Model Checking for Hybrid Logics

Massimo Franceschet

Dipartimento di Scienze,
Università "G. d'Annunzio" di Chieti-Pescara, Italy
ILLC, University of Amsterdam, The Netherlands
francesc@science.uva.nl

Maarten de Rijke

ILLC, University of Amsterdam, The Netherlands
mdr@science.uva.nl

Abstract

We investigate the complexity of the model checking problem for hybrid logics. We provide model checkers for various hybrid fragments and we prove PSPACE-completeness for hybrid fragments including binders. We apply our findings to the problems of constraint verification for semistructured data and specification checking for mobile systems.

1 Introduction

Model checking [7] is a reasoning task in which 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 taken from some logic. Model checking then boils down to search the graph in order to *check* whether the formula is true in the *model*.

Modal and temporal logics have been successfully used as specification languages in the model checking task [8]; 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* individual or sets of states, and for dynamically creating new names for individual or sets of states. In particular, traditional modal and temporal logics are able to express (only) 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. An example of a (very simple) property violating the tree model property is the following: "the current point belongs to a cycle". An interesting question is: are there extensions of modal and tem-

Language	Complexity
$\mathrm{HL}(@, \mathbf{F})$	$\mathbf{O}(k \cdot (n+m))$
$\mathrm{HL}(@, \mathbf{F}, \mathbf{P})$	$\mathbf{O}(k \cdot (n+m))$
$\mathrm{HL}(@, \mathbf{U})$	$\mathbf{O}(k \cdot n \cdot m)$
$\mathrm{HL}(@, \mathbf{U}, \mathbf{S})$	$\mathbf{O}(k \cdot n \cdot m)$
$\mathrm{HL}(\downarrow, @)$	$\mathbf{O}(k \cdot n)$
$\mathrm{HL}(\downarrow, \mathbf{F})$	PSPACE-complete
$\mathrm{HL}(\downarrow, @, \mathbf{F})$	PSPACE-complete
$\mathrm{HL}_r(\downarrow, @, \mathbf{F})$	$\mathbf{O}(k \cdot (n+m) \cdot n^r)$
$\mathrm{HL}(\exists)$	PSPACE-complete
$\mathrm{HL}(\exists, \mathbf{F})$	PSPACE-complete
$\mathrm{HL}(\exists, @, \mathbf{F})$	PSPACE-complete
$\mathrm{HL}_r(\exists, @, \mathbf{F})$	$\mathbf{O}(k \cdot (n+m) \cdot n^{2r})$

Table 1
Complexity of model checking for hybrid logics.

poral logics violating the tree model property that are still computationally tractable?

Hybrid logics, to some extent, provide an answer to the previous question. They allow one to refer to states in a truly modal framework, and in this way they mix features from first-order logic with features from modal logic, whence the name *hybrid* logic [6]. In addition to ordinary propositional variables, hybrid languages provide a new type of atomic formulas called *nominals*. Syntactically, nominals behave like ordinary propositional variables, but they have an important semantic property: nominals are true at exactly one state in any model; in this way, we can assign a name to a state and use this name in formulas.

Nominals are just the first ingredient that sets hybrid languages apart from traditional modal and temporal languages. Additionally, hybrid languages may contain the *at operator* $@_i$ which gives *random 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 very naturally with $\downarrow$: whereas $\downarrow$ *stores* the current state of evaluation (by binding a variable to it), $@$ enables us to *retrieve* the information stored by shifting the point of evaluation in the model. Finally, the *existential binder* $\exists x.$ binds the variable name x to a state in the model. For example, the formula $\downarrow x.\mathbf{F}^+ x$, where x is a variable and $\mathbf{F}^+$ is interpreted over the transitive closure of the accessibility relation, is true if and only if the current point belongs to a cycle. Moreover, $\exists x.@_x \mathbf{F}^+ x$ holds if and only if the model contains a cycle.

Characterization (with respect to first-order correspondence theory), interpolation and computational complexity (of the satisfiability problem) for hybrid modal logics have been studied in [5]. The complexity of the satisfiability problem for hybrid temporal logics with respect to different classes of frames (all frames, transitive frames, linear frames, transitive trees) has been investigated in [3,4,9]. For a more detailed guide to the field, see the hybrid logic home page [10].

Despite their importance for modeling and specification purposes, there is a big gap in our current state of knowledge of, and in the availability of methods and tools for, hybrid languages: *model checking for hybrid languages* has hardly been explored. This is the main issue of this paper. We follow an incremental approach: we first study the problem for the basic hybrid language, that is, modal logic plus nominals and @ (we denote this fragment by $\mathrm{HL}(@, \mathbf{F})$). Then, we complicate the basic hybrid language by adding operators along both a temporal direction (past $\mathbf{P}$, until $\mathbf{U}$, since $\mathbf{S}$) and a hybrid one (binders $\downarrow$ and $\exists$). It turns out that the addition of nominals and the @ operator does not increase the complexity of the model checker. In contrast, an arbitrary use of a hybrid binder, either $\downarrow$ or $\exists$, is dangerous: the model checking problem for any hybrid logic that freely mixes a hybrid binder with temporal operators is PSPACE-complete. However, the model checker runs in exponential time with respect to the nesting degree of the binder in the formula. This means that the model checker terminates in polynomial time whenever the nesting degree of hybrid binders is fixed. Table 1 summarizes the complexity results we obtained (k is the length of the formula, n and m are respectively the number of nodes and the number of edges of the graph structure, and r is the nesting degree of hybrid binders). We apply our findings to the problems of constraint verification for semistructured data and specification checking for mobile systems.

The paper is organized as follows. Section 2 briefly introduces hybrid logics. In Section 3.1 we provide model checkers for different hybrid languages and we analyze their computational complexities. In Section 3.2 we prove PSPACE-completeness the hybrid fragments allowing binders. In Section 4 we describe a couple of applications of the present investigation. We conclude the paper in Section 5.

2 Hybrid logics

Temporal logics (TLs, for short) are algorithmically well-behaved and mathematically natural fragments of classical logics. They extend propositional logic by adding the well-known temporal operators future $\mathbf{F}$, past $\mathbf{P}$, until $\mathbf{U}$ and since $\mathbf{S}$, interpreted over the accessibility relation on the set of states, as well as their transitive closures $\mathbf{F}^+$, $\mathbf{P}^+$, $\mathbf{U}^+$ and $\mathbf{S}^+$, interpreted over the transitive closure of the accessibility relation.

Hybrid logics extend temporal logics by introducing tools for naming states and for accessing states by names. Let $\mathsf{NOM} = \{i, j, \ldots\}$ be a set of nominals and $\mathsf{WVAR} = \{x, y, \ldots\}$ be a set of state variables. The syntax of hybrid logic, in its full extension, is as follows:

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

with $i \in \mathsf{NOM}$, $x \in \mathsf{WVAR}$, $t \in \mathsf{NOM} \cup \mathsf{VAR}$. 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*.

Hybrid logic is interpreted over *hybrid Kripke structures*, that is, Kripke structures $\langle M, R, V\rangle$ where the valuation function V is such that $V(i)$ is a singleton subset of M for all nominals $i \in \mathsf{NOM}$. To give meaning to the formulas, we also need the notion of *assignment*. An assignment g is a mapping $g : \mathsf{WVAR} \to M$. 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 as follows (we omit the well-known semantics for temporal operators and for Boolean connectives). Let $\mathcal{M} = \langle M, R, V\rangle$ be a hybrid Kripke structure, $m \in M$, and g an assignment. Then,

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

The at operator $@_t$ shifts evaluation to the state named by t, where t is either a nominal or a variable. The downarrow binder ${\downarrow}x$ binds the state variable x to the *current* state (the state where evaluation is being performed), while the existential binder $\exists x$ binds the state variable x to *some* state in the model. Notice that $\downarrow$ and $\exists$ do not shift evaluation away from the current state. In the following, we denote by $\mathrm{HL}(O_1, \ldots, O_n)$ the hybrid language with hybrid and temporal operators $O_1, \ldots, O_n$. As an example of hybrid formulas, the until operator can be written in hybrid logic as $\alpha\mathbf{U}\beta = {\downarrow}x.\mathbf{F}(\beta \wedge \mathbf{H}(\mathbf{P}x \to \alpha))$, and the since as $\alpha\mathbf{S}\beta = {\downarrow}x.\mathbf{P}(\beta \wedge \mathbf{G}(\mathbf{F}x \to \alpha))$. Moreover, the past operator is $\mathbf{P}\alpha = {\downarrow}x.\exists y.@_y(\mathbf{F}x \wedge \alpha)$. Finally, the downarrow binder is a particular case of the existential binder: ${\downarrow}x.\alpha = \exists x.(x \wedge \alpha)$.

3 Model checking for hybrid logics

3.1 Upper bounds

In this section we provide model checkers for different hybrid logics and we analyze their worst-case behaviour.

A hybrid Kripke structure $\mathcal{M} = \langle M, R, V\rangle$ is *finite* if M is finite. The model checking problem for hybrid logic is as follows: Given a finite hybrid Kripke structure $\mathcal{M}$, an assignment g, a state m in $\mathcal{M}$, and a hybrid formula φ, does $\mathcal{M}, g, m \models \varphi$? In the following we describe a model checker `MCLITE` for HL($@, \mathbf{F}, \mathbf{P}, \mathbf{U}, \mathbf{S}$). It receives a hybrid model $\mathcal{M} = \langle M, R, V\rangle$, an assignment g, and a hybrid formula φ in the considered language and, after termination, every state in the model is labelled with the subformulas of φ that hold at that point. The algorithm uses a *bottom-up strategy*: it examines the subformulas of φ in increasing order with respect to their lengths, from simple formulas to more complicated ones, until φ itself has been checked.

We need some auxiliary notation. Let R be an accessibility relation and R^- the inverse of R: R^-vu if and only of 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 called successors of w, while those belonging to $R^{-1}(w)$ are called 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 $|\varphi|$, is the number of operators (boolean, temporal and hybrid) of φ plus the number of atoms (propositions, nominals and variables) of φ. Let $sub(\varphi)$ be the set of subformulas of φ. Notice that $|sub(\varphi)| = |\varphi|$.

The model checker `MCLITE` updates a table L of size $|\varphi| \times |M|$ whose elements are bits. Initially, $L(\alpha, w) = 1$ if and only if α is an atomic letter in $sub(\varphi)$ 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 $\alpha \in sub(\varphi)$. Given $\alpha \in sub(\varphi)$ 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 formulas $\beta \in sub(\varphi)$ such that $L(\beta, w) = 1$. `MCLITE` uses three subroutines named $\mathtt{MC}_{\mathbf{F}}$, $\mathtt{MC}_{\mathbf{U}}$ and $\mathtt{MC}_{@}$ in order to check subformulas of the form $\mathbf{F}\alpha$, $\alpha\mathbf{U}\beta$ and $@_t\alpha$, respectively. The pseudocode for them is below.

Proposition 3.1 (Correctness of subroutines) *Let $\mathcal{M} = \langle M, R, V\rangle$ be a hybrid model, g be an assignment, α and β be hybrid formulas, and t be 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,*

(i) *after termination of $\mathtt{MC}_{\mathbf{F}}(\mathcal{M}, g, \alpha)$, we have that, for every $w \in M$, $L(\mathbf{F}\alpha, w) = 1$ if and only if $\mathcal{M}, g, w \models \mathbf{F}\alpha$;*

(ii) *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 $\mathcal{M}, g, w \models \alpha\mathbf{U}\beta$;*

(iii) *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 $\mathcal{M}, g, w \models @_t\alpha$.*

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 $\mathtt{MC}_{@}$ runs in $O(n)$. As for $\mathtt{MC}_{\mathbf{U}}$, for every $w \in L(\beta)$, the procedure backward visits twice the states reachable in 2 steps. Both the visits cost $O(m)$. Since there are $O(n)$ states in $L(\beta)$, $\mathtt{MC}_{\mathbf{U}}$ runs in $O(nm)$ time.

```
Procedure $\mathtt{MC}_{\mathbf{F}}(\mathcal{M}, g, \alpha)$
for $w \in L(\alpha)$ do
    for $v \in R^{-1}(w)$ do
        $L(\mathbf{F}\alpha, w) \leftarrow 1$
    end for
end for
```

```
Procedure $\mathtt{MC}_{@}(\mathcal{M}, g, t\alpha)$
let $\{w\} = [V, g](t)$
if $L(\alpha, w) = 1$ then
   for $v \in M$ do
       $L(@_t\alpha, v) \leftarrow 1$
   end for
end for
```

```
Procedure $\mathtt{MC}_{\mathbf{U}}(\mathcal{M}, g, \alpha, \beta)$
for $w \in M$ do
    unmark($w$)
end for
for $w \in L(\beta)$ do
    for $v \in R^{-1}(w)$ do
        if $L(\alpha, w) = 0$ then
           for $u \in R^{-1}(v)$ do
               mark($u$)
           end for
        end if
    end for
    for $v \in R^{-1}(w) \cup R^{-2}(w)$ do
        if marked($v$) then
           unmark($v$)
        else
           $L(\alpha\mathbf{U}\beta, v) \leftarrow 1$
        end if
    end for
end for
```

Using the subroutines $\mathtt{MC}_{\mathbf{F}}$, $\mathtt{MC}_{\mathbf{U}}$, and $\mathtt{MC}_{@}$ we can implement a bottom-up model checker `MCLITE` for the hybrid language $\mathrm{HL}(@, \mathbf{F}, \mathbf{P}, \mathbf{U}, \mathbf{S})$. Past temporal operators are handled by feeding the above subroutines with the inverse model $\mathcal{M}^{-}$. Boolean connectives are treated as usual. The code is omitted for space reasons. The correctness of `MCLITE` follows from the correctness of its subroutines (Proposition 3.1) and the semantics of past operators $\mathbf{P}$ and $\mathbf{S}$.

Theorem 3.2 (Correctness of `MCLITE`) *Let $\mathcal{M} = \langle M, R, V\rangle$ be a hybrid model, g an assignment, and φ a formula in $\mathrm{HL}(@, \mathbf{F}, \mathbf{P}, \mathbf{U}, \mathbf{S})$. Then, after the termination of `MCLITE`$(\mathcal{M}, g, \varphi)$, we have that, for every $w \in M$ and for every $\alpha \in sub(\varphi)$, it holds that $L(\alpha, w) = 1$ if and only if $\mathcal{M}, g, w \models \alpha$.*

Now for the computational complexity.

Theorem 3.3 (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 in $\mathrm{HL}(@, \mathbf{F}, \mathbf{P}, \mathbf{U}, \mathbf{S})$ of length k. Then, the model checker `MCLITE`$(\mathcal{M}, g, \varphi)$ terminates in $O(k \cdot n \cdot m)$ time. If φ contains neither until nor since operators, then the complexity is $O(k \cdot (n + m))$. If φ contains no temporal operators, the complexity is $O(k \cdot n)$.*

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 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 then use the model checker MCLITE on the transitive model.

We now move to hybrid languages with binders ($\downarrow$ and $\exists$). The bad news is that the efficient bottom-up strategy used in MCLITE does not work for hybrid languages with binders. 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 bottom-up, that is, from simple formulas to more complex ones, we initially have to check the subformula x. However, at this stage, do not have enough information to check x, and hence we cannot label the states of the model with x. On the other hand, a (inefficient) *top-down* strategy works: to check $\mathbf{G}{\downarrow}x.\mathbf{F}x$ at the current state m, check whether the subformula ${\downarrow}x.\mathbf{F}x$ holds at every successor of m. To check ${\downarrow}x.\mathbf{F}x$ at a successor n, assign the value n to the variable x and check $\mathbf{F}x$ at n. Now we have enough information to verify $\mathbf{F}x$, and hence we can use MCLITE's bottom-up strategy. The sketched strategy combines top-down and, whenever possible, bottom-up reasoning; it has been implemented in the recursive model checker MCFULL for $\mathrm{HL}(@, \downarrow, \exists, \mathbf{F})$. For space reasons, we only present the code for the auxiliary procedures Check$_{\mathbf{F}}$, Check$_{@}$, Check$_{\downarrow}$ and Check$_{\exists}$, that handle the cases of subformulas of the form $\mathbf{F}\alpha$, $@_t\alpha$, ${\downarrow}x.\alpha$, and $\exists x.\alpha$, respectively. These procedures use the subroutines Lite(α) to check whether or not α belongs to $\mathrm{HL}(@, \mathbf{F})$, and Clear($L, x$) to reset all the values of $L(\alpha)$, for any α containing the variable x.

```
Procedure Check$_{\mathbf{F}}$($\mathcal{M}, g, \alpha$)
if Lite($\alpha$) then
  MCLITE($\mathcal{M}, g, \mathbf{F}\alpha$)
else
  MCFULL($\mathcal{M}, g, \alpha$)
  for $w \in M$ do
      for $v \in R(w)$ do
          if $v \in L(\alpha)$ then
              $L(\mathbf{F}\alpha, w) \leftarrow 1$
          end if
      end for
  end for
end if
```

```
Procedure Check$_{@}$($\mathcal{M}, g, t, \alpha$)
if Lite($\alpha$) then
  MCLITE($\mathcal{M}, g, @_t\alpha$)
else
  MCFULL($\mathcal{M}, g, \alpha$)
  let $\{v\} = [V, g](t)$
  if $v \in L(\alpha)$ then
    for $w \in M$ do
      $L(@_t\alpha, w) \leftarrow 1$
    end for
  end if
end if
```

```
Procedure Check_↓($\mathcal{M}, g, x, \alpha$)
for $w \in M$ do
    $g(x) \leftarrow w$
    MCFULL($\mathcal{M}, g, \alpha$)
    if $w \in L(\alpha)$ then
        Clear($L, x$)
        $L(\downarrow x.\alpha, w) \leftarrow 1$
    else
        Clear($L, x$)
    end if
end for
```

```
Procedure Check_∃($\mathcal{M}, g, x, \alpha$)
for $w \in M$ do
    for $v \in M$ do
        $g(x) \leftarrow v$
        MCFULL($\mathcal{M}, g, \alpha$)
        if $w \in L(\alpha)$ then
            Clear($L, x$)
            $L(\exists x.\alpha, w) \leftarrow 1$
        else
            Clear($L, x$)
        end if
    end for
end for
```

The correctness of MCFULL follows from the correctness of MCLITE and the semantics of $\mathrm{HL}(@, \downarrow, \exists, \mathbf{F})$.

Theorem 3.4 (Correctness of MCFULL) *Let $\mathcal{M} = \langle M, R, V\rangle$ be a hybrid model, g an assignment, and φ a hybrid formula in $\mathrm{HL}(@, \downarrow, \exists, \mathbf{F})$. Then, after termination of MCFULL($\mathcal{M}, g, \varphi$), we have that:*

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

Now for the computational complexity.

Theorem 3.5 (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. Then,*

- *if φ is a hybrid formula in $\mathrm{HL}(@, \downarrow, \mathbf{F})$ of length k and r the nesting degree of $\downarrow$ in φ, then MCFULL($\mathcal{M}, g, \varphi$) terminates in time $O(k \cdot (n + m) \cdot n^r)$;*
- *if φ is a hybrid formula in $\mathrm{HL}(@, \exists, \mathbf{F})$ of length k and r the nesting degree of $\exists$ in φ, then MCFULL($\mathcal{M}, g, \varphi$) terminates in time $O(k \cdot (n + m) \cdot n^{2r})$.*

In both cases MCFULL uses a polynomial amount of space.

3.2 Lower bounds

The natural question is: can we do better that the upper bounds proved above? It is well known that model checking for first-order logic is PSPACE-complete. Since $\mathrm{HL}(\exists, @, \mathbf{F})$ is as expressive as first-order logic [6], model checking for $\mathrm{HL}(\exists, @, \mathbf{F})$ is PSPACE-complete as well. However, what about fragments of $\mathrm{HL}(\exists, @, \mathbf{F})$? The question is particulary interesting for the fragment $\mathrm{HL}(\downarrow, @, \mathbf{F})$, since it corresponds to the bounded fragment of the first-order correspondence language [3]. Unfortunately, the model checking problem for $\mathrm{HL}(\downarrow, @, \mathbf{F})$ and hence for the bounded fragment is still PSPACE-hard, even without @, nominals and propositions.

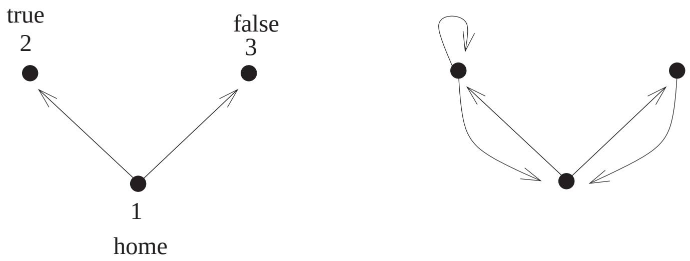

Fig. 1. Embedding QBF into model checking for hybrid logics.

Theorem 3.6 *Model checking for the pure nominal-free fragment of* HL$(\downarrow, \mathbf{F})$ *is PSPACE-complete.*

Proof. PSPACE-membership follows from Theorem 3.5. To prove PSPACE-hardness, we embed Quantified Boolean Formulas (QBF) into the model checking problem for the pure nominal-free fragment of HL$(\downarrow, \mathbf{F})$. We do it 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 by complicating the model.

Recall that an instance of QBF is $\Psi = Q_1 x_1 \ldots Q_n x_n . \alpha(x_1, \ldots, x_n)$, 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 $\mathcal{M}$ be the model is depicted in Figure 1, left 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 $\mathcal{M}, 1 \models \varphi_\Psi$.

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 1, right 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 following translation:

$$\begin{array}{ll|ll}
\tau(x) & = \mathbf{F}(x \wedge {\downarrow} y . \mathbf{F} y) & \tau(\neg x) & = \mathbf{F}(x \wedge {\downarrow} y . \mathbf{G} \neg y) \\
\tau(\alpha_1 \wedge \alpha_2) & = \tau(\alpha_1) \wedge \tau(\alpha_2) & \tau(\alpha_1 \vee \alpha_2) & = \tau(\alpha_1) \vee \tau(\alpha_2) \\
\tau(\exists x . \alpha) & = \mathbf{F} {\downarrow} x . \mathbf{F}({\downarrow} y . \mathbf{G} \neg y \wedge \tau(\alpha)) & \tau(\forall x . \alpha) & = \mathbf{G} {\downarrow} x . \mathbf{F}({\downarrow} y . \mathbf{G} \neg y \wedge \tau(\alpha))
\end{array}$$

It holds that Ψ is true if and only if $\mathcal{M}, 1 \models \tau(\Psi)$. □

Theorem 3.6 still holds if we replace $\mathbf{F}$ by $\mathbf{F}^+$:

Theorem 3.7 *Model checking for the pure nominal-free fragments of* HL$(\downarrow, \mathbf{F}^+)$

is PSPACE-complete.

Proof. PSPACE-membership follows from Theorem 3.5. To prove PSPACE-hardness, we embed QBF into the model checking problem for the pure nominal-free fragments of 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 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 $\mathcal{M}, 1 \models \varphi_\Psi$. □

If we replace $\downarrow$ by $\exists$, hardness holds even without temporal operators.

Theorem 3.8 *Model checking for the pure nominal-free fragment of* HL($\exists$) *is PSPACE-complete.*

Proof. PSPACE-membership follows from Theorem 3.5. To prove PSPACE-hardness, we embed QBF into the model checking problem for the pure nominal-free fragments of 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 $x_1, \ldots, x_n$. Note that Ψ is a pure nominal-free formula in HL($\exists$). Let $\mathcal{M}$ be the unlabelled model based on the frame $\langle \{1, 2\}, \emptyset \rangle$. We have that Ψ is true if and only if $\mathcal{M}, 1 \models \Psi$. □

4 Hybrid logic in action

We describe a couple of scenarios in which the problem of model checking for hybrid logic is relevant.

4.1 Constraint evaluation in semistructured data

In this section we are motivated by the task of verification of constraints for semistructured data [1]. Semistructured data are data without a regular schema. It is generally agreed that the appropriate data model for semistructured data is an *edge-labelled rooted graph*, where the nodes corresponds to objects, and the edges corresponds to attributes. Each edge is labelled with the attribute name. Nodes may also be labelled, with both object identifiers and values. By the way, structured data are special cases of semistructured data having a regular structure. Constraints in semistructured data are path conditions over the data graph representation of the form: every node reachable through a path in p is also reachable through a path in q, where p and q are sets of paths. They correspond and generalize integrity constraints in structured data. Consider the following ODMG schema, which is borrowed from [1].

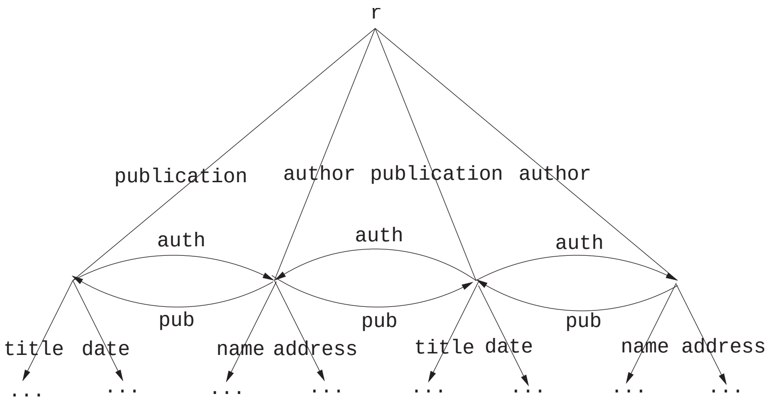

Fig. 2. A graph representation of semistructured data.

```
interface Publication
  extent publication
  { attribute String title;
     attribute Date date;
     relationship set<Author> auth inverse Author::pub;
  }

interface Author
  extent author
  { attribute String name;
     attribute String address;
     relationship set<Publication> pub inverse Publication::auth;
  }
```

It describes the entities `publication` and `author`. A publication has attributes `title`, `date` and a relationship `auth` which associates a set of authors to the publication. An author has attributes `name`, `address` and a relationship `pub` which associates a set of publications to the author. Besides this information, the schema specifies the following constraints:

(i) for any publication `p`, the set `p.auth` is a subset of the set `author`;

(ii) for any author `a`, the set `a.pub` is a subset of the set `publication`;

(iii) for any publication `p` and for any author `a` in `p.auth`, `p` is a member of `a.pub`;

(iv) for any author `a` and for any publication `p` in `a.pub`, `a` is a member of `p.auth`.

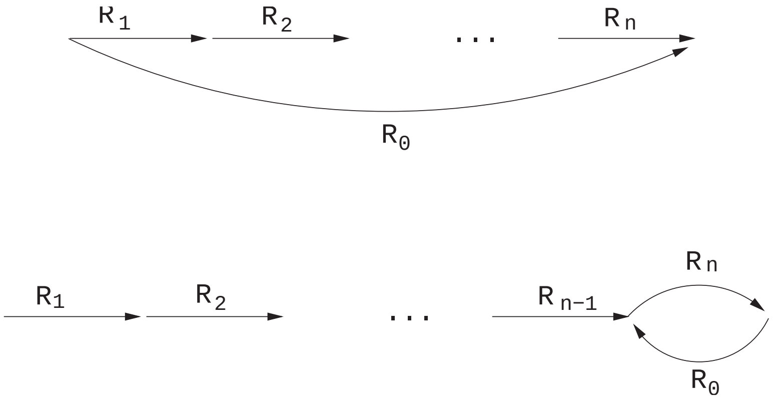

Fig. 3. Inclusion constraint (top) and inverse relationship constraint (bottom).

Conditions 1 and 2 are called inclusion constraints, and conditions 3 and 4 are called inverse relationship constraints. Of course, there is nothing semistructured in the example above. However, we will ignore in the following the structure of data and focus on constraints only. A graph representation of the ODMG schema above is depicted in Figure 2. With the aid of the graph representation of the data, the inclusion constraints above can be reformulated as follows: any node that is reached from the root by following a `publication` edge followed by a `auth` edge can also be reached from the root following a `author` edge. Similarly, any node that is reached from the root by following a `author` edge followed by a `pub` edge can also be reached from the root following a `publication` edge. The inverse relationships constraints above can be reformulated as follows: for every node x that is reached from the root by following an `publication` edge, if x can reach a node y by following an `auth` edge then y can reach x by following an `pub` edge. Similarly, for every node x that is reached from the root by following a `author` edge, if x can reach a node y by following an `pub` edge then y can reach x by following an `auth` edge.

More generally, let $R_0, \dots R_n$ be binary relations. An *inclusion constraint* $IC(\langle R_1, \dots, R_n \rangle, R_0)$ is as follows:

> Any node that is reached from the root by following an R_1 edge, followed by an R_2 edge, ..., followed by an R_n edge can also be reached from the root following a R_0 edge (see top of Figure 3).

An *inverse relationship constraint* $IRC(\langle R_1, \dots, R_n \rangle, R_0)$ is as follows:

> For every node x that is reached from the root by following an R_1 edge, followed by an R_2 edge, ..., followed by an R_{n-1}, if x can reach a node y by following an R_n edge then y can reach x by following an R_0 edge (see bottom of Figure 3).

In the following, a constraint will be either an inclusion constraint or an inverse relationship constraint. The *constraint evaluation problem* is as follows:

> Given a finite edge-labelled rooted graph $\mathcal{G}$ and a Boolean combination C

of constraints on $\mathcal{G}$, is C true in $\mathcal{G}$ at the root?

Constraint evaluation is important to maintain the integrity of data and for query optimization as well. Indeed, two query plans may be semantically equivalent in the assumption that some constraints hold, but one of them may be computationally more efficient.

4.1.1 Formalization in hybrid logic

In this section we formalize in our setting the constraint evaluation. We will show that, using the algorithms and methods we proposed in this paper, we are able to solve the constraint evaluation problem in polynomial time.

We can naturally encode both the inclusion and the inverse relationship constraints in hybrid logic. Let $C = IC(\langle R_1, \ldots, R_t\rangle, R_0)$ be an inclusion constraint. Let $\mathbf{F_j}$ and $\mathbf{P_j}$ be the future and past modalities interpreted over relation R_j, and let r be a nominal denoting the root of the graph. Then C is captured by the hybrid formula:

$$\varphi_C = @_r\mathbf{G_1}\ldots\mathbf{G_t}\mathbf{P_0}r.$$

Let $C = IRC(\langle R_1, \ldots, R_s\rangle, R_0)$ be an inverse relationship constraint. It can be expressed by

$$\psi_C = @_r\mathbf{G_1}\ldots\mathbf{G_{s-1}}{\downarrow}x.\mathbf{G_s}\mathbf{F_0}x.$$

It follows that the constraint evaluation problem can be reduced to the model checking problem for hybrid logic. More precisely, given a finite edge-labelled rooted graph $\mathcal{G}$ and a Boolean combination C of constraints $C_1, \ldots, C_n$ on $\mathcal{G}$, we can verify whether C is true in $\mathcal{G}$ at the root r by checking $\mathcal{G}, r \Vdash \Phi$, where Φ is the Boolean combination C of formulas φ_{C_i} and ψ_{C_i} as above.

Whenever only inclusion constraints are present, the model checking task $\mathcal{G}, r \Vdash \Phi$ can be solved in time $O(k \cdot (n+m))$, where k in the length of Φ, and n and m are the number of nodes and the number of edges of $\mathcal{G}$, respectively. This boils down to $k \cdot n$ in the reasonable assumption that the representation graph is sparse. Whenever both inclusion and inverse relationships constraints are present, the model checking task $\mathcal{G}, r \Vdash \Phi$ can be solved in time $O(k \cdot n \cdot (n+m))$, that is $k \cdot n^2$ whenever the graph is sparse.

It is worth remarking that in the literature the constraint evaluation problem has been addressed in the more general setting where *regular expressions* are involved in constraints. Given regular expressions p and q, a regular inclusion constraint is:

> Any node that is reached from the root by a path whose labels form a word in p can also be reached from the root by a path whose labels form a word in q.

Similarly, given regular expressions r, p and q, a regular inverse relationship constraint is:

> For every node x that is reached from the root by a path whose labels form

a word in r, if x can reach a node y by a path whose labels form a word in p then y can reach x by a path whose labels form a word in q.

It is clear that in this paper we considered a restricted version of the above constraints in which regular expressions boil down to (sequences of) atomic labels. However, as we showed, this restricted version is both interesting and computationally tractable.

The regular constraint evaluation problem is still polynomial [2]. However, the dynamic logic presented in [2] is not expressive enough to capture inverse relationship constraints, and the solution given there to the constraint evaluation problem is an ad hoc algorithm.

4.2 Verification of mobile reactive systems

Our second application concerns the verification of specifications for mobile systems. Roughly speaking, a mobile system is a program that exploits a wide area computational infrastructure like the World-Wide-Web. It consists of several spatially distributed and dynamically connected components that may go mobile among locations. From a technical point of view, we can isolate two main ingredients of mobility: the notion of location and the change of spatial configurations of locations over time. The notion of location can be naturally expressed in hybrid logics with nominals to give names to locations, and with the @ operator to access a location through its name. Mobile requirements may be encoded in hybrid logics and model checking may be used to verify whether the mobile system enjoys its specifications. For space reasons we do not describe this application in full detail.

5 Conclusion

We investigated the model checking problem for hybrid logics. We found that the addition of nominals and the @ operator does not increase the complexity of the model checking task. The reason is that the resulting language still enjoys the following subformula property: to check the truth of a formula φ, we can check the truth of subformulas of φ, and then verify the truth of φ taking advantage of the gained information about subformulas. The subformula property is important for model checking, since it permits to apply an efficient bottom-up procedure that works on subformulas in increasing length order up to the formula to be checked. In contrast, whenever hybrid binders are present in the language, the truth of a subformula containing a variable may depend on the binding of the variable, which is external to the subformula. We may not have enough information to check subformulas before knowing how to bind its variables. Therefore, a pure bottom-up strategy does not work, and a costly top-down evaluation in needed. We used a model checking strategy that combines bottom-up and top-down reasoning. 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.

Our work pointed out once more the difference in complexity between $\downarrow$ and $\exists$. While $\downarrow$ binds a variable to the *current state*, $\exists$ binds a variable to *some state*. This makes a difference from a model checking point of view, that reflects in the running times of the corresponding model checkers (see Theorem 3.5). Moreover, $\exists$ is dangerous alone, while $\downarrow$ is dangerous only in presence of temporal operators. In fact, the formula $\exists x.\varphi$ is equivalent to $\downarrow y.\mathbf{F}\downarrow x.@_y\varphi$, where $\mathbf{F}$ is interpreted over a universal accessibility relation.

Finally, we think that our work may be relevant in any setting in which properties to check refer to the *graph* like nature of the underlying structure, e.g. properties about loops.

References

[1] Abiteboul, S., P. Buneman and D. Suciu, “Data on the Web: from relations to semistructured data and XML,” Morgan Kaufmann Publishers, Los Altos, CA 94022, USA, 2000, xiii + 257 pp.

[2] Alechina, N., S. Demri and M. de Rijke, *A modal perspective on path constraints*, Journal of Logic and Computation (2002).

[3] Areces, C., P. Blackburn and M. Marx, *A road-map on complexity for hybrid logics*, in: J. Flum and M. Rodriguez-Artalejo, editors, *Computer Science Logic*, Lecture Notes in Computer Science **1683** (1999), pp. 307–321.

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

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

[6] Blackburn, P. and J. Seligman, *What are hybrid languages?*, in: M. Kracht, M. de Rijke and H. Wansing, editors, *Advances in Modal Logic, Volume 1*, CSLI Publications, Stanford, California, 1998 pp. 41–62.

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

[8] Clarke, E. M. and H. Schlingloff, *Model checking*, in: A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, Elsevier Science, 2001 pp. 1635–1790.

[9] Franceschet, M., M. de Rijke and H. Schlingloff, *Hybrid logics on linear frames: expressivity and complexity*, in: *Proceedings of the International Workshop on Temporal Representation and Reasoning - International Conference on Temporal Logic* (2003).

[10] HyLo: The Hybrid Logic home page. URL: `http://www.hylo.net`.