Model Checking Hybrid Logics (With an Application to Semistructured Data)

Massimo Franceschet [a,b,1] Maarten de Rijke [b,2]

[a] *Department of Sciences, University of Chieti-Pescara, Italy*

[b] *Informatics Institute, University of Amsterdam, The Netherlands*

Abstract

We investigate the complexity of the model checking problem for hybrid logics. We provide model checker algorithms for various hybrid fragments and we prove PSPACE-completeness for hybrid fragments including binders. We complement and motivate our complexity results with an application of model checking in hybrid logic to the problems of query and constraint evaluation for semistructured data.

1 Introduction

In *model checking* [19] we are given a formal model and a property and we have to check whether the model satisfies the property. The model is a labelled graph, sometimes called Kripke structure, and the property is a formula in some logical language. We search the graph in order to *check* whether the formula is true in the *model*. As a technique, model checking has very strong links to (at least) two areas in computer science: verification and databases. In the first half of this paper, we focus on model checking algorithms for so-called hybrid logics; in the second half, we go on to show their relevance for reasoning about semistructured data.

Email addresses: `francesc@science.uva.nl` (Massimo Franceschet), `mdr@science.uva.nl` (Maarten de Rijke).

[1] Supported by a grant from the Netherlands Organization for Scientific Research (NWO) under project number 612.000.207.

[2] Supported by grants from the Netherlands Organization for Scientific Research (NWO) under project numbers 365-20-005, 612.069.006, 612.000.106, 220-80-001, 612.000.207, and 612.066.302.

Modal and temporal logics have been successfully used as specification languages in the model checking task [20]; they are algorithmically well-behaved and mathematically natural fragments of classical logics. However, something crucial is missing in propositional modal and temporal logics: they lack mechanisms for naming states, for accessing states by names, and for dynamically creating new names for states. In particular, traditional modal and temporal logics are able to express properties that satisfy the *tree model property*, that is, properties that are satisfiable if, and only if, they are satisfiable in a tree-like model. Are there extensions of modal and temporal logics violating the tree model property that are still computationally tractable? This is where *hybrid logics* come in. They allow us to refer to states in a truly modal framework, mixing features from first-order logic and modal logic, whence the name *hybrid* logic [15]. In addition to ordinary propositional variables, hybrid languages provide a type of atomic formulas called *nominals*. Syntactically, nominals behave like propositional variables, but they have an important semantic property: nominals are true at exactly one state in any model.

Nominals are only the first ingredient that sets hybrid languages apart from traditional modal-like languages. Hybrid languages may also contain the *at operator* $@_i$ which gives direct access to the unique state named by i: $@_i p$ holds if, and only if, p holds at the state named by i. Moreover, hybrid languages may be extended with the *downarrow binder* $\downarrow x$ that assigns the variable name x to the current state of evaluation. The operator @ combines naturally with $\downarrow$: $\downarrow$ stores the current state of evaluation and @ enables us to *retrieve* the information stored by shifting the point of evaluation in the model. While $\downarrow x$ stores the current state in x, the binder $\Downarrow x$ stores the 'label' of the current state in x, that is, the set of propositions holding at the current state. Finally, the *existential binder* $\exists x$ binds the variable name x to some state in the model.

Model checking for hybrid languages has hardly been explored so far. In this paper we address this gap. Our approach is incremental: on top of well-known model checking results for Propositional Temporal Logic and Converse Propositional Dynamic Logic, we investigate the model checking problem for these languages extended with the hybrid machinery as well as with the universal modality $\mathbf{A}$. It turns out that the addition of nominals, the @ operator, and the universal modality $\mathbf{A}$ does not increase the complexity of the model checker. In contrast, an arbitrary use of hybrid binders in formulas is computationally dangerous. The model checking problem for any hybrid logic that freely mixes the hybrid binder $\downarrow$ or $\Downarrow$ with temporal operators is PSPACE-complete, while $\exists$ is hard even without temporal operators. However, the model checker runs in exponential time with respect to the *nesting degree* of the binders in the formula. This means that we can still check in polynomial time long formulas, as long as the nesting degree of the hybrid binders on the formula is bound. We summarize our complexity results in Table 1, where k is the length of the formula, n and m are the number of nodes and the number of edges of the graph

Table 1
Complexity of model checking for hybrid logics.

Language	Complexity	Language	Complexity
$\mathrm{HL}(@, \mathbf{F}, \mathbf{P}, \mathbf{A})$	$\mathbf{O}(k \cdot (n+m))$	$\mathrm{HL}_r(\downarrow, @, \mathbf{F}, \mathbf{A})$	$\mathbf{O}(k \cdot (n+m) \cdot n^r)$
$\mathrm{HL}(@, \mathbf{U}, \mathbf{S}, \mathbf{A})$	$\mathbf{O}(k \cdot n \cdot m)$	$\mathrm{HDL}_r(\downarrow, @, \mathbf{A})$	$\mathbf{O}(k \cdot (n+m) \cdot n^r)$
$\mathrm{HDL}(@, \mathbf{A})$	$\mathbf{O}(k \cdot (n+m))$	$\mathrm{HL}(\exists)$	PSPACE-complete
$\mathrm{HL}(\downarrow, @)$	$\mathbf{O}(k \cdot n)$	$\mathrm{HL}(\exists, @, \mathbf{F}, \mathbf{A})$	PSPACE-complete
$\mathrm{HL}(\downarrow, \mathbf{F})$	PSPACE-complete	$\mathrm{HL}_r(\exists, @, \mathbf{F}, \mathbf{A})$	$\mathbf{O}(k \cdot (n+m) \cdot n^{r+1})$
$\mathrm{HL}(\downarrow, \mathbf{A})$	PSPACE-complete	$\mathrm{HDL}_r(\exists, @, \mathbf{A})$	$\mathbf{O}(k \cdot (n+m) \cdot n^{r+1})$
$\mathrm{HL}(\downarrow, @, \mathbf{F}, \mathbf{A})$	PSPACE-complete		

structure, respectively, and r is the nesting degree of hybrid binders. Moreover, **F** is the Future temporal operator, **P** is Past, **U** is Until and **S** is Since. Finally, languages of the form $\mathrm{HL}(\cdot)$ are hybrid extensions of Propositional Temporal Logic, while languages of the form $\mathrm{HDL}(\cdot)$ are hybrid extensions of Converse Propositional Dynamic Logic. Notice that PSPACE-complete problems are hard with respect to expression (or formula, or query) complexity, which is the complexity of model checking if we only the length of the formula as a parameter. If data complexity (the complexity of model checking if we only the size of the model as a parameter) is taken into account, all the model checking problems summarized in the table can be solved in polynomial time.

In the second part of the paper we illustrate a general methodological point: since hybrid languages provide very natural modeling facilities, understanding the computational and algorithmic properties of hybrid languages is of great potential value. The complexity-theoretic results obtained in this paper are valuable results about hybrid logic in their own right, but we believe they get additional value because of the fact that hybrid languages provide such natural modeling facilities, which makes the formal results of this paper applicable in a fairly direct way. To back up these claims we apply model checking for hybrid logics to the problems of query and constraint evaluation for *semistructured data*: data with some structure but without a regular schema. We discuss a hierarchy of query languages for semistructured data corresponding to fragments of the language Lorel [3], the query language in the Lore system [32], which was designed for managing semistructured data. The languages that we discuss offer regular expressions to navigate the query graph at arbitrary depths, as well as the possibility of comparing object identities and object values. We embed those query languages into fragments of hybrid logics with different expressivity and establish a close relationship between the query processing problem for semistructured data and the global model checking problem for hybrid logics. Moreover, we describe languages to specify path constraints for semistructured data, including inclusion, in-

verse and functional path constraints. Path constraints generalize relational integrity constraints for semistructured databases and they are useful to provide a loose schema to the otherwise unstructured database. Once again, we provide an embedding into hybrid logic, this time of the constraint language, and we underline the close connection between path constraint evaluation for semistructured data and model checking for hybrid logics.

The paper is organized as follows. In Section 2 we discuss related work. Section 3 introduces hybrid logic. In Section 4 we provide model checkers for different hybrid languages, analyze their computational complexity, and prove PSPACE-completeness of the model checking problem for hybrid fragments allowing binders. Readers mostly interested in the relation between hybrid logic and semistructured data, can skip Section 4 and return to it for details of the results used in Section 5, where we describe the application of model checking hybrid logic to semistructured data. We conclude the paper and outline future work in Section 6.

2 Related Work

Hybrid logic was invented by Arthur Prior, the inventor of tense logic. The germs of the idea seem to have emerged in the 1950s, but the first detailed account is [36]. Prior called nominals world propositions and worked with rich hybrid languages including quantifiers $\forall$ and $\exists$. The next big step was taken by Robert Bull, Prior's student, in [16]. Bull introduced a three-sorted hybrid language (propositional variables, state nominals and *path nominals*) and proved a completeness result for this logic. Path nominals name branches in tree-like models of time by being true at all and only the points of the branch. There were no further papers on the subject till the 1980s, when hybrid logic was reinvented by a group of Bulgarian logicians (Passy, Tinchev, Gargov, and Goranko). The locus classicus of this work is Passy and Tinchev's [35]; they initiated the study of binder-free systems. During the 1990s, the emphasis has been on understanding the hybrid hierarchy in more detail. Goranko introduced the $\downarrow$ binder [26], Blackburn and Seligman examined the interrelationships between a number of binders [14]. Characterizations with respect to first-order correspondence theory, interpolation properties, and computational complexity (of the satisfiability problem) for hybrid modal logics have been studied in [11]; recent contributions completing the picture are in [38]. The complexity of the satisfiability problem for hybrid temporal logics with respect to different classes of frames has been investigated in [9,10,25]. As for implementations, a resolution-based theorem prover for hybrid logic with @ and $\downarrow$ has been implemented [12]. For a comprehensive entry point to the field see the hybrid logic home page [30].

Since the mid-1990s there has been a lot of work on the interface of computational logic and semistructured data, making use of a wide variety of logical tools and techniques. E.g., [17] use simulations and morphisms, [4] concentrate on regular expressions, and [18] make the connection with description logic. The relation between model checking and query processing has been extensively explored for *structured* data; see, e.g., [28]. The relation between model checking and query processing for *semistructured* data goes back at least to [6], where it was formulated in terms of suitable modal-like logics. Quintarelli [37] embeds a fragment of the graphical query language G-Log into CTL, and she sketches a mapping for subsets of other semistructured query languages, like Lorel, GraphLog and UnQL. It is worth noticing that the fragments considered in [37] do not allow queries with joins. De Alfaro [8] proposes the use of model checking for detecting errors in the structure and connectivity of web pages. Miklau and Suciu [33] and Gottlob et al. [27] sketch an embedding of the forward looking fragment of XPath into CTL. Finally, Marx [31] used PDL-like logics in order to extend the XPath core language to a language that is expressively complete with respect to first-order logic on finite trees.

As for the relation between model checking and path constraints evaluation for semistructured data, Alechina et al. [7] embed forward and backward path constraints into Converse Propositional Dynamic Logic. Calvanese et al. [18] use description logics, and Afanasiev et al. [5] turn to CTL and provide experimental results of the "query evaluation as model checking" perspective.

3 Hybrid Logics

Temporal logics [23] (TL, for short) may be viewed as fragments of classical logics [13]. They extend propositional logic by adding the well-known temporal operators future $\mathbf{F}$, past $\mathbf{P}$, until $\mathbf{U}$ and since $\mathbf{S}$. Let $\mathsf{PROP} = \{p, q, \ldots\}$ be a set of propositional variables. The syntax of temporal logic is as follows:

$$\phi := \top \mid p \mid \neg\phi \mid \phi \wedge \phi \mid \mathbf{F}\phi \mid \mathbf{P}\phi \mid \phi\mathbf{U}\phi \mid \phi\mathbf{S}\phi.$$

We adopt the usual Boolean shorthands. The dual of $\mathbf{P}$ is $\mathbf{H}\alpha = \neg\mathbf{P}\neg\alpha$, and the dual of $\mathbf{F}$ is $\mathbf{G}\alpha = \neg\mathbf{F}\neg\alpha$.

Temporal logic is interpreted over *Kripke structures* of the form $\langle M, R, V\rangle$, where M is a set of states (or worlds, points, nodes), R is a binary relation on M called the accessibility (or reachability) relation, and V is a valuation function from PROP to the powerset of M. We assume no specific structure of time (linear, branching, ...). Let $\mathcal{M}$ be a Kripke structure and $m \in M$. The semantics of temporal logic is given in Figure 1

Sometimes, TL also includes the transitive closure operators $\mathbf{F}^+$, $\mathbf{P}^+$, $\mathbf{U}^+$ and

Fig. 1. Semantics for temporal logic.

$$
\begin{aligned}
\mathcal{M}, m &\models \top \\
\mathcal{M}, m &\models p \quad \text{iff} \quad m \in V(p),\ p \in \mathsf{PROP} \\
\mathcal{M}, m &\models \neg\phi \quad \text{iff} \quad \mathcal{M}, m \not\models \phi \\
\mathcal{M}, m &\models \phi \wedge \psi \quad \text{iff} \quad \mathcal{M}, m \models \phi \text{ and } \mathcal{M}, m \models \psi \\
\mathcal{M}, m &\models \mathbf{F}\phi \quad \text{iff} \quad \exists m'\ (Rmm' \wedge \mathcal{M}, m' \models \phi) \\
\mathcal{M}, m &\models \mathbf{P}\phi \quad \text{iff} \quad \exists m'\ (Rm'm \wedge \mathcal{M}, m' \models \phi) \\
\mathcal{M}, m &\models \psi\mathbf{U}\phi \quad \text{iff} \quad \exists m'\ (Rmm' \wedge \mathcal{M}, m' \models \phi \wedge \\
&\qquad\qquad\qquad\quad \forall m''\ (Rmm'' \wedge Rm''m' \rightarrow \mathcal{M}, m'' \models \psi)) \\
\mathcal{M}, m &\models \psi\mathbf{S}\phi \quad \text{iff} \quad \exists m'\ (Rm'm \wedge \mathcal{M}, m' \models \phi \wedge \\
&\qquad\qquad\qquad\quad \forall m''\ (Rm'm'' \wedge Rm''m \rightarrow \mathcal{M}, m'' \models \psi))
\end{aligned}
$$

$\mathbf{S}^+$, interpreted over the transitive closure R^+ of the accessibility relation, as well as the universal modality $\mathbf{A}$. The semantics of the universal modality is as follows: $\mathcal{M}, m \models \mathbf{A}\phi$ iff for all m it holds that $\mathcal{M}, m \models \phi$. The dual of the universal modality is the existential modality $\mathbf{E}\alpha = \neg\mathbf{A}\neg\alpha$.

Hybrid logic (HL, for short) extends temporal logic with devices for naming states and accessing states by names. Let $\mathsf{NOM} = \{i, j, \ldots\}$ and $\mathsf{WVAR} = \{x, y, \ldots\}$ be sets of nominals and state variables, respectively. HL's syntax is:

$$\phi := \mathrm{TL} \mid i \mid x \mid @_t\phi \mid \downarrow x.\phi \mid \exists x.\phi,$$

with $i \in \mathsf{NOM}$, $x \in \mathsf{WVAR}$, $t \in \mathsf{NOM} \cup \mathsf{WVAR}$. The operators in $\{@, \downarrow, \exists\}$ are called *hybrid operators*. We call $\mathsf{WSYM} = \mathsf{NOM} \cup \mathsf{WVAR}$ the set of *state symbols*, $\mathsf{ALET} = \mathsf{PROP} \cup \mathsf{NOM}$ the set of *atomic letters*, and $\mathsf{ATOM} = \mathsf{PROP} \cup \mathsf{NOM} \cup \mathsf{WVAR}$ the set of *atoms*. We use $x = y$ for $@_x y$ and $x \neq y$ for $@_x \neg y$. For simplicity, we omit parenthesis after the $\downarrow$. For instance, in $\downarrow x.p \wedge @_x q$, the variable x used in $@_x q$ is bound by $\downarrow x$. Hence, it should be read as $\downarrow x.(p \wedge @_x q)$.

Hybrid logic is interpreted over *hybrid Kripke structures*, i.e., Kripke structures $\langle M, R, V\rangle$ where the valuation function V assigns singleton subsets of M to nominals $i \in \mathsf{NOM}$. To give meaning to the formulas, we also need the notion of *assignment*. An assignment g is a mapping $g : \mathsf{WVAR} \rightarrow M$. Given an assignment g, we define g^x_m by $g^x_m(x) = m$ and $g^x_m(y) = g(y)$ for $x \neq y$. For any atom a, let $[V, g](a) = \{g(a)\}$ if a is a state variable, and $V(a)$ otherwise.

The semantics of hybrid logic is given in Figure 2, where $\mathcal{M} = \langle M, R, V\rangle$ is a hybrid Kripke structure, $m \in M$, and g is an assignment; the semantics for Boolean and temporal operators is as for temporal logic. In words, the at operator $@_t$ shifts evaluation to the state named by t, where t is a nominal or

Fig. 2. Semantics for hybrid logic.

$$
\begin{array}{rll}
\mathcal{M}, g, m \models a & \text{iff} & m \in [V, g](a), \;\; a \in \mathsf{ATOM} \\
\mathcal{M}, g, m \models @_t \phi & \text{iff} & \mathcal{M}, g, m' \models \phi, \text{ where } [V, g](t) = \{m'\}, \;\; t \in \mathsf{WSYM} \\
\mathcal{M}, g, m \models \downarrow x.\phi & \text{iff} & \mathcal{M}, g^x_m, m \models \phi \\
\mathcal{M}, g, m \models \exists x.\phi & \text{iff} & \text{there is } m' \in M \text{ such that } \mathcal{M}, g^x_{m'}, m \models \phi
\end{array}
$$

a variable. The downarrow binder $\downarrow x$ binds the state variable x to the *current* state (where evaluation is being performed), while the existential binder $\exists x$ binds the state variable x to *some* state in the model; $\downarrow$ and $\exists$ do not shift evaluation away from the current state. We use $\mathrm{HL}(O_1, \ldots, O_n)$ to denote the hybrid language with hybrid and temporal operators $O_1, \ldots, O_n$.

The until operator $\mathbf{U}$ can be written in hybrid logic as follows: $\alpha \mathbf{U} \beta = \downarrow x.\mathbf{F}(\beta \wedge \mathbf{H}(\mathbf{P}x \rightarrow \alpha))$, and analoguously for the since operator $\mathbf{S}$. Moreover, the past operator is $\mathbf{P}\alpha = \downarrow x.\exists y.@_y(\mathbf{F}x \wedge \alpha)$. Finally, the downarrow binder $\downarrow$ is a particular case of the existential binder: $\downarrow x.\alpha = \exists x.(x \wedge \alpha)$, while $\exists$ can be simulated by $\downarrow$ and $\mathbf{E}$ as follows: $\exists x.\alpha = \downarrow y.\mathbf{E}\downarrow x.\mathbf{E}(y \wedge \alpha)$.

In addition to the hybrid languages listed so far, we consider hybrid *dynamic* languages; these will prove to be especially useful in Section 5. We consider a hybridization of converse propositional dynamic logic (CPDL) [29]. CPDL adds two operators to propositional logic, namely $\langle e \rangle \alpha$ and $\langle e \rangle^{-1} \alpha$, where e is a regular expression on a set of labels Σ and α is a CPDL formula. CPDL is interpreted over *Labelled Transitions Systems* (LTSs), Kripke structures in which both the nodes and the edges are labelled. The edges are labelled with symbols in Σ. Each regular expression e on the set of edge labels Σ identifies a binary relation R_e on the set of states. The relation R_e is recursively defined in terms of the structure of e. More precisely, R_l contains all the edges labelled with l, $R_{e_1.e_2} = R_{e_1} \circ R_{e_2}$, $R_{e_1+e_2} = R_{e_1} \cup R_{e_2}$, and $R_{e^*} = (R_e)^*$. A state s is *reachable* from a state r through the regular expression e if $(r, s) \in R_e$. Given a LTS $\mathcal{M}$ and a state s in $\mathcal{M}$, we have that $\langle e \rangle \alpha$ is true in $\mathcal{M}$ at s if there exists a state s' reachable from s trough e such that α is true in $\mathcal{M}$ at s'. Moreover, $\langle e \rangle^{-1} \alpha$ is true in $\mathcal{M}$ at s if there exists a state s' such that s is reachable from s' through e and α is true in $\mathcal{M}$ at s'.

Hybrid dynamic logic is the *hybridization* of CPDL. We write $\mathrm{HDL}(O_1, \ldots, O_n)$ to denote the extension of CPDL with nominals, hybrid operators and possibly the universal modality in $O_1, \ldots, O_n$. For instance, $\mathrm{HDL}(@, \downarrow, \mathbf{A})$ is CPDL with nominals, $@$, $\downarrow$ and $\mathbf{A}$ operators.

We now introduce a new hybrid binder $\Downarrow$. We have separated the introduction of this binder because it is usually not included in hybrid languages. However, $\Downarrow$ will come in handy in Section 5. Intuitively, while $\downarrow$ stores the current state

into a variable, $\Downarrow$ stores the *label* (or *value*) of the current state, that is, the set of propositions that hold at the current state. Let $\mathsf{WVAR}' = \{v, w, \ldots\}$ be a set of variables such that WVAR and WVAR' are disjoint sets. The new variables in WVAR' serve as containers for sets of propositions. We add to the hybrid language the formulas $\Downarrow v.\alpha$, v, and $v = w$, and the new shortcut $v \neq w$ for $\neg(v = w)$, where $v, w \in \mathsf{WVAR}'$, with the following meaning. Let $\mathcal{M} = \langle M, R, V\rangle$ be a hybrid Kripke structure, $m \in M$, and g an assignment extended to WVAR'; that is, g is a mapping that associates each variable in WVAR to a state in M and each variable in WVAR' to a subset of PROP. We denote by V^{-1} the function from M to the powerset of PROP such that $p \in V^{-1}(m)$ iff $m \in V(p)$. Then, $\mathcal{M}, g, m \models \Downarrow v.\alpha$ iff $\mathcal{M}, g^v_{V^{-1}(m)}, m \models \alpha$. Moreover, $\mathcal{M}, g, m \models v$ iff $g(v) = V^{-1}(m)$ and $\mathcal{M}, g, m \models v = w$ iff $g(v) = g(w)$. In words, the binder $\Downarrow v$ binds the variable v to the label of the current state, while $v = w$ compares the labels stored in v and w.

To put our results on model checking in perspective we briefly recall the complexity/decidability results for satisfiability for the logics we consider. The basic hybrid logic with just nominals and the @ operator is PSPACE-complete, not harder than modal logic. However, as soon as either the past operator, or the until operator, or the universal modality is added, the satisfiability problem becomes EXPTIME-complete. Finally, if the $\downarrow$ binder is added, decidability of the satisfiability problem is lost.

4 Model Checking for Hybrid Logics

We now investigate the complexity of the global model checking problem for various hybrid languages. A hybrid Kripke structure $\mathcal{M} = \langle M, R, V\rangle$ is *finite* if M is finite. The *global model checking problem* for hybrid logic is: Given a finite hybrid Kripke structure $\mathcal{M}$, an assignment g, and a hybrid formula ϕ, is there a state $m \in M$ such that $\mathcal{M}, g, m \models \phi$? We distinguish between *expression complexity*, i.e., the complexity of the model checking problem when the complexity parameter is the length of the formula only, and *data complexity*, i.e., the complexity of the model checking problem when the complexity parameter is the size of the model only.

4.1 Model Checkers

We provide global model checkers for different hybrid logics and we analyze their worst-case behavior. We start by describing a model checker called `MCLITE` for the language HL($@, \mathbf{F}, \mathbf{P}, \mathbf{U}, \mathbf{S}, \mathbf{A}$). It receives a hybrid model $\mathcal{M} = \langle M, R, V\rangle$, an assignment g, and a hybrid formula ϕ in the consid-

ered language and, after termination, every state in the model is labelled with the subformulas of ϕ that hold at that state. The algorithm uses a *bottom-up strategy*: it examines the subformulas of ϕ in increasing order of length, until ϕ itself has been checked.

We need some auxiliary notation. Let R be an accessibility relation; then R^- is the inverse of R: R^-vu if, and only if, Ruv. For $n \geq 1$, let $R^n(w)$ be the set of states that are reachable from w in n R-steps, and $R^{-n}(w)$ be the set of states that are reachable from w in n R^--steps. The states belonging to $R^1(w)$ are *successors* of w, while those belonging to $R^{-1}(w)$ are *predecessors* of w. Given a model $\mathcal{M} = \langle M, R, V\rangle$, we denote by $\mathcal{M}^-$ the model $\langle M, R^-, V\rangle$. The *length* of a formula ϕ, denoted by $|\phi|$, is the number of operators (Boolean, temporal and hybrid) of ϕ plus the number of atoms (propositions, nominals and variables) of ϕ. Let $sub(\phi)$ be the set of subformulas of ϕ. Notice that $|sub(\phi)| = O(|\phi|)$.

The model checker `MCLITE` updates a table L of size $|\phi| \times |M|$ whose elements are bits. Initially, $L(\alpha, w) = 1$ if, and only if, α is an atomic letter in $sub(\phi)$ such that $w \in V(\alpha)$. When `MCLITE` terminates, $L(\alpha, w) = 1$ if, and only if, $\mathcal{M}, g, w \models \alpha$ for every $\alpha \in sub(\phi)$. Given $\alpha \in sub(\phi)$ and $w \in M$, we denote by $L(\alpha)$ the set of states $v \in M$ such that $L(\alpha, v) = 1$ and by $L(w)$ the set of formulas $\beta \in sub(\phi)$ such that $L(\beta, w) = 1$. `MCLITE` uses subroutines $\mathtt{MC}_{\mathbf{F}}$, $\mathtt{MC}_{\mathbf{U}}$, $\mathtt{MC}_{\mathbf{A}}$, and $\mathtt{MC}_{@}$ in order to check subformulas of the form $\mathbf{F}\alpha$, $\alpha\mathbf{U}\beta$, $\mathbf{A}\alpha$, and $@_t\alpha$, respectively. The pseudocode for these procedures is in Figure 3.

Proposition 4.1 (Correctness of subroutines) *Let $\mathcal{M} = \langle M, R, V\rangle$ be a hybrid model, g an assignment, α and β hybrid formulas, and t a state symbol. Let L be a table such that, for every $w \in M$, $L(\alpha, w) = 1$ (respectively, $L(\beta, w) = 1$) if, and only if, $\mathcal{M}, w \models \alpha$ (respectively, $\mathcal{M}, w \models \beta$). Then,*

(1) after termination of $\mathtt{MC}_{\mathbf{F}}(\mathcal{M}, g, \alpha)$, we have that, for every state $w \in M$, $L(\mathbf{F}\alpha, w) = 1$ if, and only if, $\mathcal{M}, g, w \models \mathbf{F}\alpha$;
(2) after termination of $\mathtt{MC}_{\mathbf{A}}(\mathcal{M}, g, \alpha)$, we have that, for every state $w \in M$, $L(\mathbf{A}\alpha, w) = 1$ if, and only if, $\mathcal{M}, g, w \models \mathbf{A}\alpha$;
(3) after termination of $\mathtt{MC}_{@}(\mathcal{M}, g, t, \alpha)$, we have that, for every $w \in M$, $L(@_t\alpha, w) = 1$ if, and only if, $\mathcal{M}, g, w \models @_t\alpha$; and
(4) after termination of $\mathtt{MC}_{\mathbf{U}}(\mathcal{M}, g, \alpha, \beta)$, we have that, for every $w \in M$, $L(\alpha\mathbf{U}\beta, w) = 1$ if, and only if, $\mathcal{M}, g, w \models \alpha\mathbf{U}\beta$.

Proof. The proofs for cases 1, 2, and 3 are easy. We only show the more involved case 4. The procedure $\mathtt{MC}_{\mathbf{U}}$ works as follows. First, all nodes are set to unmarked. Then, for each w that is labelled with β, the first inner for loop marks all the nodes u such that there exists a node v which is *not* labelled with α and Ruv and Rvw. The second inner for loop labels with $\alpha\mathbf{U}\beta$ the remaining unmarked nodes. The set of states labelled with $\alpha\mathbf{U}\beta$ grows monotonically as

Fig. 3. MCLITE subprocedures.

```
Procedure MC_F(M, g, α)
for w ∈ L(α) do
    for v ∈ R^{-1}(w) do
        L(Fα, w) ← 1
    end for
end for
Procedure MC_A(M, g, α)
if L(α) = M then
  for v ∈ M do
      L(Aα, v) ← 1
  end for
end if
Procedure MC_@(M, g, t, α)
let {w} = [V, g](t)
if L(α, w) = 1 then
  for v ∈ M do
      L(@_t α, v) ← 1
  end for
end if
```

```
Procedure MC_U(M, g, α, β)
for w ∈ M do
    unmark(w)
end for
for w ∈ L(β) do
    for v ∈ R^{-1}(w) do
        if L(α, w) = 0 then
          for u ∈ R^{-1}(v) do
              mark(u)
          end for
        end if
    end for
    for v ∈ R^{-1}(w) ∪ R^{-2}(w) do
        if marked(v) then
          unmark(v)
        else
          L(αUβ, v) ← 1
        end if
    end for
end for
```

computation proceeds. Indeed, a node that is not labelled with $\alpha \mathbf{U} \beta$ during the iteration for some w may be labelled with $\alpha \mathbf{U} \beta$ during a later iteration for some w'. Let $w \in L(\beta)$. We claim that:

Claim 1 *After termination of the main for loop dedicated to w in the procedure $\mathtt{MC}_{\mathbf{U}}$, for every $v \in R^{-1}(w) \cup R^{-2}(w)$, we have that v is labelled with $\alpha \mathbf{U} \beta$ if, and only if, every successor of v that is a predecessor of w is labelled with α.*

It follows that, after termination of the procedure $\mathtt{MC}_{\mathbf{U}}$, for every $v \in M$, v is labelled with $\alpha \mathbf{U} \beta$ if, and only if, every successor of v that is a predecessor of *some* $w \in L(\beta)$ is labelled with α, which means that $\mathcal{M}, v \models \alpha \mathbf{U} \beta$. To prove the left to right direction of the claim, suppose that v is labelled with $\alpha \mathbf{U} \beta$. Then, v is unmarked before the second inner for loop. We infer that every successor of v that is a predecessor of w is labelled with α. Indeed, suppose there exists a successor z of v such that z precedes w and z is not labelled with α. Because of the first inner for loop, the predecessors of z are marked. Since v is a predecessor of z, v is marked as well, which contradicts the fact that v is unmarked. For the right to left direction, suppose every successor of v that is a predecessor of w is labelled with α. Then, after the first inner for loop, v is unmarked. Indeed, suppose v is marked. Then, there exists some successor z of v that is a predecessor of w and is not labelled with α. This contradicts the fact that every successor of v that is a predecessor of w is labelled with α. Hence, v is unmarked before entering the second inner for loop and, hence, v

is labelled with $\alpha \mathbf{U} \beta$ at the end of it. $\square$

What is the complexity of the subroutines? Let $\mathcal{M} = \langle M, R, V \rangle$ be a finite hybrid model, with $n = |M|$ and $m = |R|$. The procedure $\mathtt{MC}_{\mathbf{F}}$ runs in $O(n+m)$, and both $\mathtt{MC}_{@}$ and $\mathtt{MC}_{\mathbf{A}}$ run in $O(n)$. As for $\mathtt{MC}_{\mathbf{U}}$, for every $w \in L(\beta)$, the procedure pays two backward visits to the states reachable in 2 steps. Each visit costs $O(m)$. As there are $O(n)$ states in $L(\beta)$, $\mathtt{MC}_{\mathbf{U}}$ runs in $O(n \cdot m)$ time.

A model checker MCLITE for HL$(@, \mathbf{F}, \mathbf{P}, \mathbf{U}, \mathbf{S}, \mathbf{A})$ can easily be programmed by taking advantage of subroutines $\mathtt{MC}_{\mathbf{F}}$, $\mathtt{MC}_{\mathbf{A}}$, $\mathtt{MC}_{@}$, and $\mathtt{MC}_{\mathbf{U}}$. MCLITE works bottom-up checking all subformulas of the input formula in increasing length order. When a formula starting with one of @, **F**, **U**, **A** needs to be checked, the corresponding procedure is invoked. Past temporal operators **P** and **S** are handled by feeding the subroutines $\mathtt{MC}_{\mathbf{F}}$ and $\mathtt{MC}_{\mathbf{U}}$, respectively, with the reversed model $\mathcal{M}^{-}$. Finally, Boolean connectives are treated as usual. The correctness of MCLITE follows from the correctness of its subroutines (Proposition 4.1) and the semantics of the operators **P** and **S**.

Theorem 4.2 (Correctness of MCLITE) *Let $\mathcal{M} = \langle M, R, V \rangle$ be a hybrid model, g an assignment, and ϕ a formula in* HL$(@, \mathbf{F}, \mathbf{P}, \mathbf{U}, \mathbf{S}, \mathbf{A})$. *Then, after termination of* $\mathtt{MCLITE}(\mathcal{M}, g, \phi)$, *we have that, for every $w \in M$ and for every $\alpha \in sub(\phi)$, it holds that $L(\alpha, w) = 1$ if, and only if, $\mathcal{M}, g, w \models \alpha$.*

Theorem 4.3 (Complexity of MCLITE) *Let $\mathcal{M} = \langle M, R, V \rangle$ be a hybrid model such that $n = |M|$ and $m = |R|$. Let g be an assignment, and ϕ a hybrid formula of length k. Then,*

- *if ϕ belongs to* HL$(@, \mathbf{F}, \mathbf{P}, \mathbf{U}, \mathbf{S}, \mathbf{A})$, *then the model checker* $\mathtt{MCLITE}(\mathcal{M}, g, \phi)$ *terminates in time $O(k \cdot n \cdot m)$;*
- *if ϕ belongs to* HL$(@, \mathbf{F}, \mathbf{P}, \mathbf{A})$, *then the model checker* $\mathtt{MCLITE}(\mathcal{M}, g, \phi)$ *terminates in time $O(k \cdot (n + m))$; and*
- *if ϕ belongs to* HL$(@, \mathbf{A})$, *then the model checker* $\mathtt{MCLITE}(\mathcal{M}, g, \phi)$ *terminates in time $O(k \cdot n)$.*

Proof. There are $|sub(\phi)| = |\phi| = k$ subformulas to check. The complexity of each check depends on the form of the subformula ψ. If ψ is atomic, it is checked in constant time. If its main operator is Boolean, @, or **A**, then it is checked in $O(n)$. If ψ's main operator is **F** or **P**, then it is checked in $O(n+m)$. Finally, if its main operator is **U** or **S**, then ψ is checked in $O(n \cdot m)$. $\square$

We can extend MCLITE to cope with transitive closure operators $\mathbf{F}^{+}$ and $\mathbf{U}^{+}$, as well as with their past counterparts, without increasing the asymptotic complexity. The idea is to replace the visit to the predecessors of w with a backward depth-first visit of the nodes that can reach w. As an alternative, one may first compute the transitive closure of the accessibility relation, and

Fig. 4. MCFULL subprocedures.

```
Procedure Check_F(M, g, α)
MCFULL(M, g, α)
MC_F(M, g, α)

Procedure Check_↓(M, g, x, α)
for w ∈ M do
    g(x) ← w
    MCFULL(M, g, α)
    if w ∈ L(α) then
       L(↓x.α, w) ← 1
    end if
    Clear(L, x)
end for
```

```
Procedure Check_∃(M, g, x, α)
for v ∈ M do
    g(x) ← v
    MCFULL(M, g, α)
    for w ∈ M do
        if w ∈ L(α) then
           L(∃x.α, w) ← 1
        end if
    end for
    Clear(L, x)
end for
```

then use the model checker MCLITE on the transitive model.

We now move to hybrid languages with binders. The efficient bottom-up strategy used in MCLITE does not work for such languages. Why? Consider the formula $\mathbf{G}{\downarrow}x.\mathbf{F}x$. It says that every successor of the current point is reflexive. If we try to check this formula in a bottom-up fashion, we initially have to check the subformula x. However, at this stage, we do not have enough information to check x, and hence we cannot label the states of the model with x. Instead, we can proceed as follows: we first check ${\downarrow}x.\mathbf{F}x$ with a procedure yet to be developed, then we check $\mathbf{G}{\downarrow}x.\mathbf{F}x$, that is $\neg\mathbf{F}\neg{\downarrow}x.\mathbf{F}x$, taking advantage of the subprocedure $\mathtt{MC}_{\mathbf{F}}$ of MCLITE. In order to check ${\downarrow}x.\mathbf{F}x$, for each state w, we first assign w to x, and then check $\mathbf{F}x$ at w under the new assignment for x. The latter can again be done using $\mathtt{MC}_{\mathbf{F}}$.

The sketched strategy, which combines top-down and bottom-up reasoning, has been implemented in a recursive model checker MCFULL for the full hybrid language HL(HO ∪ TO), where HO = $\{@, {\downarrow}, \exists\}$, and TO = $\{\mathbf{F}, \mathbf{P}, \mathbf{U}, \mathbf{S}, \mathbf{A}\}$. The auxiliary procedures $\mathtt{Check}_*$, with $* \in$ HO ∪ TO, handle the cases of subformulas with main operator $*$. Whenever possible, these procedures reuse the subprocedures $\mathtt{MC}_*$, with $* \in \{@\} \cup$ TO, of MCLITE. They use the new subroutine $\mathtt{Clear}(L, x)$ to reset all the values of $L(\alpha)$, for any α containing the variable x free (we assume that different binders use different variables). Boolean operators are treated as usual. In Figure 4 we show the pseudocode for $\mathtt{Check}_{\mathbf{F}}$, $\mathtt{Check}_{\downarrow}$, and $\mathtt{Check}_{\exists}$. The procedures for the other operators are similar to $\mathtt{Check}_{\mathbf{F}}$.

Theorem 4.4 (Correctness of MCFULL) *Let $\mathcal{M} = \langle M, R, V\rangle$ be a hybrid model, g an assignment, and ϕ a hybrid formula. Then, after termination of* MCFULL($\mathcal{M}$, g, ϕ), *we have that:*

- *for every $w \in M$, $L(\phi, w) = 1$ if, and only if, $\mathcal{M}, g, w \models \phi$; and*

- *for every $w \in M$ and every sentence $\alpha \in sub(\phi)$, $L(\alpha, w) = 1$ if, and only if, $\mathcal{M}, g, w \models \alpha$.*

Theorem 4.5 (Complexity of MCFULL) *Let $\mathcal{M} = \langle M, R, V \rangle$ be a hybrid model such that $n = |M|$ and $m = |R|$, and g an assignment. Let ϕ be a hybrid formula in $\mathrm{HL}(\exists, \downarrow, S)$, and let $r_\downarrow$ and $r_\exists$ be the nesting degree of $\downarrow$ and $\exists$, respectively, in ϕ. Let $\mathcal{C}_S$ be the model checking complexity for $\mathrm{HL}(S)$. Then, $\mathtt{MCFULL}(\mathcal{M}, g, \phi)$ terminates in time $O(\mathcal{C}_S \cdot n^{r_\downarrow + r_\exists + 1})$ if $r_\exists > 0$, and in time $O(\mathcal{C}_S \cdot n^{r_\downarrow})$ if $r_\exists = 0$. Moreover, the procedure uses polynomial space.*

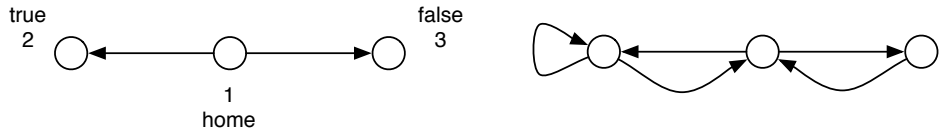
Proof. We have that (1) the procedure $\mathtt{Check}_*$ runs in time $\mathcal{C}_\alpha + \mathcal{C}_*$, where $\mathcal{C}_*$ is the cost of $\mathtt{MC}_*$ and $\mathcal{C}_\alpha$ is the cost to check α; (2) the procedure $\mathtt{Check}_\downarrow$ runs in time $n \cdot \mathcal{C}_\alpha$; and (3) the procedure $\mathtt{Check}_\exists$ runs in time $n \cdot (\mathcal{C}_\alpha + n)$. Thus, the overall worst-case time complexity is $O(\mathcal{C}_S \cdot n^{r_\downarrow + r_\exists + 1})$ if $r_\exists > 0$, and it is $O(\mathcal{C}_S \cdot n^{r_\downarrow})$ if $r_\exists = 0$. Since the height of the recursion stack for MCFULL is at most $|\phi|$, we have that MCFULL uses polynomial space. □

MCFULL can easily be extended to cope with formulas involving $\Downarrow$ and the comparison of state labels. A procedure similar to $\mathtt{Check}_\downarrow$ can be used to check formulas of the form $\Downarrow v.\alpha$. The only difference is that we have to bind the variable v to the *label* of the current state, and not to the current state itself. The complexity remains the same. Moreover, the formula v, for $v \in \mathsf{WVAR}'$, can be verified by comparing the label of the current state and the label stored in v, while the formula $v = w$, for $v, w \in \mathsf{WVAR}'$, can be checked by comparing the labels stored in v and w.

Furthermore, MCFULL can be viewed as a general model checker for the hybridization of *any* temporal logic. Given a temporal logic $\mathbf{T}$, the hybridization of $\mathbf{T}$ is the hybrid logic obtained from the language of $\mathbf{T}$ by adding the hybrid machinery. Suppose that, for each temporal operator $\mathbf{O}$ of arity k in $\mathbf{T}$, we can exploit a procedure $\mathtt{Check}_\mathbf{O}$ that, given a model and k formulas $\alpha_1, \ldots, \alpha_k$, labels each state m of the model with the formula $\mathbf{O}(\alpha_1, \ldots, \alpha_k)$ if, and only if, the formula is true at m. A model checker for the hybridization of $\mathbf{T}$ can be synthesized from these model checking procedures following the example of MCFULL. For instance, a model checker for $\mathrm{HDL}(@, \downarrow, \Downarrow, \mathbf{A})$ can be programmed by taking advantage of a CPDL model checker. Model checking for CPDL can be done in linear time with respect to the length of the formula and the size of the model [21]. Hence, we have:

Theorem 4.6 *Model checking for $\mathrm{HDL}(@, \mathbf{A})$ has linear time data and expression complexity, while model checking for $\mathrm{HDL}(@, \downarrow, \Downarrow, \mathbf{A})$ has exponential time expression complexity and polynomial time data complexity. In any case, the problem can be solved in polynomial space.*

Fig. 5. Embedding QBF into model checking for hybrid logics.



4.2 Lower Bounds

Can we do better than the upper bounds given in Section 4.1? Model checking for first-order logic is PSPACE-complete. Since HL($\exists, @, \mathbf{F}$) is as expressive as first-order logic [15], model checking for HL($\exists, @, \mathbf{F}$) is PSPACE-complete as well. What about fragments of HL($\exists, @, \mathbf{F}$)? The question is particulary interesting for the fragment HL($\downarrow, @, \mathbf{F}$), since it corresponds to the *bounded fragment* of the first-order correspondence language [9]. Unfortunately, the model checking problem for HL($\downarrow, @, \mathbf{F}$), and hence for the bounded fragment, is still PSPACE-hard, even without @, nominals and propositions. It is worth noticing that all the lower bounds in this section refer to *expression complexity*.

Theorem 4.7 *Model checking for the pure nominal-free fragment of* HL($\downarrow, \mathbf{F}$) *is* PSPACE*-complete.*

Proof. PSPACE-membership follows from Theorem 4.5. To prove PSPACE-hardness, we embed Quantified Boolean Formulas (QBF) [34] into the model checking problem for the pure nominal-free fragment of HL($\downarrow, \mathbf{F}$). We proceed in two steps. We first embed QBF into the model checking problem for HL($\downarrow, @, \mathbf{F}$). Then, we remove the @ operator and atomic letters.

Recall that an instance of QBF has the form $\Psi = Q_1 x_1 \ldots Q_n x_n . \alpha(x_1, \ldots, x_n)$, where $Q_i \in \{\exists, \forall\}$, and $\alpha(x_1, \ldots, x_n)$ is a Boolean formula using variables $x_1, \ldots, x_n$. Let $\mathsf{NOM} = \{\mathtt{true}, \mathtt{false}, \mathtt{home}\}$ and let $\mathcal{M}$ be the model depicted in Figure 5, left-hand side. Let ϕ_Ψ be the HL($\downarrow, @, \mathbf{F}$)-formula obtained from Ψ by replacing every occurrence of $\exists x$ by $@_{\mathtt{home}}\mathbf{F}{\downarrow}x$, every occurrence of $\forall x$ by $@_{\mathtt{home}}\mathbf{G}{\downarrow}x$, every occurrence of x by $@_x\mathtt{true}$, and every occurrence of $\neg x$ by $@_x\mathtt{false}$. We have that Ψ is true if, and only if, $\mathcal{M}, 1 \models \phi_\Psi$. We now remove @ and atomic letters. To remove @, we have to find a way to "come back home" after $\mathbf{F}{\downarrow}x$ has fixed the variable x. We can add to the previous model two more edges, one from 2 to 1, and the other from 3 to 1, and use the $\mathbf{F}$ operator to come home. Since we don't have atomic letters, we have to distinguish in some structural way between state 2 (denoting true) and state 3 (denoting false). We can add, for instance, a reflexive edge leaving 2. The resulting frame is depicted in Figure 5 (right-hand side). Without loss of generality, we assume that negation in $\alpha(x_1, \ldots, x_n)$ is applied only to variables. Let τ be the translation given in Figure 6. We leave it to the reader to check that Ψ is true if and only if $\mathcal{M}, 1 \models \tau(\Psi)$. $\square$

Fig. 6. From QBF to hybrid logic.

$$\tau(x) = \mathbf{F}(x \wedge \downarrow y.\mathbf{F}y) \qquad \tau(\neg x) = \mathbf{F}(x \wedge \downarrow y.\mathbf{G}\neg y)$$

$$\tau(\alpha_1 \wedge \alpha_2) = \tau(\alpha_1) \wedge \tau(\alpha_2) \qquad \tau(\alpha_1 \vee \alpha_2) = \tau(\alpha_1) \vee \tau(\alpha_2)$$

$$\tau(\exists x.\alpha) = \mathbf{F}{\downarrow}x.\mathbf{F}(\downarrow y.\mathbf{G}\neg y \wedge \tau(\alpha)) \quad \tau(\forall x.\alpha) = \mathbf{G}{\downarrow}x.\mathbf{F}(\downarrow y.\mathbf{G}\neg y \wedge \tau(\alpha))$$

Theorem 4.8 *Model checking for the pure nominal-free fragments of* $\mathrm{HL}(\downarrow, \mathbf{F}^+)$ *is* PSPACE*-complete.*

Proof. PSPACE-membership follows from Theorem 4.5. To prove hardness, we embed QBF into the model checking problem for the pure nominal-free fragment of $\mathrm{HL}(\downarrow, \mathbf{F}^+)$. Let $\Psi = Q_1 x_1 \ldots Q_n x_n.\alpha(x_1, \ldots, x_n)$, where $Q_i \in \{\exists, \forall\}$, and $\alpha(x_1, \ldots, x_n)$ is a Boolean formula using variables $x_1, \ldots, x_n$. Let $\mathcal{M}$ be the unlabelled model with frame $\langle\{1, 2\}, \{(1, 2), (2, 1)\}\rangle$. Let ϕ_Ψ be the $\mathrm{HL}(\downarrow, @, \mathbf{F})$-formula obtained from Ψ by replacing every occurrence of $\exists x$ by $\mathbf{F}^+{\downarrow}x$ and every occurrence of $\forall x$ by $\mathbf{G}^+{\downarrow}x$. Then Ψ is true if, and only if, $\mathcal{M}, 1 \models \phi_\Psi$. $\square$

Theorem 4.9 *Model checking for the pure nominal-free fragment of* $\mathrm{HL}(\exists)$ *is* PSPACE*-complete.*

Proof. PSPACE-membership follows from Theorem 4.5. To prove hardness, we embed QBF into the model checking problem for the pure nominal-free fragments of $\mathrm{HL}(\exists)$. Let $\Psi = Q_1 x_1 \ldots Q_n x_n.\alpha(x_1, \ldots, x_n)$, where $Q_i \in \{\exists, \forall\}$, and $\alpha(x_1, \ldots, x_n)$ is a Boolean formula using variables $x_1, \ldots, x_n$. Ψ is a pure nominal-free formula in $\mathrm{HL}(\exists)$. Let $\mathcal{M}$ be the unlabelled model based on the frame $\langle\{1, 2\}, \emptyset\rangle$. Then Ψ is true if, and only if, $\mathcal{M}, 1 \models \Psi$. $\square$

Corollary 4.10 *Model checking for the pure nominal-free fragment of* $\mathrm{HL}(\downarrow, \mathbf{A})$ *is* PSPACE*-complete.*

Theorem 4.11 *Model checking for* $\mathrm{HL}(\Downarrow, \mathbf{F})$ *is* PSPACE*-complete.*

Proof. The PSPACE upper bound comes from Section 4.1. To prove hardness, we embed the model checking problem for $\mathrm{HL}(\downarrow, \mathbf{F})$ into the same problem for $\mathrm{HL}(\Downarrow, \mathbf{F})$. Consider a hybrid model $\mathcal{M} = \langle M, R, V\rangle$ and a formula α in $\mathrm{HL}(\downarrow, \mathbf{F})$. We construct a hybrid model $\mathcal{M}' = \langle M, R, V'\rangle$, where, for each $m \in M$, there exists a fresh nominal i_m with $V'(i_m) = \{m\}$. Moreover, let α' be the formula in $\mathrm{HL}(\Downarrow, \mathbf{F})$ that is obtained from α by replacing each instance of $\downarrow$ by $\Downarrow$ and each instance of x by v_x, where $x \in \mathsf{WVAR}$ and $v_x \in \mathsf{WVAR}'$. Then, we have that $\mathcal{M}, g, m \models \alpha$ if, and only if, $\mathcal{M}', g, m \models \alpha'$. $\square$

Theorem 4.12 *Model checking for the pure nominal-free fragment of* $\mathrm{HDL}(\downarrow)$ *is* PSPACE*-complete.*

5 Hybrid Logic in Action

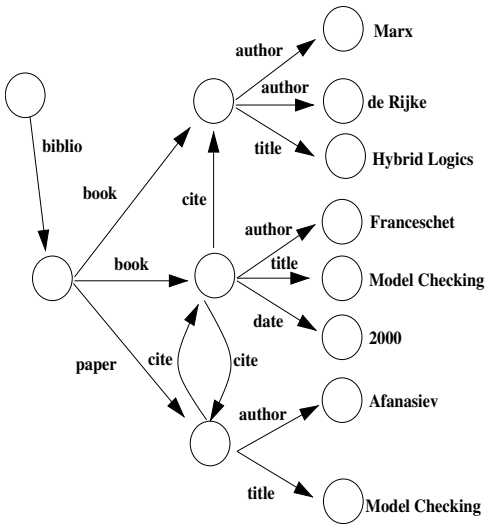
On the previous pages we have discussed, and obtained, complexity results for model checking a variety of hybrid logics. What's the point? In this section we describe an application of hybrid logic model checking to the task of evaluation of queries and constraints on semistructured data. We will encounter a number of query and constraint formats as well as reasoning tasks, all of which correspond to model checking in some specific hybrid language.

5.1 Semistructured data

The growth of the World Wide Web has given us a vast, largely accessible database. The Extensible Markup Language (XML) [22] is a textual representation of information that was designed to represent the hierarchical content of documents. It has been proposed as a data model for semistructured databases since it is able to naturally represent missing or duplicated data as well as deeply nested information [1]. As an example, Figure 7 contains the XML representation of a simple bibliography file. A very natural representation for semistructured data is a labelled graph. In this paper, we will represent

Fig. 7. (Left) An XML representation of a bibliography file. (Right) A graphical representation of the same file.

```
<biblio>
  <book id = "o1">
    <author> Marx </author>
    <author> de Rijke </author>
    <title> Hybrid Logics </title>
  </book>
  <book id = "o2">
    <author> Franceschet </author>
    <title> Model Checking </title>
    <date> 2000 </date>
    <cite idref = "o1"/>
    <cite idref = "o3"/>
  </book>
  <paper id = "o3">
    <author> Afanasiev </author>
    <title> Model Checking </title>
    <cite idref = "o2"/>
  </paper>
</biblio>
```



semistructured data as a graph in which a node corresponds to an object and an edge corresponds to an object attribute. Edges are labelled with attribute names, while leaf nodes are labelled with object values (internal nodes are not labelled). The data graph corresponding to the XML example on the left-hand side of Figure 7 is depicted on the right-hand side of Figure 7.

5.2 Query Processing via Hybrid Logic Model Checking

It is possible to perform significant query processing on semistructured data via model checking for hybrid languages. Quite a number of query languages for semistructured data have been proposed in the literature. However, many of them are similar in spirit and sometimes even in syntax [1]. For this reason, we have chosen one of them as a model: Lorel [3]. Lorel is the query language in the Lore system [32], which was designed for managing semistructured data. We discuss three fragments of Lorel, that differ in their expressive power; each will be embedded in a suitable hybrid logic. As a consequence, we are able to process a query in Lorel by taking advantage of a model checker for hybrid logics. Moreover, with the aid of the well-understood hybrid logic framework, it will be easy to investigate and compare the expressive power of various fragments of the Lorel query language.

All fragments that we consider allow regular expressions for navigating the data graph. Some offer the possibility of comparing object identities, which allows us to implement queries with joins. We will focus on *monadic queries* only: queries that return a set of objects of the database, or, equivalently, a set of nodes of the graph representation of the database. The reason for this restriction is that we want to embed the query processing problem into the global model checking problem, and the output of a global model checker is a set of nodes. We will consider 3 increasingly large fragments of Lorel: $\mathcal{L}^1_{\text{qry}} \subset \mathcal{L}^2_{\text{qry}} \subset \mathcal{L}^3_{\text{qry}}$. The most expressive language, $\mathcal{L}^3_{\text{qry}}$, captures a large subfragment of Lorel. Additional features of Lorel that are not expressible in $\mathcal{L}^3_{\text{qry}}$ will be discussed towards the end of the section.

Fragment 1: $\mathcal{L}^1_{\text{qry}}$ We start by defining the query language $\mathcal{L}^1_{\text{qry}}$. A query in $\mathcal{L}^1_{\text{qry}}$ has the following schema Q_1:

(Q_1) `select X`
`from rexp X`
`where X.fexp`,

where `rexp` is a regular expression on edge labels (i.e., attribute names), and `fexp` is a so-called filter expression. Intuitively, Q_1 retrieves all nodes reachable

from the root through `rexp` satisfying the filter `fexp`. The variable `X` is used as a container for these nodes. We call the variable `X` in the 'select' clause of the query the *focus* of the query, the condition `rexp X` in the 'from' clause of the query the *selection expression*, and the condition `X.fexp` in the 'where' clause of the query the *filter expression*. The syntax of filter expressions is:

$$\texttt{fexp} = \texttt{true} \mid \texttt{rexp} \mid \texttt{rexp} :: \texttt{a} \mid \texttt{fexp and fexp} \mid \texttt{fexp or fexp} \mid \texttt{not fexp},$$

where `a` is an object value. Given a set of nodes `X`, the filter `X.rexp` selects a node v in `X` if there exists at least one reachable node from v trough `rexp`. The filter `X.rexp::a` adds an additional constraint: it filters a node v in `X` if there exists at least one reachable node from v trough `rexp` that is labelled with a. E.g., with reference the example in Figure 7, consider the following:

```
select X
from biblio.book X
where X.(author::Franceschet and date::2000)
```

This query selects all books written in 2000 such that Franceschet is one of the authors. In our example, book $o2$ is retrieved. Moreover, the query

```
select X
from biblio._ X
where X.abstract
```

retrieves all entries for which an abstract has been provided: none.

The simple query language $\mathcal{L}^1_{\text{qry}}$ can be embedded into hybrid dynamic logic with only one nominal *root* for the data graph root and no hybrid operators. The embedding τ_1 is as follows. Let ω be the obvious embedding from filter expressions to hybrid formulas: $\omega(\texttt{true}) = \top$, $\omega(\texttt{rexp}) = \langle \texttt{rexp} \rangle \top$, $\omega(\texttt{rexp} :: \texttt{a}) = \langle \texttt{rexp} \rangle a$, $\omega(\texttt{fexp}_1 \texttt{ and } \texttt{fexp}_2) = \omega(\texttt{fexp}_1) \wedge \omega(\texttt{fexp}_2)$, $\omega(\texttt{fexp}_1 \texttt{ or} \texttt{fexp}_2) = \omega(\texttt{fexp}_1) \vee \omega(\texttt{fexp}_2)$, and $\omega(\texttt{not fexp}) = \neg\omega(\texttt{fexp})$. To the query schema Q_1 we associate the hybrid formula $\tau_1(Q_1) = \langle \texttt{rexp} \rangle^{-1} \mathit{root} \wedge \omega(\texttt{fexp})$. Notice how the selection expression is translated 'backward,' while the filter expression is translated 'forward.'

Next, we clarify the relation between query processing for semistructured data and model checking for hybrid logic. Define the *answer set* of a monadic query q with respect to a semistructured database D as the set of nodes retrieved by q belonging to the data graph G_D associated to D. Each node in the answer set corresponds to an element in the corresponding XML representation of D. Moreover, given a hybrid formula α and a hybrid model $\mathcal{M}$, let the *truth set* of α with respect to $\mathcal{M}$ be the set of nodes of $\mathcal{M}$ at which α is true. The following result relates the query processing problem for semistructured data to the model checking problem for hybrid dynamic logic.

Proposition 5.1 *Let q be a query in $\mathcal{L}^1_{\text{qry}}$ and D be a semistructured database. Then, the answer set of q with respect to D corresponds to the truth set of $\tau_1(q)$ with respect to G_D.*

Moreover, since the translation τ_1 goes inside a fragment of HDL($@, \mathbf{A}$), by virtue of Theorem 4.6, we have the following corollary.

Corollary 5.2 *Query processing for $\mathcal{L}^1_{\text{qry}}$ has linear time data and expression complexity.*

Fragment 2: $\mathcal{L}^2_{\text{qry}}$ The expressive power of $\mathcal{L}^1_{\text{qry}}$ can be extended by splitting the selection expression in more than one chunk. Consider the following query that selects the authors of all entries with title *Model Checking*:

```
select Y
from biblio._ X, X.author Y
where X.title::Model Checking
```

This query is not definable in $\mathcal{L}^1_{\text{qry}}$ but it can be embedded into the basic hybrid dynamic logic as $\langle \texttt{author} \rangle^{-1}(\langle \texttt{bibilo}\_ \rangle^{-1} root \wedge \langle \texttt{title} \rangle Model\ Checking)$. We define the query schema Q_2 as follows:

```
(Q2) select Xi
  from rexp1 X1, X1.rexp2 X2, ..., Xn−1.rexpn Xn
  where fexp1, ..., fexpm
```

Each filter expression $\texttt{fexp}_\texttt{j}$ has the form $\texttt{X.fexp}$, where $\texttt{X}$ is a variable and $\texttt{fexp}$ in a filter expression as in Q_1. Intuitively, the schema Q_2 binds the variable $\texttt{X}_1$ to the nodes reachable from the root through the regular expression $\texttt{rexp}_1$, it binds the variable $\texttt{X}_2$ to the nodes reachable from a node in $\texttt{X}_1$ through the regular expression $\texttt{rexp}_2$, and so on. The filter expression $\texttt{X.fexp}$ filters the nodes placed in the variable $\texttt{X}$ according to the Boolean filter $\texttt{fexp}$. Finally, the nodes contained in the focus $\texttt{X}_\texttt{i}$ are selected. A well-formed query q is defined as follows. Let X be the focus of q, S the selection sequence of q, F the filter sequence of q, and V the variables in S. The query q is *well-formed* if (i) $X \in V$, and (ii) all variables used in the filter sequence F are in V. Let $\mathcal{L}^2_{\text{qry}}$ be the query language containing all the well-formed queries according to the schema Q_2. Notice that $\mathcal{L}^1_{\text{qry}} \subset \mathcal{L}^2_{\text{qry}}$.

We now develop an embedding τ_2 of queries in $\mathcal{L}^2_{\text{qry}}$ into the basic hybrid dynamic logic with only one nominal *root* for the root of the data graph. Consider the query schema Q_2. For each variable $\texttt{X}_\texttt{j}$ in Q_2, let $\texttt{fexp}_\texttt{j}$ be the filter expression associated with the variable $\texttt{X}_\texttt{j}$, or $\texttt{fexp}_\texttt{j} = \texttt{true}$ if no filter expression has been explicitly associated with $\texttt{X}_\texttt{j}$. We use the following two auxiliary functions ν and ν^{-1} mapping a variable in Q_2 into a hybrid formula.

$$\nu(X_j) = \begin{cases} \omega(\mathtt{fexp_j}) \wedge \langle \mathtt{rexp_{j+1}} \rangle\, \nu(\mathtt{X_{j+1}}) & \text{if } j < n \\ \omega(\mathtt{fexp_j}) & \text{if } j = n \end{cases}$$

$$\nu^{-1}(X_j) = \begin{cases} \omega(\mathtt{fexp_j}) \wedge \langle \mathtt{rexp_j} \rangle^{-1}\, \nu^{-1}(\mathtt{X_{j-1}}) & \text{if } j > 1 \\ \omega(\mathtt{fexp_j}) \wedge \langle \mathtt{rexp_j} \rangle^{-1} root & \text{if } j = 1 \end{cases}$$

Let $\mathtt{X_i}$ be the focus of Q_2, that is, $\mathtt{X_i}$ is the variable in the select clause of Q_2. The translation τ_2 of Q_2 into hybrid logic is as follows:

$$\tau_2(Q) = \begin{cases} \nu(\mathtt{X_i}) \wedge \langle \mathtt{rexp_i} \rangle^{-1}\, \nu^{-1}(\mathtt{X_{i-1}}) & \text{if } i > 1 \\ \nu(\mathtt{X_i}) \wedge \langle \mathtt{rexp_i} \rangle^{-1} root & \text{if } i = 1 \end{cases}$$

Notice that $\tau_2(\mathcal{L}^1_{\text{qry}}) = \tau_1(\mathcal{L}^1_{\text{qry}})$.

Proposition 5.3 *Let q be a query in $\mathcal{L}^2_{\text{qry}}$ and D be a semistructured database. Then, the answer set of q with respect to D corresponds to the truth set of $\tau_2(q)$ with respect to G_D.*

Since the codomain of the translation τ_2 is a fragment of HDL($@, \mathbf{A}$), by virtue of Theorem 4.6, we have the following corollary:

Corollary 5.4 *Query processing for $\mathcal{L}^2_{\text{qry}}$ has linear time data and expression complexity.*

The results in Corollaries 5.2 and 5.4 do not really depend on previous results for model checking hybrid logics, and in fact they can be obtained directly from model checking converse PDL. Indeed, the root nominal can be simulated by a propositions holding at exactly the root node. The material below does use the expressive power of hybrid logic in an essential way.

Fragment 3: $\mathcal{L}^3_{\text{qry}}$ So far, we have not used the full power of hybrid logic. In particular, the hybrid binder $\downarrow$ has not been exploited in the query translation. Consider the following query that selects papers with at least two authors:

```
select X
from biblio.paper X, X.author Y, X.author Z
where Y ≠ Z
```

It can be embedded in hybrid logic, but we need the binder $\downarrow$:

$$\downarrow x.\langle \mathtt{biblio.paper} \rangle^{-1} root \wedge \langle \mathtt{author} \rangle \downarrow y.@_x \langle \mathtt{author} \rangle \downarrow z.\ y \neq z.$$

Moreover, the following query selects all the self-reference papers, that is, all papers whose authors cite themselves:

```
select X
from biblio.paper X, X.cite Y
where X = Y
```

It corresponds to the hybrid formula $\downarrow x.\langle \texttt{biblio.paper}\rangle^{-1} root \wedge \langle \texttt{cite}\rangle \downarrow y.x = y$. The following query retrieves all papers p such that there exists a paper q reachable from p trough a path of cite edges that cites back to p:

```
select X
from biblio.paper X, X.cite.cite* Y
where X = Y
```

This query maps to $\downarrow x.\langle \texttt{biblio.paper}\rangle^{-1} root \wedge \langle \texttt{cite.cite}^*\rangle \downarrow y.x = y$.

We define a query schema Q_3 that allows the above queries as follows:

(Q_3) `select` $\texttt{X}_\texttt{i}$
`from` $\texttt{sexp}_1$, $\texttt{sexp}_2$, ..., $\texttt{sexp}_\texttt{n}$
`where` $\texttt{fexp}_1$, $\texttt{fexp}_2$, ..., $\texttt{fexp}_\texttt{m}$,

where each $\texttt{sexp}_\texttt{j}$ is a selection expression and each $\texttt{fexp}_\texttt{j}$ is a filter expression. A selection expression is either `rexp X` or `X.rexp Y`, where `rexp` is a regular expression and `X` and `Y` are variables. A filter expression is either `X.fexp`, or $\texttt{X} = \texttt{Y}$, or $\texttt{X} \neq \texttt{Y}$, where `X` and `Y` are variables and `fexp` is a filter expression as in Q_1. Not every selection sequence is legal. For instance, `X.a Y`, `X.b Y` is not legal for two reasons. First, it does not specify the content of `X`. Second, the content of `Y` is ambiguous. A well-formed query q is defined as follows. Let X be the focus of q, S the selection sequence of q, F the filter sequence of q, and V the variables in S. We construct the edge-labelled directed graph T_S with nodes in $V \cup \{/\}$. There is an edge $(/, \texttt{X})$ labelled with `rexp` if `rexp X` is in S, and there is an edge $(\texttt{X}, \texttt{Y})$ labelled with `rexp` if `X.rexp Y` is in S. We say that S is *legal* if T_S is a tree rooted at $/$. That is (i) each node in T_S is reachable from $/$, (ii) the root $/$ of T_S has no predecessor and all the other nodes in T_S have exactly one predecessor. The query q is *well-formed* if (i) $X \in V$, (ii) S is legal, and (iii) all the variables used in the filter sequence F are in V. Let $\mathcal{L}^3_{\text{qry}}$ be the query language containing all the well-formed queries according to the schema Q_3. Notice that $\mathcal{L}^2_{\text{qry}} \subset \mathcal{L}^3_{\text{qry}}$.

We now develop an embedding τ_3 of queries in $\mathcal{L}^3_{\text{qry}}$ into the hybrid dynamic logic with nominals, @ and $\downarrow$. For the sake of simplicity, we describe the embedding with an example. It is not difficult from this example to reconstruct the full query translation. Consider the following abstract query q:

```
select Z
from a X, X.b Y, X.c Z, Z.d W
where X.e,Y.f,Z.g,W.h,Y = W
```

Notice that q is well-formed. The corresponding hybrid formula is as follows:

$$\downarrow z.\langle g\rangle\top \wedge \langle d\rangle\downarrow w.\langle h\rangle\top \wedge$$
$$@_z\langle c\rangle^{-1}\downarrow x.\langle e\rangle\top \wedge \langle b\rangle\downarrow y.\langle f\rangle\top \wedge @_x\langle a\rangle^{-1}root \wedge$$
$$y = w$$

The formula has been deliberately divided into three lines. The first line constraints the focus of the query (node `Z`) and its subtree (node `W`). The second line predicates over the unique path from the parent of the focus to the root of the tree as well as over all the subtrees rooted at children of nodes on this path not belonging to the path (nodes `X`, `Y`, and `/`). The third line captures the identity checking constraints. This technique can be generalized to arbitrary trees. Notice that the use of the binder $\downarrow$ is necessary to encode the constraint that compares object identities. Indeed, if we remove the filter `Y = W` from the above query, the latter can be encoded without $\downarrow$ as follows:

$$\langle g\rangle\top \wedge \langle d\rangle\langle h\rangle\top \wedge$$
$$\langle c\rangle^{-1}(\langle e\rangle\top \wedge \langle b\rangle\langle f\rangle\top \wedge \langle a\rangle^{-1}root)$$

Proposition 5.5 *Let q be a query in $\mathcal{L}^3_{\text{qry}}$ and D be a semistructured database. Then, the answer set of q with respect to D corresponds to the truth set of $\tau_3(q)$ with respect to G_D.*

Since the the translation τ_3 embeds into HDL$(@, \downarrow, \Downarrow, \mathbf{A})$, by virtue of Theorem 4.6, we have the following corollary (we do not know whether it is optimal):

Corollary 5.6 *Query processing for $\mathcal{L}^3_{\text{qry}}$ has exponential time expression complexity and polynomial time data complexity.*

Additional Features of Lorel Lorel includes additional queries that are not immediately expressible in our language. For instance, a query in Lorel may use regular expressions on object values, as in the following:

```
select X
from biblio.paper X, X.title Y
where matches("*.(D|d)ata.*",Y)
```

The query selects all papers with a title containing either `Data` or `data`. Regular expressions on object values can be incorporated into our model checking framework as follows. Given a query q featuring the regular expression r and a database D, the data graph representing D is preprocessed and all leaf nodes with a value matching r are labelled with a fresh symbol `reg_r`. Moreover, each instance of `matches("*.(D|d)ata.*",Y)` in q is replaced by `Y.`ϵ`::reg_r`. The query is then translated into hybrid logic and model checking is applied.

Lorel queries may also allow variables containing object values, instead of object identifiers. Consider the following query that retrieves all papers with two different authors with the same first name (we assume that the `author` element has a subelement `name` which in turn has an atomic subelement `first`):

```
select X
from biblio.paper X, X.author Y, Y.name.first N,
X.author Z, Z.name.first M
where Y ≠ Z, N = M
```

The constraint `Y` $\neq$ `Z` compares object identities, while `N = M` compares object values, since `Y` and `Z` contain internal nodes, while `N` and `M` contain leaf nodes. This query can be implemented in hybrid logic by using the hybrid binder $\Downarrow x$ that binds the current node value to the variable x:

$\langle \texttt{biblio.paper} \rangle^{-1} root \wedge \downarrow x.\langle \texttt{author} \rangle \downarrow y.\langle \texttt{name.first} \rangle \Downarrow n.$
$@_x \langle \texttt{author} \rangle \downarrow z.\langle \texttt{name.first} \rangle \Downarrow m. y \neq z \wedge n = m$

Finally, Lorel includes label and path variables as well. These variables can be used to combine schema and data information. These features are beyond the expressive power of hybrid logic with nominals for states. *Path nominals*, i.e., nominals interpreted over paths, and corresponding path binders, have been introduced in hybrid logics [16]. However, at present model checking results for hybrid logics with path nominals and path binders are non-existent.

Constraint Evaluation via Hybrid Model Checking So far we have considered hybrid logic model checking as a mechanism for evaluating queries in (fragments of) Lorel. We now change tack and consider constraint evaluation for semistructured data. Recall that integrity constraints on structured data are conditions that restrict the possible populations of the database. They are important to maintain the integrity of data as well as for query optimization [2]. *Path constraints* are generalizations of integrity constraints in the context of semistructured data [1,24]. They are navigation conditions imposing conditions on nodes at arbitrary depths in the data graph. They include functional, inclusion and inverse path constraints [24].

Consider a semistructured database containing information about authors and publications. Each author has a list of publications and each publication has a corresponding list of authors. An example in XML is shown in Figure 8. Constraints such as the following are reasonable for this type of data:

(1) for each author a, all the publications in a's publications list should be contained in the database;
(2) for each publication p, all the authors in p's authors list should be contained in the database;

Fig. 8. Authors and publications in XML.

```
<author id = "a1">
  <name> Marx </author>
  <has_written idref = "p1"/>
  <has_written idref = "p2"/>
</author>
<author id = "a2">
  <name> de Rijke </author>
  <has_written idref = "p1"/>
</author>
<publication id = "p1">
  <title> Hybrid Logics </title>
  <code> MdeR03 </code>
  <year> 2003 </year>
  <written_by idref = "a1"/>
  <written_by idref = "a2"/>
</publication>
<publication id = "p2">
  <title> Computational Complexity </title>
  <code> M00 </code>
  <year> 2000 </year>
  <written_by idref = "a1"/>
</publication>
⋮
```

(3) for each author a and for each publication p in a's publications list, a should be contained in p's authors list; and
(4) for each publication p and for each author a in p's authors list, p should be contained in a's publications list.

The data in Figure 8 satisfies the above constraints. We could require that the attribute `code` is a *key* for the element publication: different publications should have different code values. Less restrictively, we could ask that the attribute `code` *functionally determines* the authors of a publication: any two different publications with the same code values should have the same authors.

More generally, let r, p and q be regular expressions on attribute names. Let τ be an attribute name and S a set of attribute names. A τ node is a node reachable through an edge labelled with τ. We consider the following constraints:

- *key constraints*, denoted $\tau[S] \rightarrow \tau$, saying that, for all τ nodes x and y, if x and y agree on the values of nodes reachable through attributes in S, then x and y are the same node;
- *path functional constraints*, denoted $\tau.p \rightarrow \tau.q$, saying that, for all τ nodes

x and y, if x and y agree on the values of nodes reachable through p, then they should agree on the values of nodes reachable through q as well;

- *circular constraints*:
  - · *forward* version, denoted $p \rightrightarrows q$, saying that all nodes reachable from the root through p are also reachable from the root through q;
  - · *backward* version, denoted $p \leftrightarrows q$, saying that all nodes reachable from the root through p can reach back to the root through q;
- *lollipop constraints*:
  - · *forward* version, denoted $r \rightarrow p \rightrightarrows q$, saying that, for each node x reachable from the root through r it holds that all nodes that are reachable from x through p are also reachable from x through q;
  - · *backward* version, denoted $r \rightarrow p \leftrightarrows q$, saying that, for each node x reachable from the root through r it holds that all nodes that are reachable from x through p can reach back to x through q.

Forward circular constraints are special cases of forward lollipop constraints in which r is empty; similarly for backward constraints. Forward circular constraints are also called *inclusion path constraints*, and backward lollipop constraints are sometimes referred to as *inverse path constraints* [1,24]. Whenever S is a singleton $\{a\}$, the key constraint $\tau[\{a\}] \rightarrow \tau$ corresponds to the functional constraint $\tau.a \rightarrow \tau$.

Constraints (1) and (2) in our example above are forward circular constraints: (1) is a forward circular constraint in which $p =$ `*.publication. written_by` and $q =$ `*.author`, and (2) is a forward circular constraint constraint with $p =$ `*.author.has_written` and $q =$ `*.publication`. Constraints (3) and (4) above are backward lollipop constraints: (3) is a backward lollipop constraint with $r =$ `*.publication`, $p =$ `written_by` and $q =$ `has_written`, and (4) is a backward lollipop constraint with $r =$ `*.author`, $p =$ `has_written` and $q =$ `written_by`. The key constraint `publication`[`code`] $\rightarrow$ `publication` states that the attribute code of the element publication is a key for the element publication. Finally, the functional path constraint `publication.code` $\rightarrow$ `publication.written_by.name` asks that the code of the element publication determines the set of authors of the publication.

To show how such constraints can be expressed in hybrid dynamic languages, we define a constraint language $\mathcal{L}_{\text{con}}$ containing any Boolean combination of constraints as introduced above. Formally, a constraint c in $\mathcal{L}_{\text{con}}$ has the following syntax:

- $c = atom \mid c \wedge c \mid c \vee c \mid \neg c$
- $atom = \tau[S] \rightarrow \tau \mid \tau.p \rightarrow \tau.q \mid p \rightrightarrows q \mid p \leftrightarrows q \mid r \rightarrow p \rightrightarrows q \mid r \rightarrow p \leftrightarrows q$,

where τ is an attribute name, S is a set of attribute names, and r, p, q are regular expressions on attribute names.

Next we show how to express the constraints in $\mathcal{L}_{\text{con}}$ within hybrid dynamic logic. We define a mapping σ from the constraint language $\mathcal{L}_{\text{con}}$ to HDL$(@, \downarrow, \Downarrow)$. Let *root* be a nominal for the root of the data graph. The easiest constraints to encode are the circular ones. Their encodings do not require hybrid binders:

$$\sigma(p \rightrightarrows q) = @_{root}[p]\langle q\rangle^{-1} root$$
$$\sigma(p \leftrightarrows q) = @_{root}[p]\langle q\rangle root$$

We encode lollipop constraints. Their encodings require only one nesting of the hybrid binder $\downarrow$:

$$\sigma(r \rightarrow p \rightrightarrows q) = @_{root}[r]{\downarrow}x.[p]\langle q\rangle^{-1} x$$
$$\sigma(r \rightarrow p \leftrightarrows q) = @_{root}[r]{\downarrow}x.[p]\langle q\rangle x$$

Key and functional constraints are more involved, since they involve both the comparison of node identities and the comparison of node values. The translation of the key constraint $\tau[S] \rightarrow \tau$ states that, for all different τ nodes x and y, if x and y agree on the values of nodes reachable through attributes in S, then x and y are the same node:

$$\begin{aligned}\sigma(\tau[S] \rightarrow \tau) = \mathbf{A}{\downarrow}x.\mathbf{A}{\downarrow}y.(@_x\langle\tau\rangle^{-1}\top \wedge @_y\langle\tau\rangle^{-1} \wedge x \neq y) \rightarrow \\ (\bigwedge_{a\in S} @_x[a]{\Downarrow}v_1.@_y\langle a\rangle{\Downarrow}w_1.v_1 = w_1 \wedge \\ @_y[a]{\Downarrow}w_2.@_x\langle a\rangle{\Downarrow}v_2.v_2 = w_2) \rightarrow x = y\end{aligned}$$

The translation of the path functional constraints $\tau.p \rightarrow \tau.q$ states that, for all different τ nodes x and y, if x and y agree on the values of nodes reachable through p, they should also agree on the values of nodes reachable through q:

$$\begin{aligned}\sigma(\tau.p \rightarrow \tau.q) = \mathbf{A}{\downarrow}x.\mathbf{A}{\downarrow}y.(@_x\langle\tau\rangle^{-1}\top \wedge @_y\langle\tau\rangle^{-1} \wedge x \neq y) \rightarrow \\ (@_x[p]{\Downarrow}v_1.@_y\langle p\rangle{\Downarrow}w_1.v_1 = w_1 \wedge \\ @_y[p]{\Downarrow}w_2.@_x\langle p\rangle{\Downarrow}v_2.v_2 = w_2) \rightarrow \\ (@_x[q]{\Downarrow}v_1.@_y\langle q\rangle{\Downarrow}w_1.v_1 = w_1 \wedge \\ @_y[q]{\Downarrow}w_2.@_x\langle q\rangle{\Downarrow}v_2.v_2 = w_2)\end{aligned}$$

We conclude the description of our translation by stipulating that σ distributes over the Boolean operators. In σ it is convenient to replace the universal modality $\mathbf{A}$ with $@_{root}[*]$. Notice that in this way each translated constraint is a formula starting with $@_{root}$. Given a semistructured database D and an integrity constraint $c \in \mathcal{L}_{\text{con}}$, it is possible to check whether D satisfies c as follows. The database D is represented as a rooted graph G_D and the constraint c is translated into the hybrid formula $\sigma(c)$. Then, the formula $\sigma(c)$ is checked on G_D by using a hybrid model checker (in fact, it is sufficient to check the formula at the root of the graph). If the outcome of the model checker is the

empty set of nodes, then D does not satisfy c, otherwise it does.

Proposition 5.7 *Let c be an integrity constraint in $\mathcal{L}_{\text{con}}$ and D be a semistructured database. Then, D satisfies c if, and only if, the truth set of $\sigma(c)$ with respect to G_D is non-empty.*

The translation σ embeds (any Boolean combination of) circular constraints into HDL(@), lollipop constraints into the fragment of HDL(@, $\downarrow$) in which $\downarrow$ is nested only once, and key and functional constraints into the fragment of HDL(@, $\downarrow, \Downarrow, \mathbf{A}$) in which the hybrid binders are nested a fixed number of times. As a consequence, by virtue of Theorem 4.6, we have the following.

Corollary 5.8

- *The constraint evaluation problem for circular constraints can be solved in linear time both in the length of the constraint (expression complexity) and in the size of the database (data complexity);*
- *The constraint evaluation problem for lollipop constraints can be solved in linear time in the length of the constraint (expression complexity) and in quadratic time in the size of the database (data complexity);*
- *The constraint evaluation problem for key and functional constraints can be solved in linear time in the length of the constraint (expression complexity) and in polynomial time in the size of the database (data complexity).*

6 Conclusion and Work for the Future

We investigated the model checking problem for hybrid logics. We gave model checkers for a large number of fragments of hybrid and hybrid dynamic logic. We obtained lower bounds on the computational complexity of the model checking problem for hybrid logics with binders. We found that the addition of nominals and the @ operator does not increase the complexity of the model checking task. In contrast, whenever hybrid binders are present in the language, the running time of the resulting model checker is exponential in the nesting level of the binders. We cannot do better, since we proved that the model checking problem for hybrid logics with binders is PSPACE-complete.

We applied our findings to the problems of query and constraint evaluation for semistructured data. We identified significant fragments of well-known query and constraint languages for semistructured data that can be efficiently embedded into hybrid languages. These embeddings allowed us to solve query and constraint evaluation problems via model checking for hybrid logics.

An implementation of the model checkers `MCLITE` and `MCFULL` proposed in

this paper is available at `http://www.luigidragone.com/hlmc`. The code is written in C available under the GNU General Public License. It can be freely used, modified and distributed in conformity with this license.

Acknowledgements

We are very grateful to Carlos Areces, Daniel Gorin, Maarten Marx and the anonymous referees for helpful comments and suggestions.

References

[1] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web*. Morgan Kaufmann Publishers, Los Altos, CA 94022, USA, 2000.

[2] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, Reading, Mass., 1995.

[3] S. Abiteboul, D. Quass, J. McHugh, J. Widom, J. Wiener, and J. Widom. The Lorel Query Language for Semistructured Data. *International Journal on Digital Libraries*, 1(1):68–88, 1997.

[4] S. Abiteboul and V. Vianu. Regular path queries with constraints. *Journal of Computer and System Sciences*, 58(3):428–452, 1999.

[5] L. Afanasiev, M. Franceschet, M. Marx, and M. de Rijke. CTL Model Checking for Processing Simple XPath Queries. In *Proceedings TIME 2004*, pages 117–124, 2004.

[6] N. Alechina and M. de Rijke. Describing and querying semistructured data: Some expressiveness results. In S.M. Embury, N.J. Fiddian, W.A. Gray, and A.C. Jones, editors, *Advances in Databases*, LNCS. Springer, 1998.

[7] N. Alechina, S. Demri, and M. de Rijke. A modal perspective on path constraints. *Journal of Logic and Computation*, 13(6):939–956, 2003.

[8] L. De Alfaro. Model checking the World Wide Web. In *Proceedings CAV 2001*, volume 2102 of *LNCS*, pages 337–349, 2001.

[9] C. Areces, P. Blackburn, and M. Marx. A road-map on complexity for hybrid logics. In J. Flum and M. Rodriguez-Artalejo, editors, *Computer Science Logic*, volume 1683 of *LNCS*, pages 307–321. Springer, 1999.

[10] C. Areces, P. Blackburn, and M. Marx. The computational complexity of hybrid temporal logics. *Logic Journal of the IGPL*, 8(5):653–679, 2000.

[11] C. Areces, P. Blackburn, and M. Marx. Hybrid logics: Characterization, interpolation, and complexity. *Journal of Symbolic Logic*, 66(3):977–1010, 2001.

[12] C. Areces and J. Heguiabehere. HyLoRes 1.0: Direct resolution for hybrid logics. In A. Voronkov, editor, *Automated Deduction – CADE-18*, volume 2392 of *LNCS*, pages 156–160. Springer-Verlag, July 27-30 2002.

[13] P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*. Cambridge University Press, 2001.

[14] P. Blackburn and J. Seligman. Hybrid languages. *Journal of Logic, Language and Information*, 4:251–272, 1995.

[15] P. Blackburn and J. Seligman. What are hybrid languages? In M. Kracht, M. de Rijke, H. Wansing, and M. Zakharyaschev, editors, *Advances in Modal Logic, Volume 1*, pages 41–62. CSLI Publications, 1998.

[16] R. Bull. An approach to tense logic. *Theoria*, 36:282–300, 1970.

[17] P. Buneman, W. Fan, and S. Weinstein. Path constraints on semistructured and structured data. In *Proceedings PODS*, 1998.

[18] D. Calvanese, G. De Giacomo, and M. Lenzerini. Representing and reasoning on XML documents: A description logic approach. *Journal of Logic and Computation*, 9(3):295–318, 1999.

[19] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1999.

[20] E. M. Clarke and H. Schlingloff. Model checking. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume II, chapter 24, pages 1635–1790. Elsevier Science, 2001.

[21] R. Cleaveland and B. U. Steffen. A linear-time model checking algorithm for the alternation-free modal mu-calculus. *Formal Methods in System Design*, 2:121–147, 1993.

[22] World Wide Web Consortium. Extensible markup language (XML). Available at http://www.w3.org/XML, 1998.

[23] E.A. Emerson. Temporal and modal logic. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science, Vol. B*, pages 995–1072. Elsevier Science Publishers B.V., 1990.

[24] W. Fan and J. Siméon. Integrity constraints for XML. *Journal of Computer and System Sciences*, 66(1):254–291, 2003.

[25] M. Franceschet, M. de Rijke, and B. H. Schlingloff. Hybrid logics on linear structures: expressivity and complexity. In *Proceedings TIME 2003*, pages 166–173. IEEE Computer Society Press, 2003.

[26] V. Goranko. Hierarchies of modal and temporal logics with reference pointers. *Journal of Logic, Language and Information*, 5(1):1–24, 1996.

[27] G. Gottlob and C. Koch. Monadic Queries over Tree-Structured Data. In *Logic in Computer Science*, pages 189–202, Los Alamitos, CA, USA, July 22–25 2002. IEEE Computer Society.

[28] J. Y. Halpern, R. Harper, N. Immerman, P. G. Kolaitis, M. Vardi, and V. Vianu. On the unusual effectiveness of logic in computer science. *The Bulletin of Symbolic Logic*, 7(2):213–236, 2001.

[29] D. Harel, J. Tiuryn, and D. Kozen. *Dynamic Logic*. MIT Press, 2000.

[30] HyLo: The Hybrid Logic home page. URL: `http://hylo.loria.fr`.

[31] M Marx. Conditional XPath, the first order complete XPath dialect. In *Proceedings PODS*, pages 13–22, 2004.

[32] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: A database management system for semistructured data. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 26(3):54–66, 1997.

[33] G. Miklau and D. Suciu. Containment and equivalence for an XPath fragment. In *Proceedings PODS*, pages 65–76, 2002.

[34] C. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[35] S. Passy and T. Tinchev. An essay in combinatory dynamic logic. *Information and Computation*, 93(2):263–332, 1991.

[36] A. Prior. *Past, Present and Future*. Clarendon, Oxford, 1967.

[37] E. Quintarelli. *Model-Checking Based Data Retrieval: an application to semistructured and temporal data*. PhD thesis, Dipartimento di Elettronica e Informazione, Politecnico di Milano, 2000.

[38] B. ten Cate. *Model theory for extended modal languages*. PhD thesis, University of Amsterdam, 2005. ILLC Dissertation Series DS-2005-01.