Available online at [www.sciencedirect.com](http://www.sciencedirect.com/)



[Electronic Notes in Theoretical Computer Science 278 (2011) 115–128](http://dx.doi.org/10.1016/j.entcs.2011.10.010)

[www.elsevier.com/locate/entcs](http://www.elsevier.com/locate/entcs)

Expressiveness of Hybrid Temporal Logic on Data Words

Ahmet Kara[1](#_bookmark0) *,*[2](#_bookmark0) Thomas Schwentick[1](#_bookmark0) *,*[3](#_bookmark0)

*TU Dortmund, Germany*

Abstract

Hybrid temporal logic (HTL) on data words can be considered as an extension of the logic LTL↓ introduced

by Demri and Lazic [[3](#_bookmark18)]. The paper compares the expressive power of HTL on data words with that of LTL↓. It is shown that there are properties of data words that can be expressed in HTL with two variables but not in LTL↓. On the other hand, every property that can be expressed in HTL with one variable can also

be expressed in LTL↓ with one variable. The paper further studies the succinctness of HTL in comparison with LTL↓ and shows that the number-of-variables hierarchy of HTL is infinite.

*Keywords:* Hybrid logics, temporal logics, data words.

# Introduction

In this paper, a data word is a finite sequence of positions which carry a data value and a set of propositions. Logics on data words have been investigated a lot in recent years, e.g., in [[2](#_bookmark19),[3](#_bookmark18)] and many follow-up papers. In this paper we consider hybrid temporal logic on data words. As an example, the formula *ϕ* = G(*p* ∧ ↓*x.*F(*q* ∧ ∼*x*)) expresses that for every position carrying the proposition *p* there is a position in the future that has the same data value and carries proposition *q*. The quantifier

↓*x* binds variable *x* to the current position and ∼*x* compares the current data value with the value at the position bound to *x*. The logic further allows formulas as @*x.χ* (evaluate *χ* at position *x*) and *x* (true if *x* is the current position).

Hybrid temporal logic was first considered in [[13](#_bookmark30)], and intensively studied on linear structures, e.g., in [[1](#_bookmark20),[9](#_bookmark26),[15](#_bookmark32)]. It has been noted before that the logic LTL↓ on data words, introduced in [[3](#_bookmark18)] is essentially a hybrid temporal logic [[4](#_bookmark21),[5](#_bookmark22),[19](#_bookmark36)]. In

1 We acknowledge the financial support by the German DFG under grant SCHW 678/4-1.

2 Email: [ahmet.kara@cs.uni-dortmund.de](mailto:ahmet.kara@cs.tu-dortmund.de)

3 Email: [thomas.schwentick@cs.tu-dortmund.de](mailto:thomas.schwentick@cs.tu-dortmund.de)

1571-0661 © 2011 Elsevier B.V. Open access under [CC BY-NC-ND license](http://creativecommons.org/licenses/by-nc-nd/3.0/).

[doi:10.1016/j.entcs.2011.10.010](http://dx.doi.org/10.1016/j.entcs.2011.10.010)

fact, LTL↓ can be considered as the syntactical fragment of hybrid temporal logic without formulas of the forms @*x.χ* and *x*. Formally, in LTL↓, variables are bound to data values instead of positions. However, the difference does not matter as long as the only way to refer to a variable *x* is through an atomic formula ~*x*. Thus, formula *ϕ* above can be seen as a LTL↓ formula.

We study hybrid temporal logics with future *and* past temporal operators (HTL)

on data words and compare its expressive power with that of LTL↓.

There is a correspondence between HTL and LTL↓ on one hand and automata models for data words, on the other hand. As shown in [[4](#_bookmark21)], every property express- ible in LTL↓ can be decided by an alternating register automaton. The logic HTL, on the other hand, is captured by pebble automata. This follows directly from the fact that HTL can only express first-order properties and that even deterministic one-way pebble automata can express [4](#_bookmark1) all first-order properties on data words [[12](#_bookmark29)].

The relationship between LTL↓ and alternating register automata together with previous results in [[12](#_bookmark29)] immediately yields a separation between LTL↓ and HTL. In- deed, in [[12](#_bookmark29)] a first-order expressible property of data words was defined that cannot be decided by alternating register automata and therefore cannot be expressed by LTL↓. On the other hand, it is not hard to show that HTL can express all first-order properties of data words and thus also this particular property. We strengthen this result by showing that a similar property that still cannot be expressed in LTL↓ can be expressed by an HTL formula with only two variables, thus establishing LTL↓ /≤ HTL2.

Interestingly, the difference between HTL and LTL↓ vanishes when we restrict formulas to one variable. Our main result shows that HTL1 = LTL↓.

1

We further show that

* HTL1-formulas can be exponentially more succinct than LTL↓-formulas and HTL- formulas can even be non-elementarily more succinct than LTL↓-formulas; and that
* the variable hierarchy of HTL on data strings is infinite. To this end, we show that Rossman’s result that the variable hierarchy for first-order logic on finite ordered graphs is strict can be carried over to data strings.

Most of our results also hold for infinite data words (cf. Section [5](#_bookmark17)).

We already mentioned related work above. The paper is organized as follows. After the preliminaries (Section [2](#_bookmark2)) we show in Section [3](#_bookmark3) the results on HTL with at least two variables and in Section [4](#_bookmark13) the results on HTL1. We conclude in Section [5](#_bookmark17). Due to lack of space we do not present all proofs in full detail. In particular, some correctness proofs for translations of formulas are missing. However, we always tried to design the translated formulas so that the correctness can be verified easily by the reader.

We thank Thomas Zeume and an anonymous reviewer for many useful sugges- tions.

4 Formally, this was only shown for finite data words over an empty proposition set in [[12](#_bookmark29)]. However, the proof simply goes through for data words.

# Preliminaries

Data Words. Let Σ be a finite set of propositions and D an infinite set of data values. A *string* over Σ is a finite sequence of elements from 2Σ. A *data word* over Σ is a finite sequence of elements from (2Σ × D). A data word is denoted by

*w* = *P*1 ··· *Pn* , where *P*1*,... , Pn* ∈ 2Σ and *d*1*,... , dn* ∈ D. If *p* ∈ *Pi* we say that *i* is

*d*1 *dn*

*labelled* with *p* and that *i* is a *p-position*. We call *di the data value of position i*. If

*Pi* is a singleton set {*p*} we write *p* instead of {*p*} .

*d d*

Logics. *Hybrid temporal logic* (HTL) on data words over Σ is an extension of the

linear temporal logic LTL (see e.g. [[7](#_bookmark24)]) by past operators and variables. The syntax of HTL is as follows.

*ϕ* ::= *p* | *x* | ↓*x.ϕ* | ~*x* | @*x.ϕ* | X*ϕ* | *ϕ*U*ϕ* | X−*ϕ* | *ϕ*U−*ϕ* | ¬*ϕ* | *ϕ* ∧ *ϕ*

where *p* ∈ Σ and *x* is a variable from an infinite set VAR of variables.

As usual, we consider the other logical operators ∨, → and ↔ and the other temporal operators F*ϕ* := TU*ϕ*, F−*ϕ* := TU−*ϕ*, G*ϕ* := ¬F¬*ϕ* and G−*ϕ* := ¬F−¬*ϕ* as abbreviations. The operators X*,* U*,* F*,* G*,* X−*,* U−*,* F−*,* G− are called *temporal op- erators*. We refer to the operators X*,* U*,* F*,* G as *future operators* and to the latter four as *past operators*.

For a data word *w* with |*w*| = *n* an *assignment* is a mapping *g* : VAR →

{1*,... , n*}. For an assignment *g*, a variable *x* and a position *i* in *w*, *gx* denotes the mapping defined by *gx*(*x*) := *i* and *gx*(*y*) := *g*(*y*) for all *y* /= *x*. The semantics of a

*i*

*i* *i*

HTL formula over Σ is defined with respect to a data word *w* = *P*1 ··· *Pn* over Σ, a

position *i* in *w* and an assignment *g* for *w*.

* + *w, g, i* |= *p* if *p* ∈ *Pi*
  + *w, g, i* |= ↓*x.ϕ* if *w, gx,i* |= *ϕ*

*i*

* + *w, g, i* |= *x* if *g*(*x*)= *i*

*d*1 *dn*

* *w, g, i* |= ~*x* if *di* = *dg*(*x*)
* *w, g, i* |= @*x.ϕ* if *w, g, g*(*x*) |= *ϕ*
* *w, g, i* |= ¬*ϕ* if *w, g, i* |= *ϕ*
  + *w, g, i* |= *ϕ* ∧ *ψ* if *w, g, i* |= *ϕ* and *w, g, i* |= *ψ*
  + *w, g, i* |= X*ϕ* if *i < n* and *w, g, i* +1 |= *ϕ*
  + *w, g, i* |= *ϕ*U*ψ* if there is a *j* with *i* ≤ *j* ≤ *n* and *w, g, j* |= *ψ* and

*w, g, k* |= *ϕ* for all *k* with *i* ≤ *k < j*

* + *w, g, i* |= X−*ϕ* if *i >* 1 and *w, g, i* − 1 |= *ϕ*
  + *w, g, i* |= *ϕ*U−*ψ* if there is a *j* with 1 ≤ *j* ≤ *i* and *w, g, j* |= *ψ* and

*w, g, k* |= *ϕ* for all *k* with *j < k* ≤ *i*

If at most one variable *x* occurs in *ϕ* and *g*(*x*) = *j*, we also write *w, j, i* |= *ϕ* instead of *w, g, i* |= *ϕ*. We say that *a data word w satisﬁes a formula ϕ* (denoted as *w* |= *ϕ*) if *w, g,* 1 |= *ϕ* where *g*(*x*) := 1 for all *x* ∈ VAR.

The notions of *bound* and *free* variable occurrences and of *closed formulas* are defined as usual. We say that two formulas *ϕ*1 and *ϕ*2 are *equivalent* (denoted as *ϕ*1 ≡ *ϕ*2) if *w, g, i* |= *ϕ*1 if and only if *w, g, i* |= *ϕ*2 for all data words *w*, all

assignments *g* and all positions *i* of *w*. The formulas are *initially equivalent* if for all data words *w* and assignments *g* it holds *w, g,* 1 |= *ϕ*1 if and only if *w, g,* 1 |= *ϕ*2. For two logics L1 and L2 we write L1 ≤ L2 if for every closed formula *ϕ*1 of L1 there is an equivalent formula *ϕ*2 in L2. Furthermore, L1 ≡ L2 if L1 ≤ L2 and L2 ≤ L1.

Freeze LTL (LTL↓) ([[3](#_bookmark18)]) is the fragment of HTL that does not allow (sub-) formulas of the form *x* and @*x.ϕ*. The fragments of HTL and LTL↓ where at most *k* variables are allowed, are denoted as HTL*k* and LTL↓, respectively. For a set O of temporal operators and a logic L, we denote by L(O) the fragment of L in which only operators from O are used. The extension of LTL by past operators is denoted as PLTL.

*k*

# HTL with more than one variable

In this section, we consider HTL without a bound on the number of variables. We first show that HTL is more expressive than LTL↓ and that it actually only needs two variables to express a property that LTL↓ cannot express. Furthermore, we show that the variable hierarchy for HTL is infinite and that HTL is non-elementarily more succinct than LTL↓.

* 1. *Expressiveness*

The expressive power of HTL on data words coincides with the expressive power of first-order logic (FO). Here, first-order formulas can use the atomic formulas *p*(*x*) (stating that proposition *p* holds at position *x*), *x < y* (stating that position *y* is to the right of position *x*) and *x* ~ *y* (stating that *x* and *y* carry the same data value).

Theorem 3.1 HTL ≡ FO *on data words.*

Proof. For every *k* ≥ 1 every HTL*k*-formula can be translated into a first-order formula with at most *k* + 3 variables by a standard translation (similarly as in, e.g. [[9](#_bookmark26)]). The translation of first-order formulas into HTL-formulas is also along

standard lines. A first-order formula *ϕ* can be translated into a HTL-formula *ϕ*j as follows (we omit the Boolean cases).

* + - (∃*xψ*)j = FF—↓*x.ψ*j
    - (*x* = *y*)j = FF—(*x* ∧ *y*)
    - (*x < y*)j = @*x.*XF*y*
    - (*p*(*x*))j = @*x.p*
    - (*x*~*y*)j = @*x.*~*y*

2

It follows from existing results that HTL is strictly more expressive than LTL↓. In [[3](#_bookmark18)] it was shown that every property of data words expressed by a LTL↓-formula can be decided by a 2-way alternating register automaton. In [[12](#_bookmark29)], for every *m* ≥ 1, a set *L*= of data words was defined such that

*m*

1. *L*= is definable in first-order logic for every *m*, but

*m*

1. for every *m* ≥ 4, there is no 2-way alternating register automaton for *L*= .

*m*

Of course, (b) and [[3](#_bookmark18)] yield [5](#_bookmark6) that *L*= cannot be defined by a LTL↓-formula, for

*m*

any *m* ≥ 4. By Theorem [3.1](#_bookmark4) and (a), every *L*= can be expressed in HTL and thus

*m*

HTL is strictly more expressive than LTL↓. In Theorem [3.3](#_bookmark5) we show that, for every *m*, there is a set *Lm*, similarly defined as *L*= , such that *Lm* can be defined by a HTL2(X*,* U)-formula, but there is no 2-way alternating register automaton for *Lm* if *m* ≥ 4. Thus, HTL2(X*,* U) /≤ LTL↓.

*m*

A *1-hyperset* over Ð is a finite subset of Ð. For *m >* 1, an *m-hyperset* over

Ð is a finite set of (*m* — 1)-hypersets over Ð. We assume for simplicity that Ð

is the set of natural numbers. We associate *m*-hypersets *Hm*(*w*) with data words

*w* over {*z, b ,e ,... ,b ,e* } as follows. A data word *w* = *b*1 *z*

··· *z e*1 , where *d*

1 1 *m m*

*d n*1

*nj d*′

and *d*j are arbitrary data values, represents the 1-hyperset *H*1(*w*) = {*n*1*,... , nj*}. If for some *m* ≥ 2 and *l* ≥ 0 every data word *uj*, 1 ≤ *j* ≤ *l*, represents an

(*m* — 1)-hyperset, then the data word *u* = *bm u*1 *Hm*(*u*)= {*Hm*—1(*u*1)*,... , Hm*—1(*ul*)}, e.g.,

*d*

*d*′

··· *ul em*

represents the *m*-hyperset

*H* *b*2 *b*1 *z z e*1 *b*1 *z z z e*1 *b*1 *z z e*1 *e*2 = {{1*,* 2}*,* {7*,* 8*,* 9}*,* {2*,* 5}}*.*

2

2

1 1 2 2

3 7 9 8 2

1 2 5 9

3

If *u* does not represent any *m*-hyperset, we write *Hm*(*u*)= ⊥. For every *m* ≥ 1 we define the set *Lm* (with the additional proposition *s*) as

*Lm* = {*us v* | *Hm*(*u*)= *Hm*(*v*) /= ⊥*,d* ∈ Ð}*.*

*d*

Proposition 3.2 ([[12](#_bookmark29)]) *For m* ≥ 4*, Lm cannot be decided by a 2-way alternating register automaton.*

This proposition can be shown along the same lines as the above mentioned results. Indeed, the result of [[12](#_bookmark29)] easily carries over to *Lm* and to the model of 2-way alternating register automata as defined in [[3](#_bookmark18)].

Theorem 3.3 *For each m* ≥ 1 *the set Lm is deﬁnable in* HTL2*.*

Proof. We define, for every *m* ≥ 1, a formula *ϕm* ∈ HTL2(X*,* U) such that *w* |= *ϕm* if and only if *w* ∈ *Lm* for all data words *w*. The formula *ϕm* is a conjunction of several subformulas. The following three subformulas together express that *w* is of the form *us v* with *Hm*(*u*) /= ⊥ and *Hm*(*v*) /= ⊥.

*d*

* + - * A straightforward formula *χone* expresses that “every position carries exactly one proposition from {*z, s, b*1*,... , bm, e*1*,... , em*}.”
      * “*w* is of the form *us v*, *u* and *v* start with a *bm*-position and end with a *em*-position and there are no other positions carrying *bm*, *em* or *s*.”

*d*

*χmain* = *bm* ΛX[ч(*bm* V*s*V*em*)U(*em* ΛX(*s*ΛX(*bm* Λ(ч(*bm* V*s*V*em*)U(*em* ΛчXT)))))]

5 Actually, the definitions of 2-way alternating register automata in [[3](#_bookmark18)] and [[12](#_bookmark29)] do not coincide completely. However, it is not hard to see that the result of [[3](#_bookmark18)] also holds for the automata of [[12](#_bookmark29)] over data words without propositions.

* + - “Every *bi*-position is directly followed by a *bi*—1- or an *ei*-position, every *z*-position is directly followed by a *z*- or an *e*1-position and, for *i < m*, every *ei*-position is directly followed by a *bi*- or an *ei*+1-position.” Here, *b*0 denotes *z*.

*m*

*χhyp* = G(*z* → X(*z* V *e*1)) Λ (*bi* → X(*bi*—1 V *ei*)) Λ

*i*=1

*m*—1

(*ei* → X(*bi* V *ei*+1))*.*

*i*=1

Next, we construct a formula *ψm* that actually expressess *Hm*(*u*)= *Hm*(*v*).

* + - The formula *ψ*1 checks that, if *x* and *y* are bound to *b*1-positions, the two 1- hypersets whose encodings start at *x* and *y*, respectively, are equal.

*ψ*1 =@*x.*X ч*e*1 Λ ↓*x.*@*y.*X(ч*e*1U(ч*e*1Λ ~ *x*)) U*e*1 Λ

@*y.*X ч*e*1 Λ ↓*y.*@*x.*X(ч*e*1U(ч*e*1Λ ~ *y*)) U*e*1

* + - Likewise, for 2 ≤ *i* ≤ *m* the formula *ψi* expresses that, if *x* and *y* are bound to

*bi*-positions, the *i*-hypersets starting at *x* and *y*, respectively, are equal:

*ψi* = @*x.* *bi*—1 → ↓*x.*@*y.*(ч*ei*U(*bi*—1 Λ ↓*y.ψi*—1)) U*ei* Λ

@*y.* *bi*—1 → ↓*y.*@*x.*(ч*ei*U(*bi*—1 Λ ↓*x.ψi*—1)) U*ei*

Finally, the desired formula is *ϕm* = *χone* Λ *χmain* Λ *χhyp* Λ ↓*x.*F(*s* Λ X↓*y.ψm*)*.*

Every word *w* = *us v* ∈ *Lm* satisfies *χone*, *χmain* and *χhyp*, by construction. As

*d*

*u* and *v* represent the same hypersets, both parts of *ψm* are satisfied, too.

If a data word *w* satisfies *ϕm*, the formulas *χone*, *χmain* and *χhyp* ensure that *w* is of the form *us v* and that *u* and *v* encode *m*-hypersets. The two parts of *ψm* make sure that every (*m* — 1)-hyperset encoded in *u* also occurs in *v* and vice versa. Thus, the completeness and correctness of *ϕm* follow. 2

*d*

Corollary 3.4 HTL2 /≤ LTL↓*.*

A closer inspection of the proof of Theorem [3.3](#_bookmark5) gives further insights. First, the formulas *ϕm* do not use any past operators and do not use atomic formulas of the form *x*. On the other hand, as already observed in [[15](#_bookmark32)], formulas of the form @*x.χ* can be replaced by formulas of the form FF—(*x* Λ *χ*) and thus, in the presence of

past operators and atomic formulas *x*, there is no need for an @-operator. Thus,

we get the following corollary of the proof of Theorem [3.3](#_bookmark5).

Corollary 3.5 LTL↓ *cannot express all properties expressible in*

1. HTL2(X*,* U) *without atomic formulas of the form x, and in*
2. HTL2 *without subformulas of the form* @*x.χ.*

However, without past operators, the @-operator cannot be simulated any more. Indeed, the future fragment of HTL without the @-operator is as expressive as the future fragment of LTL↓ with respect to closed formulas.

Proposition 3.6 *Every closed* HTL(X*,* U) *formula without the* @*-operator can be translated into an equivalent* LTL↓(X*,* U)*-formula.*

Proof. For *k* ≥ 1 let *ϕ* be a closed HTL*k*(X*,* U) formula without any occurrences of @. Without loss of generality we assume that at most the variables *x*1*,... , xk* occur in *ϕ*. The idea is that in a formula ↓*x.ψ*, atomic formulas *x* evaluate to true only until some temporal operator has “moved” the “current position”. Thus, it suffices is to keep track of whether a subformula of *ϕ* is evaluated on a position bound to a variable or not. Depending on this, atomic formulas *xi* can be replaced by T or ⊥. As an example, the formula ↓*xi.*X*xi* is equivalent to the formula ↓*xi.*X⊥.

We define a mapping *onS* that translates, for every subset *S* of {*x*1*,... , xk*}, HTL*k*(X*,* U)-formulas *ψ* into LTL↓(X*,* U)-formulas *onS*(*ψ*) such that for every data word *w*, every assignment *g* and every position *i* of *w* it holds

*k*

*w, g, i* |= *ψ* ⇔ *w, g, i* |= *onS*(*ψ*)*,*

if *S* is chosen as {*xj* | *g*(*xj*)= *i*}.

The transformation is defined as follows (we omit the Boolean operators).

* *onS*(*p*)= *p* for propositions *p*

T*,* if *xi* ∈ *S*

* *onS*(*xi*)=

⊥*,* otherwise

* *onS*(~*xi*)= ~*xi*
* *onS*(↓*xi.χ*)= ↓*xi.onS*∪{*xi*}(*χ*)
* *onS*(X*χ*)= X*on*$(*χ*)
* *onS*(*χ*U*χ*j)= *onS*(*χ*j) V *onS*(*χ*) Λ X(*on*$(*χ*)U*on*$(*χ*j))

Finally, formula *ϕ* is equivalent to the LTL↓(X*,* U) formula *on*$(*ϕ*). 2

*k*

We finally note that for HTL on data words similar observations can be made as for HTL on linear frames in [[9](#_bookmark26)]. In particular, for *k* ≥ 1, every HTL*k* formula can be converted into an initially equivalent HTL*k*+2 future formula. The idea is to bind the first additional variable, say *x*, to the first position of the word. A similar technique was used in [[19](#_bookmark36)] in the context of branching time logics.

Lemma 3.7 *For k* ≥ 1*, every* HTL*k formula can be converted into an initially equivalent* HTL*k*+2(X*,* U) *formula.*

Proof. Let *x* and *y* be two additional variables. We use the following translation of HTL*k* formulas *ϕ* into HTL*k*+2 formulas *ϕ*j.

* (X*ψ*)j = X*ψ*j
* (*ψ*U*χ*)j = *ψ*jU*χ*j
* (X—*ψ*)j = ↓*y.*@*x.*F(X*y* Λ *ψ*j)
* (*ψ*U—*χ*)j: ↓*y.*@*x.*F(F*y* Λ *χ*j Λ (*y* V XG(F*y* → *ψ*j)))

The trivial cases are omitted, here. Then, every HTL*k* formula *ϕ* is initially equivalent to the HTL*k*+2(X*,* U) formula ↓*x.ϕ*j. 2

* 1. *Hierarchy results*

In this section, we show that both the HTL- and the LTL↓-hierarchy have infinitely many levels. More precisely, there is no *k >* 0 such that for every *i*, HTL*i* ⊆ HTL*k*

or LTL↓ ⊆ LTL↓. It turns out that this can be concluded from Rossman’s celebrated

*i k*

theorem that the variable hierarchy for first-order logic on ordered graphs is strict

[[14](#_bookmark31)]. In the following, we denote the restriction of first-order logic to formulas with at most *k* variables by FO*k*.

Theorem 3.8 (Rossman [[14](#_bookmark31)]) *For every k* ≥ 1*,* FO*k* Ç FO*k*+1 *on ﬁnite undi- rected, ordered graphs.*

We define *canonical encodings* of finite undirected ordered graphs by data words and show the following two lemmas.

Lemma 3.9 *For every formula ϕ* ∈ FO*k there is a formula ϕ*j ∈ LTL↓

*k*+1

*such that*

*for every undirected ordered graph G and every canonical encoding w of G it holds*

*G* |= *ϕ* ⇔ *w* |= *ϕ*j*.*

Lemma 3.10 *For every formula ϕ*j ∈ HTL*k there is a formula ϕ* ∈ FO2*k*+6 *such that for every undirected ordered graph G and every canonical encoding w of G it holds w* |= *ϕ*j ⇔ *G* |= *ϕ.*

These two lemmas and Rossman’s result yield the following theorem.

Theorem 3.11 *(a) The* LTL↓*-hierarchy on data words is inﬁnite.*

*k*

*(b) The* HTL*k-hierarchy on data words is inﬁnite.*

Proof. For the sake of a contradiction, let us assume that there is some *i* such that

for every *j > i*, LTL↓ = LTL↓ or such that for every *j > i*, HTL*j* = HTL*i*. Let *ϕ*

*j* *i*

be an arbitrary formula on ordered graphs from FO2*i*+7. By Lemma [3.9](#_bookmark8), there is a

formula *ϕ*j ∈ LTL↓

2*i*+8

such that *G* |= *ϕ* ⇔ *w* |= *ϕ*j. By assumption, there is [6](#_bookmark11) a

formula *ψ*j ∈ HTL*i* such that for every data word *w* it holds *w* |= *ψ*j ⇔ *w* |= *ϕ*j. By Lemma [3.10](#_bookmark9), there is a formula *ψ* ∈ FO2*i*+6 such that for every undirected ordered graph *G* and every canonical encoding *w* of *G* it holds *w* |= *ψ*j ⇔ *G* |= *ψ*.

Thus, *ϕ* ≡ *ψ*. As *ϕ* was arbitrarily chosen from FO2*i*+7 we can conclude that

FO2*i*+7 = FO2*i*+6, the desired contradiction. 2

It only remains to define canonical encodings and to prove the two lemmas.

In the following, we only consider graphs without self-loops. This is consistent with Rossman’s Theorem. We say that a data word *w* is a *canonical encoding* of a finite ordered undirected graph *G* = (*V, E, <*) if the following conditions hold.

* *w* has |*V* | + 2|*E*| positions,
  + a *node position p*(*u*), for every *u* ∈ *V* and
  + an *edge position q*(*u, v*), for every [7](#_bookmark12) edge (*u, v*).

*i*

6 If we assume that the LTL↓-hierarchy collapses then we get a formula *ψ*′ ∈ LTL↓ at this point which is,

however, also in HTL*i*.

7 Every edge (*u, v*) gives rise to two positions, *q*(*u, v*) and *q*(*v, u*).

* Node positions *p*(*u*) have a unique data value and carry the proposition *p*.
* For every *u, v*, (*u, v*) ∈ *E*, the edge positions *q*(*u, v*) and *q*(*v, u*) have the same data value (and this value does not occur otherwise) and do not carry any propositions.
* The order of the positions obeys the following rules, for every *u, u*j*, v, v*j ∈ *V* :
  + *p*(*u*) *< p*(*v*) if *u < v* in *G*.
  + *p*(*u*) *< q*(*u*j*, v*j) if *u* ≤ *u*j.
  + *q*(*u, v*) *< p*(*u*j) if *u < u*j.
  + *q*(*u, v*) *< q*(*u, v*j) if *v < v*j.

It should be noted that these rules define a unique order on every canonical data string *w*.

Example 3.12 If in the undirected finite graph *G* = the nodes are

*a b*

*c d*

ordered by *a < b < c < d*, then

*p* ∅ ∅ *p* ∅ ∅ *p* ∅ ∅ ∅ *p* ∅

7 8 3 2 8 1 6 3 1 5 4 5

is a canonical encoding of *G*. It

should be observed that the underlying linear order of data values (1 *<* 2 *<* 3 *<* ··· )

is not relevant for the encoding.

A maximal subword in a canonical encoding where only the first position is labeled by *p* is called a *variable block*. As an example, the subword from the 4th to the 6th position of the above encoding constitutes a variable block.

Proof of Lemma [3.9](#_bookmark8). We assume without loss of generality that formulas from FO*k* only use variables *x*1*,... , xk*. The translation *ϕ* '→ *ϕ*j can be defined inductively as follows.

* (*xi* = *xj*)j = FF—(~*xi* Λ ~*xj*)
* (*xi < xj*)j = FF—(~*xi* Λ XF~*xj*)

* *E*(*xi, xj*)j = FF—(~*xi* Λ ч*p*U↓*y.*FF—(~*y* Λ ч*p*U—(~*xj* Λ /~*xi*))
* (∃*xiψ*)j = FF—↓*xi.*(*p* Λ *ψ*j)
* (ч*ψ*)j = ч*ψ*j
* (*ψ* Λ *χ*)j = *ψ*j Λ *χ*j

In *ϕ*j, variables *xi* are always only bound to first positions of variable blocks. Whether two nodes bound to *xi* and *xj* are connected by an edge can then be tested by checking whether the blocks starting at *xi* and *xj*, respectively, share some data value. 2

Proof of Lemma [3.10](#_bookmark9). The proof consists of two steps. First, we construct, as in the proof of Theorem [3.1](#_bookmark4), an FO*k*+3-formula *ϕ*jj that is equivalent to *ϕ*j on data words.

Then we transform *ϕ*jj into *ϕ* by means of a quantifier-free logical interpretation [[6](#_bookmark23), Section 11.2] that defines, for every finite ordered undirected graph *G* = (*V, E, <*

), a (unique) representation of the canonical encodings of *G* on the set *V* × *V* . However, the translation of *ϕ*jj into *ϕ* requires two variables, *xi, x*j , for every variable

*i*

*xi* of *ϕ*jj, thus resulting in a formula with 2*k* +6 variables. More precisely, the logical interpretation Φ = (*ϕU , ϕp, ϕ<, ϕ*∼) is defined as follows.

* *ϕU* (*x*1*, x*2) defines the set of pairs that are actual positions of the representation of the canonical encodings. Thus, it is just *x*1 = *x*2 V *E*(*x*1*, x*2).
* *ϕp*(*x*1*, x*2) defines the positions that carry the proposition *p* and is thus just

*x*1 = *x*2.

* *ϕ<*(*x*1*, x*2*, y*1*, y*2) defines the linear order on positions. It is (*x*1 = *x*2 Λ (*x*1 *< y*1 V (*x*1 = *y*1 Λ *x*2 /= *y*2)))V

(*x*1 /= *x*2 Λ (*x*1 *< y*1 V (*x*1 = *y*1 Λ *x*2 *< y*2)))*.*

* Finally, *ϕ*∼(*x*1*, x*2*, y*1*, y*2) defines the pairs of positions that have the same data value. It is simply *x*1 = *y*2 Λ *x*2 = *y*1.

It is not hard to see that Φ indeed defines, for every *G*, the unique representation of the canonical encodings of *G*. 2

Remark 3.13 We conjecture that both hierarchies are even strict and that this can be shown in a similar way as the strictness of the FO*k*-hierarchy can be concluded from its infinity [[14](#_bookmark31)]. However, we did not have time to figure out the details before the deadline for this submission.

* 1. *Succinctness*

We have seen before that HTL2 can express properties that cannot be expressed by any LTL↓-formula. We show next that there are also LTL↓-expressible properties that can be expressed non-elementarily more succinct in HTL2. This is basically a

corollary from results in [[17](#_bookmark34)], [[16](#_bookmark33)] and [[15](#_bookmark32)] and the observation that on data words in which all positions carry the same data value, LTL↓ is expressively equivalent to PLTL.

In [[17](#_bookmark34)] it is shown that there are star-free regular expressions (*αn*)*n*≥1, built from union, concatenation, and negation, such that, there is no elementary function *f* for which *f* (*n*) bounds the length of the size of the smallest string satisfying *αn*, for every *n* ≥ 1. In [[8](#_bookmark25)] it is explained how for every star-free regular expression *α*

one can build an equivalent FO formula. Following a similar technique, [[15](#_bookmark32)] gives a translation from star-free regular expressions to hybrid logic formulas. In our setting, the translation yields HTL-formulas of linear size in |*αn*|. On the other hand, [[16](#_bookmark33)] proves that every satisfiable LTL formula *ψ* can be satisfied by a string of length at most exponential in |*ψ*|. Combining these results we get:

Proposition 3.14 HTL2(X*,* F) *is non-elementarily more succinct than* LTL↓*.*

# One Variable

In this section, we show our main result, that HTL1 ≡ LTL↓ on data words. Further, we prove that HTL1(F) is exponentially more succinct than LTL↓.

1

* 1. *Expressiveness*

The translation of HTL1 to LTL↓

1

relies on a kind of separation property. We

show that if *ϕ* = ↓*x.χ* then the “top level” of *χ* can be rewritten into a Boolean

combination of future and past formulas.

We introduce some new notation for the proof.

For a data word *w*,a position *i* of *w* and a set Φ of HTL1-formulas we denote by (*w, i*)Φ the string that is obtained from *w* by removing the data values and adding to each position *j* all propositions *pψ*, *ψ* ∈ Φ, for which *w, i, j* |= *ψ*.

A subformula *ψ* of a HTL1-formula *ϕ* is called a *top-level subformula* of *ϕ* if *ψ*

is not in the scope of any ↓*x* quantifier.

—→

For every HTL1-formula *ϕ* we let *T* (*ϕ*) be the set of all top-level subformulas of

*ϕ* that are of one of the forms ↓*x.χ*, ~*x* or @*x.χ* and of all formulas @*x.*X—*χ* and @*x.χ*U—*θ* for which X—*χ* or *χ*U—*θ*, respectively, is a top-level subformula of *ϕ*. We

—→

define *Tx*(*ϕ*) analogously, but with the additional atomic formula *x*.

If *j* is a position of a data word *w* we write *w*[*j,.. .*) for the subword of *w* starting at position *j*.

Lemma 4.1 *For every* HTL1*-formula ϕ there is an* LTL*-formula* —→*ϕ such that for every data word w and all positions i, l of w it holds*

*w, l, i* |= ↓*x.ϕ* ⇔

—→

(*w, i*) *T* (*ϕ*)

*,i* |= —→*ϕ.*

*The size of* —→*ϕ is at most triply exponential in* |*ϕ*|*.*

Proof. We inductively define, for every top-level subformula *ψ* of *ϕ*, a PLTL- formula *ψ* such that, for all positions *i, j* with *j* ≥ *i*, it holds

˜

—→

*w, i, j* |= *ψ* ⇔ (*w, i*)*Tx* (*ϕ*)*,j* |= *ψ*˜*.* (1)

To this end, let *ψ* be

˜

* *p* if *ψ* is a proposition *p*;
* *pψ* if *ψ* is of one of the forms *x*, ~*x*, ↓[*x.χ*, @*x.χ*;](mailto:x.χ%2C@x.χ) if *ψ* = X*χ*;
* X*χ*˜
* *χ*˜U*θ*˜, if *ψ* = *χ*U*θ*;
* (ч*px* Λ X—*χ*) V (*px*

˜

Λ [*p*@*x.*X](mailto:p@x.X)−*χ*

) if *ψ* = X—*χ*;

* (ч*px* Λ *χ*)U—((ч*px* Λ *θ*) V (*px* Λ [*p*@*x.χ*U](mailto:p@x.χU)−*θ* )) if *ψ* = *χ*U—*θ*.

˜ ˜

That is, the usual evaluation of past operators is restricted to positions ≥ *i*. If this is insufficient then the new propositions are used. It is straightforward to show by induction that Equation [1](#_bookmark15) indeed holds.

Let now *ϕ* be the formula that results from *ϕ* by replacing every occurrence of

1 ˜ —→

*p*

with чX—T. Clearly, for all data words *w* and positions *i* ≤ *j*, (*w, i*)*Tx* (*ϕ*)*,j* |= *ϕ*

*x* —→ ˜

if and only if (*w, i*) *T* (*ϕ*)[*i,.. .*)*,* (*j* — *i* + 1) |= *ϕ*1.

By [[10](#_bookmark27), Theorem 2.4] the PLTL-formula *ϕ*1 can be effectively translated into an LTL-formula —→*ϕ* that is initially equivalent to *ϕ*1. Moreover, by [[11](#_bookmark28)] it follows that

—→*ϕ* can be computed in triply exponential time. As the positions *< i* are irrelevant

for the validity of —→*ϕ* at position *i*, altogether the lemma follows. 2

→—

Similarly, we let, for every HTL1-formula *ϕ*, *T* (*ϕ*) be the set of all top-level

subformulas of *ϕ* that are of one of the forms ↓*x.χ*, [~*x*, @*x.χ*](mailto:~x%2C@x.χ)and of all formulas @*x.*X*χ* and @*x.χ*U*θ* for which X*χ* or *χ*U*θ*, respectively, is a top-level subformula of *ϕ*. Analogously we get the following lemma.

Lemma 4.2 *For every* HTL1*-formula ϕ there is a* PLTL*-formula* →*ϕ*— *which does not use any future operator such that for every data word w and all positions i, l of*

*w it holds*

→— →—

*w, l, i* |= ↓*x.ϕ* ⇔ (*w, i*) *T* (*ϕ*)*,i* |= *ϕ.*

*The size of* →*ϕ*— *is at most triply exponential in* |*ϕ*|*.*

Theorem 4.3 *Every closed* HTL1 *formula can be translated into an equivalent*

LTL↓ *formula of at most triply exponential size.*

1

Proof. It suffices to prove that for every HTL1 formula *ϕ* there is a LTL↓ formula *ϕ*j of triply exponential size with ↓*x.ϕ* ≡ ↓*x.ϕ*j. We show this by induction on the size of *ϕ*. Thanks to the equivalences

1

* + - ↓*x.p* ≡ *p*,
    - ↓*x.x* ≡ T,
    - ↓*x.*~*x* ≡ T,
    - ↓[*x.*@*x.ψ*](mailto:x.@x.ψ)≡ ↓*x.ψ*,
    - ↓*x.*ч*ψ* ≡ ч↓*x.ψ*,
    - ↓*x.*(*ψ* Λ *χ*) ≡ ↓*x.ψ* Λ ↓*x.χ*

we can assume that *ϕ* is of one of the forms (1) *ψ*U*χ*, (2) X*ψ*, (3) *ψ*U—*χ*, (4) X—*ψ*.

We first consider the cases (1) and (2). Let —→*ϕ* be as guaranteed by Lemma [4.1](#_bookmark14).

Thus, for every data word *w* and all positions *i, l* of *w*, it holds

*w, l, i* |= ↓*x.ϕ* ⇔

—→

(*w, i*) *T* (*ϕ*)

*,i* |= —→*ϕ.*

Clearly, replacing every atomic formula *p*~*x* by ~*x* in —→*ϕ* and evaluating the formula

—→

in the data word resulting from (*w, i*) *T* (*ϕ*)[*i,.. .*) by putting back the data values

from *w* does not change the validity of —→*ϕ* . Likewise, replacing every atomic formula *p*↓*x.ψ* by the formula ↓*x.ψ*j, where *ψ*j is a LTL↓-formula equivalent to *ψ* obtained by induction, does not change the validity either. We denote the resulting formula by *ϕ*.

1

^

It only remains to eliminate atomic formulas of the kind *p* from *ϕ*. For every

@*x.ψ* ^

—→

assignment *α* : *T* (*ϕ*) → {T*,* ⊥} let *ϕα* be the formula resulting from *ϕ* by replacing

every occurrence of an atomic formula [*p*@*x.ψ*](mailto:p@x.ψ)with [*α*(@*x.ψ*).](mailto:α(@x.ψ) We finally let

^

*ϕ*j =

—→

*α*: *T* (*ϕ*)→{T*,*⊥}

*ϕα* Λ

—→

@*x.ψ*∈ *T* (*ϕ*)

*α*(@*x.ψ*)=T

*ψ*j Λ

—→

@*x.ψ*∈ *T* (*ϕ*)

[*α*(@*x.ψ*)=](mailto:α(@x.ψ)⊥

ч*ψ*j *,*

where *ψ*j is again a LTL↓-formula equivalent to *ψ* obtained by induction that does not use any additional propositions anymore.

1

The cases (3) and (4) are completely analogous.

The size of the resulting formula is dominated by the (at most) triply exponential size of —→*ϕ* . The elimination of @ only contributes an exponential factor and we end up with a formula of at most triply exponential size. 2

* 1. *Succinctness*

Even though HTL1 ≡ LTL↓, HTL1 can express some properties exponentially more succinct than LTL↓ and, actually, even LTL↓.

1

1

Proposition 4.4 HTL1(F) *is exponentially more succinct than* LTL↓*.*

Proof. The proof essentially follows the proof of [[8](#_bookmark25), Theorem 3 (1)] that FO2 is exponentially more succinct than *unary* LTL. Let *En* be the property *“Any two positions of the word that agree on propositions p*1*, p*2*,... , pn also agree on proposition p*0*.”* and let *Ln* be the set of data words fulfilling *En* in which all positions have the same data value. For every *n* ≥ 1, *Ln* is expressed by the

following HTL1(F) formula of length O(*n*):

G↓*x.*G ~ *x* Λ G[↓*x.*G( *n* (*pi* ↔ @*x.pi*) → (*p*0 ↔ @*x.p*0))]*.*

*i*=1

Let us assume now that, for every *n*, there is a LTL↓-formula *ψn* expressing *Ln* and |*ψn*| = 2*o*(*n*). This formula can be translated into a formula *χn* of roughly the same size that expresses *En* on words without data. Similarly as in [[18](#_bookmark35)], there is,

*o*(*n*)

for every *n*, a non-deterministic automaton for *χn* of size 2|*χn*| = 22 . However,

it can be shown as in [[8](#_bookmark25)] that every automaton for *En*

requires at least 22*n* states,

the desired contradiction. 2

# Discussion

In this paper, we compared the expressive power of hybrid temporal logic on data words with LTL↓. Although HTL is more powerful in general, the two logics coincide if only one variable is allowed.

The main results of this paper carry over to data words of infinite length. For Corollary [3.4](#_bookmark7) and Theorem [3.11](#_bookmark10) this simply holds because the separating languages can be easily turned into *ω*-languages by padding with an infinite number of posi- tions. The generalization of Theorem [4.3](#_bookmark16) to infinite data words is also straightfor- ward. Here, it is important that the result of [[11](#_bookmark28)] was already shown for *ω*-strings. Clearly, all considered logics have an undecidable satisfiability problem (for LTL↓

1

this was shown in [[3](#_bookmark18)]). However, the model checking remains largely unexplored,

especially for the case of infinite data words.

As already mentioned in Section [3](#_bookmark3), we conjecture that the variable-hierarchy for HTL and LTL↓ are both strict.

# References

1. Carlos Areces, Patrick Blackburn, and Maarten Marx. The computational complexity of hybrid temporal logics. *Logic Journal of the IGPL*, 8(5):653–679, 2000.
2. Mikolaj Bojanczyk, Anca Muscholl, Thomas Schwentick, Luc Segoufin, and Claire David. Two-variable logic on words with data. In *LICS*, pages 7–16. IEEE Computer Society, 2006.
3. Stephane Demri and Ranko Lazic. LTL with the freeze quantifier and register automata. In *LICS ’06: Proceedings of the 21st Annual IEEE Symposium on Logic in Computer Science*, pages 17–26, Washington, DC, USA, 2006. IEEE Computer Society.
4. St´ephane Demri and Ranko Lazic. LTL with the freeze quantifier and register automata. *ACM Trans.* *Comput. Log.*, 10(3), 2009. Full version of [[3](#_bookmark18)].
5. St´ephane Demri, Ranko Lazi´c, and David Nowak. On the freeze quantifier in constraint LTL: Decidability and complexity. *Inf. Comput.*, 205(1):2–24, 2007.
6. Heinz-Dieter Ebbinghaus and Jo¨rg Flum. *Finite Model Theory*. Springer, Heidelberg, 2005.
7. E. Allen Emerson. Temporal and modal logic. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 995–1072. 1990.
8. Kousha Etessami, Moshe Y. Vardi, and Thomas Wilke. First-order logic with two variables and unary temporal logic. *Information and Computation*, 179(2):279 – 295, 2002.
9. Massimo Franceschet, Maarten de Rijke, and Bernd-Holger Schlingloff. Hybrid logics on linear structures: Expressivity and complexity. In *TIME*, pages 166–173, 2003.
10. Dov M. Gabbay. The declarative past and imperative future: Executable temporal logic for interactive systems. In *Temporal Logic in Specification*, pages 409–448, 1987.
11. Nicolas Markey. Temporal logic with past is exponentially more succinct, concurrency column. *Bulletin of the EATCS*, 79:122–128, 2003.
12. Frank Neven, Thomas Schwentick, and Victor Vianu. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Log.*, 5(3):403–435, 2004.
13. Arthur Prior. *Past, Present, and Future*. Oxford University Press, 1967.
14. Benjamin Rossman. On the constant-depth complexity of k-clique. In *STOC ’08: Proceedings of the 40th annual ACM symposium on Theory of computing*, pages 721–730, New York, NY, USA, 2008. ACM.
15. Thomas Schwentick and Volker Weber. Bounded-variable fragments of hybrid logics. In *STACS*, pages 561–572, 2007.
16. A. Prasad Sistla and Edmund M. Clarke. The complexity of propositional linear temporal logics. *J. ACM*, 32(3):733–749, 1985.
17. L. Stockmeyer. The complexity of decision problems in automata and logic, 1974. Ph.D. Thesis, MIT, 1974.
18. Moshe Y. Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *LICS*, pages 332–344. IEEE Computer Society, 1986.
19. Volker Weber. Branching-time logics repeatedly referring to states. *Journal of Logic, Language and Information*, 18(4):593–624, 2009.