

Tableaux-Based Prover for Typed Hybrid Multimodal Logic (System Description)

Dean Rosenzweig and Davor Runje

University of Zagreb

Abstract

Some potential application domains of hybrid multimodal logic suggest typing of states and imposing type-constraints on modalities. In [7] we have investigated the logical/computational cost of such an extension with a very simple system of types and concluded that there is none: both expressivity and complexity are preserved by the extension.

Rewrite rules coping with state equality and state succession have been a major obstacle for efficient implementation of tableaux systems for hybrid modal logic. We handle state equality, state succession and valuation on the metalevel instead, constructing an explicit model-to-be in the process. Handling of types on the metalevel comes naturally in the presence of an explicit model, and it can considerably prune the search space.

We present a prototype implementation of a tableaux-based theorem prover built along these lines. The prover handles type-free hybrid modal logic and its subsystems as well.

Introduction

In some potential application domains for hybrid multimodal logic relations to be formalized come naturally typed: they relate only objects of some given sorts. While for a simple type system it is not difficult to express such constraints within the machinery of HML, say with propositional variables and axioms, this can be tedious for automatized theorem-provers. Enriching the logic itself with types, i.e. attaching types to frames, could considerably prune the search space, if implemented the right way.

With this kind of motivation, in [7] we have enriched the hybrid multimodal logic with the world's simplest type system: a finite set of pairwise disjoint types. The typed logic, THML, is briefly summarized in section 1 below, and explained in more detail in [7]. It preserves the expressive power and

complexity of HML, see [7]. Tableaux rules for THML are briefly summarized in section 2.

Here we present a tableaux-based prover for THML, intended to exploit the search-pruning potential of types. Spawning many branches, most of which will close due to type conflicts, did not seem the right way to exploit it: type constraints should be used on the metalevel, in order to spawn as few branches as necessary in the first place. The same, of course, holds for other constraints expressed by near atomic formulae and their negations.

Such considerations have led us to explicit representation of the model-to-be of a tableau branch, explained in section 3: we directly represent a typed Kripke structure, together with constraints (disequalities, banned edges and absence of propositional variables). The near atomic formulae and their negations are handled by direct manipulation of the model-to-be, and only the logical connectives and modal operators are treated in a more traditional tableaux fashion.

This gives us a strong feel of drifting towards model construction, rather than theorem proving. Any tableau system can be viewed as a model construction method, but it is rather implicit: what is constructed is a description of a model. We construct the model explicitly.

While our prover is, at this stage, only a running prototype which still needs extensive testing, comparing and streamlining, our first experiments seem rather encouraging: the prototype handles proofs containing several thousands branches easily. Of course, by setting the set of types to a singleton, it can be used also for type-free HML and its subsystems.

1 Typed hybrid multimodal logic

In this section we introduce a simple typing scheme: every state in a model has a unique type assigned to it, from a finite set of pairwise disjoint types. On the language level, types are atomic formulae treated in the same way as nominals and propositional variables.

Definition 1.1 (Typed hybrid multimodal languages) *Let TYPE be a nonempty finite set disjoint from PROP, MOD and NOM. The elements of TYPE are called types. We define a typed hybrid multimodal language (over PROP, NOM, MOD and TYPE) to be:*

$$WFF := i \mid p \mid t \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \varphi \rightarrow \psi \mid \langle \pi \rangle \varphi \mid [\pi] \varphi \mid @_i \varphi$$

where $i \in NOM$, $p \in PROP$, $\pi \in MOD$, $t \in TYPE$ and $\varphi, \psi \in WFF$.

Definition 1.2 (Typed hybrid models, satisfaction, and validity)

A typed frame is a triple $(W, T, \{R_\pi \mid \pi \in MOD\})$ where W is a set of states, T is a function assigning a type to each state, and each R_π is a binary relation over W . A typed hybrid model is a tuple $(W, T, \{R_\pi \mid \pi \in MOD\}, V)$ where $(W, T, \{R_\pi \mid \pi \in MOD\})$ is a typed frame and V is a standard hybrid

valuation: a function from $PROP \cup NOM$ to $Pow(W)$ such that $V(i)$ is a singleton set for each $i \in NOM$.

We interpret a typed hybrid language over a typed hybrid model by adding the following clause

$$\mathcal{M}, w \Vdash t \quad \text{iff} \quad T(w) = t$$

to the usual Kripke satisfaction definition:

$$\begin{array}{lll} \mathcal{M}, w \Vdash a & \text{iff} & w \in V(a), \text{ where } a \in NOM \cup PROP \\ \mathcal{M}, w \Vdash \neg\varphi & \text{iff} & \mathcal{M}, w \not\Vdash \varphi \\ \mathcal{M}, w \Vdash \varphi \wedge \psi & \text{iff} & \mathcal{M}, w \Vdash \varphi \text{ and } \mathcal{M}, w \Vdash \psi \\ \mathcal{M}, w \Vdash \varphi \vee \psi & \text{iff} & \mathcal{M}, w \Vdash \varphi \text{ or } \mathcal{M}, w \Vdash \psi \\ \mathcal{M}, w \Vdash \varphi \rightarrow \psi & \text{iff} & \mathcal{M}, w \not\Vdash \varphi \text{ or } \mathcal{M}, w \Vdash \psi \\ \mathcal{M}, w \Vdash \langle \pi \rangle \varphi & \text{iff} & \exists w' \in W ((w, w') \in R_\pi \text{ and } \mathcal{M}, w' \Vdash \varphi) \\ \mathcal{M}, w \Vdash [\pi] \varphi & \text{iff} & \forall w' \in W ((w, w') \in R_\pi \Rightarrow \mathcal{M}, w' \Vdash \varphi) \\ \mathcal{M}, w \Vdash @_i \varphi & \text{iff} & \mathcal{M}, w' \Vdash \varphi, \text{ where } w' \text{ is the denotation of } i. \end{array}$$

Global satisfaction in a model, validity on a frame and on a class of frames are defined in the usual way.

Remark 1.3 We could define a typed hybrid model by extending the domain of valuation V to include elements of $TYPE$ in the same way nominals are handled by standard hybrid models, with the following semantic restrictions on assignment, reflecting the fact that each state has a unique type assigned to it:

- if $t_1 \neq t_2$ then $V(t_1) \cap V(t_2) = \emptyset$,
- $\bigcup_{t \in TYPE} V(t) = W$.

We are interested in the validity problem on a class of frames defined by a set of formulae for our modelling purposes. We wish to use types to restrict relations $\{R_\pi \mid \pi \in MOD\}$, and for that reason we must attach type information to frames.

The primary motivation for introducing types into the language was modelling of relations with restricted types of domain and codomain. Let π be a modality and let types of a domain of a relation R_π induced by π be $Dom(\pi) \subseteq TYPE$, and let types of a codomain be $Cod(\pi) \subseteq TYPE$. We can define a class of typed frames \mathfrak{F} , where relation R_π induced by π has that property, by the following pure formula:

$$@_i \langle \pi \rangle j \rightarrow \bigwedge_{t \in TYPE \setminus Dom(\pi)} \neg @_i t \wedge \bigwedge_{t \in TYPE \setminus Cod(\pi)} \neg @_j t.$$

For a given set of modality constraints, we define the set of pure formulæ defining the class of frames respecting those constraints as:

$$\left\{ @_i \langle \pi \rangle j \rightarrow \bigwedge_{t \in TYPE \setminus Dom(\pi)} \neg @_i t \wedge \bigwedge_{t \in TYPE \setminus Cod(\pi)} \neg @_j t \mid \pi \in MOD \right\}.$$

The (unexciting) logical properties of THML are investigated in [7]. THML is sound and strongly complete wrt typed hybrid models, and, since a straightforward translation to HML is possible, it doesn't increase its expressive power.

Since types serve only to prune the proof-search space, the PSPACE lower and upper bound of [4] remain preserved; with slight adaptation their proof works for THML, see [7] for more detail.

2 Tableaux for THML

We extend the rules of the type-free tableaux with the additional three rules:

$$\frac{[i \text{ on a branch}] \quad @_i \bigvee_{t \in TYPE} t \quad [\text{Type}] \quad \frac{@_i t}{\neg @_i t_1} \quad [\text{@}_i t] \quad \begin{array}{c} \vdots \\ \{t_1, \dots, t_n\} = TYPE \setminus \{t\} \\ \neg @_i t_n \end{array}}{[\text{Cons}] \quad \frac{@_i \langle \pi \rangle j}{\begin{array}{c} \neg @_i t_1 \\ \vdots \\ \{t_1, \dots, t_n\} = TYPE \setminus Dom(\pi) \\ \neg @_i t_n \\ \neg @_j s_1 \\ \vdots \\ \{s_1, \dots, s_k\} = TYPE \setminus Cod(\pi) \\ \neg @_j s_k \end{array}}}$$

Rule **Type** expresses the fact that every state has a type assigned to it, while rule $@_i t$ forces this assignment to be well defined: an assigned type must be unique. Rule **Cons** is an optional rule needed only if we have constraints on relation R_π induced by modality π .

Soundness and strong completeness of the typed tableaux is proven in [7].

3 States of hybrid multimodal tableaux

Detailed discussion of hybrid multimodal logic and its tableaux can be found in [6], [5] and [3]. We rely on notation from [6].

While tableau-provers can in general be seen as model construction tools, this is rather implicit: a *description* of a model is constructed. We have chosen to construct a model-to-be explicitly, as follows.

Our (conceptual) representation of a type-free tableau branch is a tuple:

$$(\Gamma, (Equal^+, Edges^+, Val^+), (Equal^-, Edges^-, Val^-))$$

where Γ is a set of formulæ, $(Equal^+, Edges^+, Val^+)$ is an explicit representation of the model-to-be, and $(Equal^-, Edges^-, Val^-)$ is a representation of constraints on the model. Let $NODE$ be a set representing the nodes of the model, and MOD , NOM and $PROP$ be sets from the language definition [6], then:

$$\begin{aligned} Equal^+ &: NOM \rightarrow NODE, \\ Edges^+ &: MOD \rightarrow (NODE \rightarrow Pow(NODE)), \\ Val^+ &: NODE \rightarrow Pow(PROP), \\ Equal^- &: NODE \rightarrow Pow(NODE), \\ Edges^- &: MOD \rightarrow (NODE \rightarrow Pow(NODE)), \text{ and} \\ Val^- &: NODE \rightarrow Pow(PROP). \end{aligned}$$

We view the model-to-be as a constrained hybrid Kripke structure. The Kripke structure (W, R_π, V) is represented by $Equal^+$, $Edges^+$, Val^+ in the following sense:

$$\begin{aligned} W &= Cod(Equal^+), \\ R_\pi &= \{(m, n) \mid n \in Edges^+(\pi)(m)\}, \pi \in MOD, \\ V &= Val^+. \end{aligned}$$

We say that a nominal i names a node I iff $Equal^+(i) = I$. Equality of nominals is expressed by mapping them to the same node. The maps $Equal^-$, $Edges^-$ and Val^- represent constraints, expressed by negated near atomic formulæ, banning some nominal equalities, edges and valuations.

All formulæ on a branch are viewed modulo equivalence induced by $Equal^+$. This representation is equivalent to the representation of branch as a set of formulæ, with all rewrite rules iteratively applied. Instead of applying all rewrite rules dealing with state equality and state succession, we update our representation of the model built so far. When an equality formula $@_i j$ is resolved, the model is updated in two steps:

1. Nodes I and J , named by nominals i and j , are substituted with a fresh node K in codomains of all maps with codomain $NODE$ or $Pow(NODE)$.
2. Nodes I and J are removed, and node K is added, to the set of nodes $NODE$. Every map Map with domain $NODE$ is updated as follows:

$$Map(K) := Map(I) \cup Map(J).$$

Resolving of other near atomic satisfaction statements updates the model or constraints in an obvious way. For instance, resolving of the formula $@_i \langle \pi \rangle j$

updates the map $Edges^+(\pi)$ as follows:

$$Edges^+(\pi)(I) := Edges^+(\pi)(I) \cup \{J\}$$

where nodes I and J are named by nominals i and j .

If constraints are in conflict with the model in a branch, then the branch gets closed. The conflicts are easy to express and verify on the representation:

- (i) node w of a model is absurd: $\exists w. w \in Equal^-(w)$,
- (ii) the model has an absurd edge: $\exists w. Edges^+(\pi)(w) \cap Edges^-(\pi)(w) \neq \emptyset$,
- (iii) the model has an absurd valuation: $\exists w. Val^+(w) \cap Val^-(w) \neq \emptyset$.

Actual implementation of maps is distributed over the structure of tableaux, partially shared among branches with shared ancestors. Each branch node has only incremental information on each map. Every query not understood by a branch node is sent to its parent.

Optionally, we can check if a branch is closed early when adding a formula to Γ , by checking existence of its direct contradiction on the branch (of course, modulo $Equal^+$). We can also look for the direct contradictions every time we update $Equal^+$. Flipping this option doesn't influence either soundness or completeness properties of the tableaux.

Another option is to explore targets of existing edges as well, when the prover creates a new node processing any instance of $@_i\langle\pi\rangle\varphi$, in the hope of finding as small a model as possible. With tautologies, having this option on certainly doesn't help. Flipping the explore-old-nodes option on and off almost feels like switching between model construction and theorem proving.

4 Adding types

The way of handling types in our representation of a branch comes naturally: every node in the model has a set of possible types assigned to it.

$$Type : NODE \rightarrow Pow(TYPE)$$

Initially, we set $Type(w) := TYPE$ for each node $w \in NODE$.

Thus our model-to-be represents a family of typed Kripke structures, satisfying $T(n) \in Type(n)$ for any $n \in W$.

Resolving formulæ $@_i\langle\pi\rangle j$, $@_i t$ and $\neg@_i t$ restricts the set of possible types of nodes named by i and j with modality constraints as follows:

$$\frac{@_i t}{Type(I) := Type(I) \cap \{t\}} \quad [@_i t] \qquad \frac{\neg@_i t}{Type(I) := Type(I) \setminus \{t\}} \quad [\neg@_i t]$$

$$\frac{\textcircled{!}_i \langle \pi \rangle j}{\text{Type}(I) := \text{Type}(I) \cap \text{Dom}(\pi)} \quad [\text{Cons}]$$

$$\text{Type}(J) := \text{Type}(J) \cap \text{Cod}(\pi)$$

where $t \in \text{TYPE}$, I and J are nodes named by nominal i and j , and $\text{Dom}(\pi)$ and $\text{Cod}(\pi)$ are type restrictions on modality π .

On the explicit model-to-be, the rule for updating the map Type when resolving an equality formula $\textcircled{!}_i j$ is different from the rule for other maps:

$$\text{Type}(K) := \text{Type}(I) \cap \text{Type}(J)$$

where I and J are nodes named by nominal i and j .

To conflict cases (i)–(iii) we add:

(iv) the model has a type conflict: $\exists w. \text{Type}(w) = \emptyset$.

In order to process type free hybrid modal formulæ, we set TYPE to a singleton. This makes all type related operations noops.

5 Implementation

Our running prototype is implemented in the language AsmL, under development at Microsoft Research, primarily as a specification language [1]. The choice has been motivated by several reasons:

- (i) Most of AsmL has a very precise mathematical semantics in terms of the basic notion of first order structure, which considerably facilitates reasoning about implementations.
- (ii) AsmL supports handling of sets and maps, with a rich set of high level operations, and notation close to the one usual in mathematics, which considerably reduces the programming effort.
- (iii) AsmL is integrated into the .NET platform, which considerably facilitates bidirectional communication with any software on the platform, such as parsing and testing tools, graph visualizers, application domain models etc.
- (iv) Last not least, we like it.

The code has been written with clarity as the primary design goal, and no optimization has been done so far (except for the decision on branch representation). While we yet have no systematic performance measurements, we have been pleasantly surprised to see the prototype handle proofs containing several thousands branches easily.

The full code of the current prototype is freely downloadable from [2]. It will run on any .NET port which can execute AsmL runtimes [1], what currently, to our knowledge, means only Win32 with .NET framework, but

this might change. We hope to keep [2] updated with current information.

References

- [1] *AsmL web site.*
URL <http://research.microsoft.com/fse/AsmL>
- [2] *Tableaux for typed hybrid modal logic.*
URL <http://www.fsb.hr/matematika/index.php?ulaz=hylo>
- [3] Areces, C. and P. Blackburn, *Bringing them all together*, Journal of Logic and Computation **11** (2001), pp. 657–669.
- [4] Areces, C., P. Blackburn and M. Marx, *A road-map on complexity for hybrid logics*, in: J. Flum and M. Rodriguez-Artalejo, editors, *the 8th Annual Conference of the EACSL*, Madrid, 1999.
- [5] Blackburn, P., *Internalizing labelled deduction*, Journal of Logic and Computation **10(1)** (2000), pp. 137 – 168.
- [6] Blackburn, P., *Representation, reasoning, and relational structures: a hybrid logic manifesto*, Logic Journal of the IGPL **8** (2000), pp. 339–625.
- [7] Rosenzweig, D., D. Runje and N. Slani, *Typed hybrid multimodal logic*, manuscript (2003).
URL <http://www.fsb.hr/~drosenzw/hylo/typedhylo.ps>