\tau\rangle)\{v_+\backslash\langle u,u_*\rangle\} \\
\upsilon(\langle c|\sigma\rangle,\langle v_*|\tau\rangle) &= \upsilon(\langle c|\sigma\rangle,\tau)\{v_*\backslash\langle\rangle\}\cup\{\langle u\rangle\};\upsilon(\sigma,\langle C_*|\tau\rangle)\{v_*\backslash\langle u,u_*\rangle\} \\
\upsilon(\langle C_\bullet|\sigma\rangle,\langle C_\bullet|\tau\rangle) &= \upsilon(\sigma,\tau)\cup\upsilon(\langle c|\sigma\rangle,\tau)\cup\upsilon(\sigma,\langle c|\tau\rangle)\cup\{\langle c\rangle\};\upsilon(\sigma,\tau) \\
\upsilon(\langle C_\bullet|\sigma\rangle,\langle C_+|\tau\rangle) &= \upsilon(\sigma,\langle C_+|\tau\rangle)\cup\{\langle c\rangle\};\upsilon(\sigma,\langle C_*|\tau\rangle) \\
\upsilon(\langle C_\bullet|\sigma\rangle,\langle C_*|\tau\rangle) &= \upsilon(\sigma,\tau)\cup\upsilon(\sigma,\langle C_+\tau\rangle)\cup\upsilon(\langle c|\sigma\rangle,\tau)\cup\{\langle c\rangle\};\upsilon(\sigma,\langle C_*|\tau\rangle) \\
\upsilon(\langle C_\bullet|\sigma\rangle,\langle v|\tau\rangle) &= \upsilon(\sigma,\langle v|\tau\rangle)\cup\{\langle v\rangle\};\upsilon(\sigma,\tau) \\
\upsilon(\langle C_\bullet|\sigma\rangle,\langle v_\bullet|\tau\rangle) &= \upsilon(\sigma,\tau)\{v_\bullet\backslash\langle\rangle\}\cup\upsilon(\langle c|\sigma\rangle,\tau)\{v_\bullet\backslash\langle\rangle\}\cup \\
&\quad\ \upsilon(\sigma,\langle u|\tau\rangle)\{v_\bullet\backslash\langle u\rangle\}\cup\{\langle u\rangle\};\upsilon(\sigma,\tau)\{v_\bullet\backslash\langle u\rangle\} \\
\upsilon(\langle C_\bullet|\sigma\rangle,\langle v_+|\tau\rangle) &= \upsilon(\sigma,\langle v_+|\tau\rangle)\cup\{\langle u\rangle\};\upsilon(\sigma,\langle u_*|\tau\rangle)\{v_+\backslash\langle u,u_*\rangle\} \\
\upsilon(\langle C_\bullet|\sigma\rangle,\langle v_*|\tau\rangle) &= \upsilon(\sigma,\tau)\{v_*\backslash\langle\rangle\}\cup\upsilon(\langle c|\sigma\rangle,\tau)\{v_*\backslash\langle\rangle\}\cup \\
&\quad\ \upsilon(\sigma,\langle u_+|\tau\rangle)\{v_*\backslash\langle u_+\rangle\}\cup\{\langle u\rangle\};\upsilon(\sigma,\langle u_*|\tau\rangle)\{v_*\backslash\langle u,u_*\rangle\} \\
\upsilon(\langle C_+|\sigma\rangle,\langle C_+|\tau\rangle) &= \{\langle C_+\rangle\};\upsilon(\sigma,\tau)\cup\{\langle C_+\rangle\};\upsilon(\langle C_+|\sigma\rangle,\tau)\cup \\
&\quad\ \{\langle C_+\rangle\};\upsilon(\sigma,\langle C_+|\tau\rangle) \\
\upsilon(\langle C_+|\sigma\rangle,\langle C_*|\tau\rangle) &= \upsilon(\langle C_+|\sigma\rangle,\tau)\cup\{\langle C_+\rangle\};\upsilon(\sigma,\tau)\cup \\
&\quad\ \{\langle C_+\rangle\};\upsilon(\langle C_+|\sigma\rangle,\tau)\cup\{\langle C_+\rangle\};\upsilon(\sigma,\langle C_+|\tau\rangle) \\
\upsilon(\langle C_+|\sigma\rangle,\langle v|\tau\rangle) &= \{\langle v\rangle\};\upsilon(\langle C_*|\sigma\rangle,\tau) \\
\upsilon(\langle C_+|\sigma\rangle,\langle v_\bullet|\tau\rangle) &= \upsilon(\langle C_+|\sigma\rangle,\tau)\{v_\bullet\backslash\langle\rangle\}\cup\{\langle u\rangle\};\upsilon(\langle C_*|\sigma\rangle,\tau)\{v_\bullet\backslash\langle u\rangle\} \\
\upsilon(\langle C_+|\sigma\rangle,\langle v_+|\tau\rangle) &= \{\langle v_+\rangle\};\upsilon(\sigma,\tau)\cup\{\langle v_+\rangle\};\upsilon(\langle C_+|\sigma\rangle,\tau)\cup \\
&\quad\ \{\langle u_+\rangle\};\upsilon(\sigma,\langle w_+|\tau\rangle)\{v_+\backslash\langle u_+,v_+\rangle\} \\
\upsilon(\langle C_+|\sigma\rangle,\langle v_*|\tau\rangle) &= \upsilon(\langle C_+|\sigma\rangle,\tau)\{v_*\backslash\langle\rangle\}\cup\{\langle v_+\rangle\};\upsilon(\sigma,\tau)\cup \\
&\quad\ \{\langle v_+\rangle\};\upsilon(\langle C_+|\sigma\rangle,\tau)\cup\{\langle u_+\rangle\};\upsilon(\sigma,\langle w_+|\tau\rangle)\{v_+\backslash\langle u_+,v_+\rangle\} \\
\upsilon(\langle C_*|\sigma\rangle,\langle C_*|\tau\rangle) &= \{\langle C_*\rangle\};\upsilon(\sigma,\tau)\cup\{\langle C_*\rangle\};\upsilon(\langle C_+|\sigma\rangle,\tau)\cup \\
&\quad\ \{\langle C_*\rangle\};\upsilon(\sigma,\langle C_+|\tau\rangle)
\end{aligned}
$$

$$
\begin{aligned}
\upsilon(\langle C_*|\sigma\rangle, \langle v|\tau\rangle) &= \upsilon(\sigma, \langle v|\tau\rangle) \cup \{\langle v\rangle\}; \upsilon(\langle C_*|\sigma\rangle, \tau) \\
\upsilon(\langle C_*|\sigma\rangle, \langle v_\bullet|\tau\rangle) &= \upsilon(\sigma, \tau)\{v_\bullet\backslash\langle\rangle\} \cup \upsilon(\sigma, \langle u|\tau\rangle)\{v_\bullet\backslash\langle u\rangle\}\cup \\
&\quad \upsilon(\langle C_+|\sigma\rangle, \tau) \cup \{\langle c\rangle\}; \upsilon(\sigma, \langle C_*|\tau\rangle) \\
\upsilon(\langle C_*|\sigma\rangle, \langle v_+|\tau\rangle) &= \upsilon(\sigma, \langle v_+|\tau\rangle) \cup \{\langle u\rangle\}; \upsilon(\langle C_*|\sigma\rangle, \langle u_*|\tau\rangle)\{u_*\backslash\langle u, u_*\rangle\} \\
\upsilon(\langle C_*|\sigma\rangle, \langle v_*|\tau\rangle) &= \{\langle v_*\rangle\}; \upsilon(\sigma, \tau) \cup \{\langle v_*\rangle\}; \upsilon(\langle C_+|\sigma\rangle, \tau)\cup \\
&\quad \{\langle u_*\rangle\}; \upsilon(\sigma, \langle w_+|\tau\rangle)v_*\backslash\langle u_*, w_+\rangle \\
\upsilon(\langle v|\sigma\rangle, \langle v'|\tau\rangle) &= \{\langle v\rangle\}; \upsilon(\sigma, \tau)\{v'\backslash v\} \\
\upsilon(\langle v|\sigma\rangle, \langle v_\bullet|\tau\rangle) &= \upsilon(\langle v|\sigma\rangle, \tau)\{v_\bullet\backslash\langle\rangle\} \cup \{\langle v\rangle\}; \upsilon(\sigma, \tau)\{v_\bullet\backslash\langle v\rangle\} \\
\upsilon(\langle v|\sigma\rangle, \langle v_+|\tau\rangle) &= \{\langle v\rangle\}; \upsilon(\sigma, \langle u_*|\tau\rangle)\{v_+\backslash\langle v, u_*\rangle\} \\
\upsilon(\langle v|\sigma\rangle, \langle v_*|\tau\rangle) &= \upsilon(\langle v|\sigma\rangle, \tau)\{v_*\backslash\langle\rangle\} \cup \{\langle v\rangle\}; \upsilon(\sigma, \langle u_*|\tau\rangle)\{v_*\backslash\langle v, u_*\rangle\} \\
\upsilon(\langle v_\bullet|\sigma\rangle, \langle v'_\bullet|\tau\rangle) &= \upsilon(\sigma, \tau)\{v_\bullet\backslash\langle\rangle, v'_\bullet\backslash\langle\rangle\} \cup \upsilon(\langle u|\sigma\rangle, \tau)\{v_\bullet\backslash\langle u\rangle, v'_\bullet\backslash\langle\rangle\}\cup \\
&\quad \upsilon(\sigma, \langle u|\tau\rangle)\{v_\bullet\backslash\langle\rangle, v'_\bullet\backslash\langle u\rangle\} \cup \{\langle u\rangle\}; \upsilon(\sigma, \tau)\{v_\bullet\backslash\langle u\rangle, v'_\bullet\backslash\langle u\rangle\} \\
\upsilon(\langle v_\bullet|\sigma\rangle, \langle v_+|\tau\rangle) &= \upsilon(\sigma, \langle v_+|\tau\rangle)\{v_\bullet\backslash\langle\rangle\}\cup \\
&\quad \{\langle u\rangle\}; \upsilon(\sigma, \langle u_*|\tau\rangle)\{v_\bullet\backslash\langle u\rangle, v_+\backslash\langle u, u_*\rangle\} \\
\upsilon(\langle v_\bullet|\sigma\rangle, \langle v_*|\tau\rangle) &= \upsilon(\sigma, \tau)\{v_\bullet\backslash\langle\rangle, v_*\backslash\langle\rangle\} \cup \upsilon(\langle u|\sigma\rangle, \tau)\{v_\bullet\backslash\langle u\rangle, v_*\backslash\langle\rangle\}\cup \\
&\quad \upsilon(\sigma, \langle u_+|\tau\rangle)\{v_\bullet\backslash\langle\rangle, v_*\backslash\langle u_+\rangle\}\cup \\
&\quad \{\langle u\rangle\}; \upsilon(\sigma, \langle u_*|\tau\rangle)\{v_\bullet\backslash\langle u\rangle, v_*\backslash\langle u, u_*\rangle\} \\
\upsilon(\langle v_+|\sigma\rangle, \langle v'_+|\tau\rangle) &= \{\langle v_+\rangle\}; \upsilon(\sigma, \tau)\{v'_+\backslash\langle v_+\rangle\}\cup \\
&\quad \{\langle v'_+\rangle\}; \upsilon(\langle u_+|\sigma\rangle, \tau)\{v_+\backslash\langle v'_+, u_+\rangle\}\cup \\
&\quad \{\langle v_+\rangle\}; \upsilon(\sigma, \langle u_+|\tau\rangle)\{v'_+\backslash\langle v_+, u_+\rangle\} \\
\upsilon(\langle v_+|\sigma\rangle, \langle v_*|\tau\rangle) &= \upsilon(\langle v_+|\sigma\rangle, \tau)\{v_*\backslash\langle\rangle\} \cup \upsilon(\langle v_+|\sigma\rangle, \langle u_+|\tau\rangle)\{v_*\backslash\langle u_+\rangle\} \\
\upsilon(\langle v_*|\sigma\rangle, \langle v'_*|\tau\rangle) &= \{\langle v_*\rangle\}; \upsilon(\sigma, \tau)\{v'_*\backslash\langle v_*\rangle\}\cup \\
&\quad \{\langle v'_*\rangle\}; \upsilon(\langle u_+|\sigma\rangle, \tau)\{v_*\backslash\langle v'_*, u_+\rangle\}\cup \\
&\quad \{\langle v_*\rangle\}; \upsilon(\sigma, \langle u_+|\tau\rangle)\{v'_*\backslash\langle v_*, u_+\rangle\}
\end{aligned}
$$

where $u, u_\bullet, u_+, u_*, w, w_\bullet, w_+$ and w_* are fresh variables, i and j are distinct numbers and v and v' are distinct variables.

4 Comparison With Other Prefix Systems

4.1 Fitting's prefixes

In his prefixed tableaux system ([2]), Fitting defined a prefix as a non-empty finite sequence of positive integers. In our notation this would be $\Sigma_{\mathbb{Z}++}$. In terms of expressivity, this is the same as $\Sigma_{\mathbb{N}+}$.

Fitting doesn't actually talk about the idea of unification. Instead he refers to accessibility. The definition of accessibility varies depending on the logic under consideration. The table below describes different types of accessibility.

general	σn is accessible from σ for every n
symmetry	σ is accessible from σn for every n
reflexivity	σ is accessible from itself
transitivity	τ is accessible from σ if σ is a proper segment of τ
universal	any prefix is accessible from any prefix

We can express this form of accessibility using unification.

general	σ is accessible from τ if $\upsilon(\sigma, \tau; \langle c \rangle)$
symmetry	σ is accessible from τ if $\upsilon(\sigma; \langle c \rangle, \tau)$
reflexivity	σ is accessible from τ if $\sigma = \tau$
transitivity	σ is accessible from τ if $\upsilon(\sigma, \tau; \langle C_+ \rangle)$
universal	σ is accessible from τ

4.2 Massacci's SST

In Single Step Tableaux (SST [4]), Massacci uses the same prefix notation as Fitting, but removes the need to determine anything other than immediate successors of a particular prefix. In which case, with regards to our Σ_{XY} prefixes, SST can be presented in the same way as Fitting's prefixes except we only ever need to apply the general accessibility rule.

4.3 leanK

The Free Variable SST (leanK) system ([1]) enriches Fitting style prefixes with variables that capture all the worlds that are immediately accessible from another. Beckert and Goré present two forms of variable which they refer to as universal and free. A variable, x, is universal if an instantiation of x on one branch need not affect the value of x on another. Universal variables may be instantiated multiple times. Free variables may only be instantiated once.

It is clear that universal variables relate to our $\mathcal{C}$ symbols and free variables to the variables in $\mathcal{V}$. As we are only interested in immediate successors, we need only consider the $\mathcal{C}$ symbol, c and variables of the type V. In this case we can see that their notation is equivalent to $\Sigma_{\mathbb{N}\{c\}V+}$.

Beckert and Goré do use unification in their work. If we translate their prefixes into $\Sigma_{\mathbb{N}\{c\}V+}$ prefixes through a sensible mapping of universal variables to c and free variables to members of V, our unification algorithm captures their requirements. The converse is not the case, however. leanK prefixes do not have the same expressivity as general prefixes. For example, the prefix $\langle 1, C_* \rangle$ represents an infinite set of leanK prefixes $\{\langle 1 \rangle, \langle 1, u_1 \rangle, \langle 1, u_1, u_2 \rangle, ...\}$, where each u_n is a universal variable.

4.4 □KE

The prefix system described in this paper was inspired by the work of Pitt and Cunningham in their □KE system ([8]). Although the syntax has been changed somewhat and the underlying semantics have been expanded, the prefixes used in □KE are essentially the same as general ($\Sigma_{\mathbb{N}\mathcal{V}\mathcal{C}*}$) prefixes

4.5 KEM

It is KEM ([3]) that we have most difficulty relating our work to. KEM's philosophy is that whilst the process of encoding modal formulas into prefixed ones is straightforward, it is at the point of unification that we need to deal with the semantics of the logic in question. This is an opposing point of view to how we wish to deal with unification[1]. That isn't to say that there isn't a correspondence between the two systems, as one could take a KEM unification problem and translate the prefixes into general prefixes taking into account the current modal semantics. However, this would be a cumbersome process and of questionable benefit.

5 Properties of Σ_{XY} unification

5.1 Termination

In [7], Otten and Kreitz, describe a class of strings that satisfy certain restrictions. They call these strings, T-strings[2]. Two strings s and t have the T-string property iff

1. no character is repeated either in s or in t, and
2. equal characters occur only at the beginning of s and t.

Although, up until now we have placed no such restriction on a prefixes, we find that in practice prefixes generated by our modal to prefix translation procedure ([10]) have a similar property[3]. Because of the use of our $\mathcal{C}$ symbols, we can actually have the same symbol (character) being repeated. However, we can simply revise the T-string property to reflect prefixes.

Definition 15. *Let σ and τ be two prefixes. σ and τ have the* T-string property *iff,*

1. $\forall_{1 \leq i,j \leq |\sigma|} \,.\, i \neq j \rightarrow (\sigma(i) \neq \sigma(j)$ *or* $\sigma(i) \in \mathcal{C})$ *and*
   $\forall_{1 \leq i,j \leq |\tau|} \,.\, i \neq j \rightarrow (\tau(i) \neq \tau(j)$ *or* $\tau(i) \in \mathcal{C})$, *and*
2. $\exists_{0 \leq j \leq min(|\sigma|,|\tau|)} \,.\, (\, \forall_{1 \leq i \leq j} \,,\, \sigma(i) = \tau(i)$ *and*
   $\forall_{j \leq i \leq |\sigma| \;\&\; j \leq k \leq |\tau|} \,.\, (\sigma(i) \neq \tau(k)$ *or* $\sigma(i) \in \mathcal{C}))$

Theorem 1. *For any two T-prefixes σ and τ, Σ_{XY} unification will terminate on σ and τ.*

Proof. For any two prefixes, there is only one rule that can be applied at a particular level.

[1] KEM has more in common with Otten and Kreitz's T-string unification [7]
[2] Ohlbach([5]) uses a similar idea which he calls Prefix-Stability
[3] This translation procedure was inspired by the prefixed tableaux expansions in [8]

In the case that neither σ or τ contain variables then we can see that after each application of our rules for υ our subgoal(s) are simpler than our previous goal. Eventually, either of the first two rules will apply and we terminate.

In the case that we do have to deal with variables, we need to take into account the properties of a T-prefixes. As we know that no variable will appear more than once in a prefix and that if the same variable appears in both prefixes that it will only appear as part of a common initial prefix, no substitutions made during unification will affect the structure of a subgoal. Therefore, once again we can show that our subgoal(s) will always be simpler than our previous goal.

5.2 Soundness

We define soundness for our most important classes of prefixes. Although the definition of soundness for Σ_* is applicable to all sub-classes, the definitions of soundness for $\Sigma_{\mathbb{N}\mathcal{V}*}$ and $\Sigma_{\mathbb{N}\mathcal{C}*}$ are clearer and aid understanding.

$\boldsymbol{\Sigma_{\mathbb{N}\mathcal{V}*}}$ prefixes
The unification algorithm described in 3.2 does not give us the unifiers of two prefixes directly. Instead, for each unification that is produced there is an *implied* unifier that was required to make that unification. When necessary we will write $(\pi, \theta) \in \upsilon(\sigma, \tau)$ if $\pi \in \upsilon(\sigma, \tau)$ and θ was implicitly constructed during π's discovery. We use the term *implied* to refer to any unifier that is implicit in a particular unification process.

Our unification algorithm is sound with respect to $\Sigma_{\mathbb{N}\mathcal{V}*}$ prefixes if for all σ and τ, if $\upsilon(\sigma, \tau)$ implies θ then $\sigma\theta = \tau\theta$

$\boldsymbol{\Sigma_{\mathbb{N}\mathcal{C}*}}$ prefixes
Our unification algorithm is sound with respect to $\Sigma_{\mathbb{N}\mathcal{C}*}$ prefixes if for all σ and τ, if $\pi \in \upsilon(\sigma, \tau)$ then $\mathcal{G}(\pi) \subseteq \mathcal{G}(\sigma) \cap \mathcal{G}(\tau)$

If we extend $\mathcal{G}$ to apply on sets of prefixes such that

$$\mathcal{G}(\Pi) = \bigcup_{\pi \in \Pi} \mathcal{G}(\pi)$$

we can restate the above as:

Our unification algorithm is sound with respect to $\Sigma_{\mathbb{N}\mathcal{C}*}$ prefixes if for all σ and τ, $\mathcal{G}(\upsilon(\sigma, \tau)) \subseteq \mathcal{G}(\sigma) \cap \mathcal{G}(\tau)$

$\boldsymbol{\Sigma_*}$ prefixes
υ is sound with respect to Σ_* prefixes if for all σ and τ, if $(\pi, \theta) \in \upsilon(\sigma, \tau)$ then there exists a prefix mapping, m s.t $\mathcal{G}(\pi_m) \subseteq \mathcal{G}(\sigma\theta_m) \cap \mathcal{G}(\tau\theta_m) \subseteq \mathcal{G}(\sigma_m) \cap \mathcal{G}(\tau_m)$

5.3 Completeness

$\Sigma_{\mathbb{N}\mathcal{V}*}$ prefixes
υ is complete with respect to $\Sigma_{\mathbb{N}\mathcal{V}*}$ if for all σ,τ and θ, if $\sigma\theta = \tau\theta$ then there exists a substitution, θ' s.t. $\upsilon(\sigma, \tau)$ implies θ' and θ is an instance of θ'.

$\Sigma_{\mathbb{N}\mathcal{C}*}$ prefixes
υ is complete with respect to $\Sigma_{\mathbb{N}\mathcal{C}*}$ if for all σ and τ, $\mathcal{G}(\sigma) \cap \mathcal{G}(\tau) \subseteq \mathcal{G}(\upsilon(\sigma, \tau))$

Σ_* prefixes
υ is complete with respect to Σ_* prefixes if for all σ and τ, if $(\pi, \theta) \in \upsilon(\sigma, \tau)$ then for all prefix mappings, m, $\mathcal{G}(\sigma_m) \cap \mathcal{G}(\tau_m) \subseteq \mathcal{G}(\sigma\theta_m) \cap \mathcal{G}(\tau\theta_m) \subseteq \mathcal{G}(\pi_m)$

Theorem 2. *υ is sound and complete w.r.t. $\Sigma_{\mathbb{N}\mathcal{C}*}$ prefixes. i.e.* $\mathcal{G}(\upsilon(\sigma, \tau)) = \mathcal{G}(\sigma) \cap \mathcal{G}(\tau)$

Proof. As we have already noted, each step of our unification introduces simpler subgoals. Our proof is a straightforward induction over the *length* of a prefix pair s.t. $(\sigma, \tau) < (\pi, \psi)$ iff $(|\sigma| < |\pi|$ and $|\tau| \leq |\psi|)$ or $(|\sigma| \leq |\pi|$ and $|\tau| < |\psi|)$. Our rules satisfy the property that if our subgoals satisfy the induction hypothesis $\mathcal{G}(\upsilon(\sigma, \tau)) = \mathcal{G}(\sigma) \cap \mathcal{G}(\tau)$ then so does our goal.

6 Discussion

The problem of unifying two prefixes is essentially a word unification problem with a single equation and additional constraints. Although, there has a been significant work in solving general word equation problems (e.g. Makanin, Schulz, Gei...), it is generally accepted that the problem is too hard to be solved in practical applications. Instead, we choose to restrict the problem to a useful but tractable subset. The prefix notation presented in this paper was initially proposed as one aspect of a formalisation of $\Box$KE. As work progressed it became clear that its ability to encode accessibility relations was strong enough to allow for a straightforward extension to clausal resolution that could be applied to prefixed formulas. This prefixed resolution system then became the basis of a 'modal Prolog' ([9]).

There are a number of directions that future work could take. For instance, we are yet to determine the complexity of the unification process when applied to different classes of prefix. Theoretically, the use of $\mathcal{C}$ symbols significantly reduces the number of possible unifications of two particular prefixes when compared to two equivalent prefixes containing variables. However, whether this reduction is reflected in the worst case or just in the average case is unclear. As it stands the unification algorithm described in 3.2 does not reflect this potential reduction of solutions. It doesn't always find the simplest set of unifications and can find the same unification multiple times. e.g. The unification of the two prefixes $\sigma = \langle C_\bullet, C_\bullet \rangle$ and $\tau = \langle C_\bullet, C_\bullet \rangle$ finds the set of unifications $\{\langle\rangle, \langle c \rangle, \langle cc \rangle\}$ and it finds the unification $\langle c \rangle$ 4 times. Obviously the set $\{\langle C_\bullet, C_\bullet \rangle\}$ would be a preferable solution.

To further confuse matters, there are properties of the $\mathcal{C}$ symbols that have been observed but which need more consideration. For example, the prefixes $\langle C_\bullet, C_+ \rangle$, $\langle c, C_* \rangle$, and $\langle C_+ \rangle$ all represent the same set of basic prefixes. It would appear that given three prefixes, σ, σ' and τ, if σ and σ' are *equivalent* and $|\sigma| < |\sigma'|$ the process of unifying σ and τ is simpler than unifying σ' and τ.

Let us briefly demonstrate the impact simplifications could make. Consider a prefix, σ, which is simply a chain of n C_+ symbols. If we look at our unification algorithm it is clear that any attempt to unify σ with another prefix, τ, is going to have an exponential search space with respect to n. However, we can demonstrate that σ is equivalent to the prefix $\sigma'; \langle C_+ \rangle$, where σ' is a chain of $n-1$ c symbols. As we can see, the search space implied by a c is linear. If we ignore the complexity implied by the structure of τ we have essentially reduced the search space from $O(c^n)$ to $O(c) = O(1)$

It is clear that a better understanding of these issues would result in a more efficient unification algorithm.

In addition to issues of efficiency we may also wish to increase our expressivity. As it stands we do not have the ability to capture the essence of a multi-modal logic. The semantics of our unification algorithm assumes that we only have one accessibility relation. In order to deal with multiple modalities, we would need each element of a prefix to be paired with the relation it encodes.

It is our hope that work in such areas will result in an efficient and useful logic programming language for (multi)modal logics.

References

1. B Beckert and R Goré. Free variable tableaux for propositional modal logics. In *TABLEAUX-97*, volume 1227 of *LNCS*, pages 91–106. Springer, 1997.
2. M Fitting. *Proof Methods For Modal And Intuitionistic Logics.* Reidel, 1983.
3. Governatori. Labelled tableaux for multi-modal logics. In *TABLEAUX-95*, volume 918 of *LNCS*, pages 79–94. Springer, 1995.
4. F Massacci. Strongly analytic tableaux for normal modal logics. In A Bundy, editor, *CADE-12*, volume 814 of *LNCS*, pages 723–737. Springer, 1994.
5. Hans Jürgen Ohlbach. A resolution calculus for modal logics. In E. Lusk and R. Overbeek, editors, *Proceedings 9th Int. Conf. on Automated Deduction, CADE'88, Argonne, IL, USA, 23–26 May 1988*, volume 310, pages 500–516. Springer-Verlag, Berlin, 1988.
6. H.J. Ohlbach, A. Nonnengart, M. de Rijke, and D. Gabbay. Encoding two-valued nonclassical logics in classical logic. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume II, chapter 21, pages 1403–1486. Elsevier Science, 2001.
7. Jens Otten and Christoph Kreitz. T-string unification: Unifying prefixes in non-classical proof methods. In *Analytic Tableaux and Related Methods*, pages 244–260, 1996.
8. 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.
9. A. Williams and J. Cunningham. Motivation for a resolution system for a class of prefixed formulas. In *Proceedings of 10th Workshop of Automated Reasoning*, 2003.
10. A. Williams and J. Cunningham. A resolution system for a class of prefixed formulas. In *Proceedings of 4th International Workshop on First order Theorem Proving*, 2003. Submitted.