

Electronic Notes in Theoretical Computer Science 239 (2009) 105–117

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

On Decidability of LTL+Past Model Checking for Process Rewrite Systems

Mojm´ır Kˇret´ınsky´[1](#_bookmark0) *,*[4](#_bookmark1) Vojtˇech Rˇeha´k[2](#_bookmark1) *,*[5](#_bookmark1) Jan Strejˇcek[3](#_bookmark1) *,*[6](#_bookmark1)

*Faculty of Informatics, Masaryk University Botanicka´ 68a, 60200 Brno, Czech Republic*

Abstract

The paper [[4](#_bookmark14)] shows that the model checking problem for (weakly extended) Process Rewrite Systems and properties given by LTL formulae with temporal operators *strict eventually* and *strict always* is decidable. The same paper contains an open question whether the problem remains decidable even if we extend the set of properties by allowing also past counterparts of the mentioned operators. The current paper gives a positive answer to this question.

*Keywords:* rewrite systems, infinite-state systems, model checking, decidability, linear temporal logic

# Introduction

To specify (the classes of) infinite-state systems we employ term rewrite systems called *Process Rewrite Systems* (PRS) [[16](#_bookmark25)]. PRS subsume a variety of the formalisms studied in the context of formal verification, e.g. *Petri nets* (PN), *pushdown pro- cesses* (PDA), and process algebras like PA. Moreover, they are suitable to model current software systems with restricted forms of dynamic creation and synchroniza- tion of concurrent processes or recursive procedures or both. The relevance of PRS (and their subclasses) for modelling and analysing programs is shown, for example, in [[7](#_bookmark17)]; for automatic verification we refer to surveys [[5](#_bookmark15),[19](#_bookmark26)].

1 Email: [kretinsky@fi.muni.cz](mailto:kretinsky@fi.muni.cz)

2 Email: [rehak@fi.muni.cz](mailto:rehak@fi.muni.cz)

3 Email: [strejcek@fi.muni.cz](mailto:strejcek@fi.muni.cz)

4 Supported by Ministry of Education of the Czech Republic, project No. MSM0021622419, and by the Czech Science Foundation, grant No. 201/09/1389.

5 Supported by the research centre “Institute for Theoretical Computer Science (ITI)”, project No. 1M0545, and by the Czech Science Foundation, grant No. 201/08/P459.

6 Supported by the Academy of Sciences of the Czech Republic, grant No. 1ET408050503, and by the Czech Science Foundation, grant No. 201/08/P375.

1571-0661/© 2009 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.2009.05.033

Another merit of PRS is that the *reachability problem* is decidable for PRS [[16](#_bookmark25)]. In [[13](#_bookmark23)], we have presented *weakly extended PRS* (wPRS), where a finite-state control unit with self-loops as the only loops is added to the standard PRS formalism (addi- tion of a general finite-state control unit makes PRS language equivalent to Turing machines). This *weak* control unit enriches PRS by abilities to model a bounded number of arbitrary communication events and global variables whose values are changed only a bounded number of times during any computation. We have shown that the reachability problem remains decidable for wPRS [[12](#_bookmark22)].

One of the mainstreams in an automatic verification of programs is model check- ing. Here we focus on *Linear Temporal Logic* (LTL). Recall that LTL model checking is decidable for both PDA (EXPTIME-complete [[1](#_bookmark11)]) and PN (at least as hard as the reachability problem for PN [[6](#_bookmark16)]). Conversely, LTL model checking is undecidable for all the classes subsuming PA [[2](#_bookmark12),[15](#_bookmark27)]. So far, there are few positive results for these classes. Model checking of infinite runs is decidable for the PA class and the fragment *simple PLTL*, see [[2](#_bookmark12)], and also for the PRS class and a fragment of LTL expressing exactly fairness properties [[3](#_bookmark13)]. Recently, the model checking problem has been shown decidable for (w)PRS and properties given by an LTL fragment LTL(Fs*,* Gs), i.e. that with operators *strict eventually* and *strict always* only, see [[4](#_bookmark14)]. Our contribution: As a main result we extend a proof technique used in [[4](#_bookmark14)] with past modalities and show that the model checking problem stays decidable even for wPRS and LTL(Fs*,* Ps), i.e. an LTL fragment with modalities *strict eventually* and *eventually in the strict past* (and where *strict always* and *always in the strict past* can be used as derived modalities). We note that a role of past operators in program verification is advocated e.g. in [[14](#_bookmark24),[9](#_bookmark19)]. Let us mention that the expressive power of the fragment LTL(Fs*,* Ps) semantically coincides with formulae of First- Order Monadic Logic of Order containing at most 2 variables and no successor predicate (FO2[*<*]), see [[8](#_bookmark18)] for effective translations. Thus we also positively solve

the model checking problem for the wPRS class and FO2[*<*].

# Preliminaries

* 1. *Weakly Extended PRS (wPRS)*

Let *Const* = {*X,.. .*} be a set of *process constants*. A set T of *process terms t* is defined by the abstract syntax *t* ::= *ε* | *X* | *t.t* | *t * *t*, where *ε* is the *empty term*, *X* ∈ *Const* , and ’*.*’ and ’  ’ mean *sequential* and *parallel compositions*, respectively. We always work with equivalence classes of terms modulo commutativity and asso- ciativity of ’  ’, associativity of ’*.*’, and neutrality of *ε*, i.e. *ε.t* = *t.ε* = *t * *ε* = *t*.

Let *M* = {*o, p, q,.. .*} be a set of *control states*, ≤ be a partial ordering on this set, and *Act* = {*a, b, c,.. .*} be a set of *actions*. An *wPRS* (*weakly extended process rewrite system*) Δ is a tuple (*R, p*0*, t*0), where

* + - *R* is a finite set of *rewrite rules* of the form (*p, t* ) *‹a* (*q, t* ), where *t ,t*

∈ T ,

1 → 2 1 2

*t*1 /= *ε*, *a* ∈ *Act* , and *p, q* ∈ *M* satisfy *p* ≤ *q*,

* + - the pair (*p*0*, t*0) ∈ *M* ×T forms the distinguished *initial state*.

By *Act* (Δ), *Const* (Δ), and *M* (Δ) we denote the respective sets of actions, process constants, and control states occurring in the rewrite rules or the initial state of Δ. A wPRS Δ = (*R, p*0*, t*0) induces a labelled transition system, whose states are pairs (*p, t*) such that *p* ∈ *M* (Δ) and *t* is a process term over *Const* (Δ). The transition relation −→ is the least relation satisfying the following inference rules:

*a*

((*p, t* ) *‹*

(*q, t* )) ∈ *R*

(*p, t* ) −*a*→ (*q, t* )

(*p, t* ) −*a*→ (*q, t* )

1 → 2

(*p, t*1) −*a*→ (*q, t*2)

1 2 1 2

(*p, t*1  *t*' ) −*a*→ (*q, t*2  *t*' ) (*p, t*1*.t*' ) −*a*→ (*q, t*2*.t*' )

1 1 1 1

To shorten our notation we write *pt* in lieu of (*p, t*). A state *pt* is called *terminal* if there is no state *p*'*t*' and no action *a* such that *pt* −*a*→ *p*'*t*'. Here, we always consider only such systems where the initial state is not terminal. A (finite or infinite) sequence

+1

*σ* = *p*0*t*0

−*a*→0

*p*1*t*1

−*a*→1

*...* −*a*→n

*pn*+1*tn*+1

*a*n

−→ *...*

is called a *run of* Δ *over the word u* = *a*0*a*1 *... an*(*an*+1 *.. .*) if it starts in the initial state and, provided it is finite, ends in a terminal state. Further, *L*(Δ) denotes the set of words *u* such that there is a run of Δ over *u*.

If *M* (Δ) is a singleton, then wPRS Δ is called a *process rewrite system* (*PRS* ) [[16](#_bookmark25)]. PRS, wPRS, and their respective subclasses are discussed in more detail in [[18](#_bookmark28)].

* 1. *Linear Temporal Logic (LTL) and the Studied Problems*

The syntax of *Linear Temporal Logic* (LTL) [[17](#_bookmark29)] is defined as follows

*ϕ* ::= *tt* | *a* | ¬*ϕ* | *ϕ* ∧ *ϕ* | X*ϕ* | *ϕ* U *ϕ* | Y*ϕ* | *ϕ* S *ϕ*,

where X and U are future modal operators *next* and *until*, while Y and S are their past counterparts *previously* and *since*, and *a* ranges over *Act* . The logic is inter- preted over infinite and nonempty finite pointed words of actions. Given a word *u* = *a*0*a*1*a*2 *...* ∈ *Act* ∗ ∪ *Act ω*, |*u*| denotes the length of the word (we set |*u*| = ∞ if *u* is infinite). A *pointed word* is a pair (*u, i*) of a nonempty word *u* and a *position* 0 ≤ *i <* |*u*| in this word.

The semantics of LTL formulae is defined inductively as follows: (*u, i*) |= *tt*

(*u, i*) |= *a* iff *u* = *a*0*a*1*a*2 *...* and *ai* = *a*

(*u, i*) |= ¬*ϕ* iff (*u, i*) |= *ϕ*

(*u, i*) |= *ϕ*1 ∧ *ϕ*2 iff (*u, i*) |= *ϕ*1 and (*u, i*) |= *ϕ*2

(*u, i*) |= X*ϕ* iff *i* +1 *<* |*u*| and (*u, i* + 1) |= *ϕ*

(*u, i*) |= *ϕ*1 U *ϕ*2 iff ∃*k.* *i* ≤ *k <* |*u*| ∧ (*u, k*) |= *ϕ*2 ∧

∧ ∀*j.* (*i* ≤ *j < k* ⇒ (*u, j*) |= *ϕ*1)

(*u, i*) |= Y*ϕ* iff 0 *< i* and (*u, i* − 1) |= *ϕ*

(*u, i*) |= *ϕ*1 S *ϕ*2 iff ∃*k.* 0 ≤ *k* ≤ *i* ∧ (*u, k*) |= *ϕ*2 ∧

∧ ∀*j.* (*k < j* ≤ *i* ⇒ (*u, j*) |= *ϕ*1)

We say that (*u, i*) *satisﬁes ϕ* whenever (*u, i*) |= *ϕ*. Further, a nonempty word *u satisﬁes ϕ*, written *u* |= *ϕ*, whenever (*u,* 0) |= *ϕ*. Given a set *L* of words, we write *L* |= *ϕ* if *u* |= *ϕ* holds for all *u* ∈ *L*. Finally, we say that a run *σ* of a wPRS Δ over a word *u* satisfies *ϕ*, written *σ* |= *ϕ*, whenever *u* |= *ϕ*.

Formulae *ϕ, ψ* are *(initially) equivalent*, written *ϕ* ≡*i ψ*, iff, for all words *u*, it holds that *u* |= *ϕ* ⇐⇒ *u* |= *ψ*. Formulae *ϕ, ψ* are *globally equivalent*, written *ϕ* ≡ *ψ*, iff, for all pointed words (*u, i*), it holds that (*u, i*) |= *ϕ* ⇐⇒ (*u, i*) |= *ψ*. Clearly, if two formulae are globally equivalent then they are also initially equivalent.

The following table defines some derived future operators and their past coun- terparts.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| future modality | | meaning | past | modality | meaning |
| F*ϕ* | *eventually* | *tt* U *ϕ* | P*ϕ* | *eventually in the past* | *tt* S *ϕ* |
| G*ϕ* | *always* | ¬F¬*ϕ* | H*ϕ* | *always in the past* | ¬P¬*ϕ* |
| Fs*ϕ* | *strict eventually* | XF*ϕ* | Ps*ϕ* | *eventually in the strict past* | YP*ϕ* |
| Gs*ϕ*  ∞  F *ϕ* | *strict always*  *inﬁnitely often* | ¬Fs¬*ϕ*  GF*ϕ* | Hs*ϕ*  I*ϕ* | *always in the strict past*  *initially* | ¬Ps¬*ϕ*  HP*ϕ* |

Given a set {*O*1*,... , On*} of modalities, then LTL(*O*1*,... , On*) denotes an LTL fragment containing all formulae with modalities *O*1*,... , On* only. Such a frag- ment is called *basic* if it contains future operators only or with each future op- erator it contains its past counterpart. For example, the fragment LTL(F*,* S) is not basic. Figure [1](#_bookmark3) shows an expressiveness hierarchy of all studied basic LTL fragments. Indeed, every basic LTL fragment using standard [7](#_bookmark2) modalities is equiva- lent to one of the fragments in the hierarchy, where equivalence between fragments means that every formula of one fragment can be effectively translated into an ini- tially equivalent formula of the other fragment and vice versa. We also mind the result of [[9](#_bookmark19)] stating that each LTL formula can be converted to the one which em- ploys future operators only, i.e. LTL(U*,* X) ≡*i* LTL(U*,* S*,* X*,* Y). However note that LTL(Fs*,* Ps*,* Gs*,* Hs) ≡ LTL(Fs*,* Ps) is strictly more expressive than LTL(Fs*,* Gs) as can be exemplified by a formula Fs(*b* ∧ Hs*a*) ≡*i a* ∧ X(*a* U *b*). We refer to [[20](#_bookmark30)] for greater detail.

This paper deals with the following two verification problems. Let F be an LTL fragment. The *model checking problem* for F and wPRS is to decide, for any given formula *ϕ* ∈ F and any given wPRS system Δ, whether *L*(Δ) |= *ϕ* holds. Further, given any formula *ϕ* ∈ F, any wPRS system Δ, and any nonterminal state *pt* of Δ, the *pointed model checking problem* for F and wPRS is to decide whether *L*(*pt,* Δ) |= *ϕ*; here *L*(*pt,* Δ) denotes the set of all pointed words (*u, i*) such that

*a*

0

Δ has a (finite or infinite) run *p*0*t*0 −→

*a*

*p*1*t*1 −→

1

*a*i−1

*...* −→

*a*i

*piti* −→

*...* satisfying

*u* = *a*0*a*1*a*2 *...* and *pt* = *piti*.

7 By standard modalities we mean the ones defined here and also other commonly used modalities like *strict until*, *release*, *weak until*, etc. However, it is well possible that one can define a new modality such that there is a basic fragment not equivalent to any of the fragments in the hierarchy.

LTL(U*,* X) ≡*¸i* FO3

*¸¸¸¸*

*¸¸*

*¸¸¸¸*

*¸¸*

*¸¸¸¸*

LTL(U*,* Fs*,* S*,* Ps)

*ssss*

*sss*

s

LTL(F*,* X*,* P*,* Y) ≡ FO2

*, \_ ¸*

*ssss*

LTL( U *,* F )

*¸¸¸¸¸ .*

*¸¸¸*

*¸¸¸*

*¸ s¸ss*

*sss ¸*

*c ¸¸¸¸*

*sss ,*

LTL(F*,* X)

*¸*

*ı* LTL(Fs*,* Ps) ≡ FO2[*<*] *¸ ¸*

LTL(U)

*¸¸¸¸¸¸¸ı*

*¸¸¸*

*¸¸¸¸¸*

*¸¸¸ ¸*

*¸¸¸¸*

*¸ \_ \_ \_ \_ \_*

*s ¸¸¸ ¸*

∞ *c*

LTL(F *,* G )

*¸¸¸*

LTL(F*,* P)

LTL( F *,* X) *c* s s

*, ¸ ¸¸¸¸¸*

*¸¸¸¸¸¸¸*

*\_ \_ \_ \_ \_ \_*

*¸ ¸¸¸*

*¸¸¸¸*

*¸¸¸¸¸¸¸*

*¸¸¸¸¸¸¸*

*¸¸¸¸¸¸¸*

*¸¸¸*

LTL(F*,* G)

LTL(X)

*¸¸¸¸¸¸¸*

∞

LTL( F)

*¸¸¸**¸¸*

*¸¸¸¸¸*

*¸¸¸¸¸*

*¸¸¸¸¸*

*¸*

*...*

*...*

*..*

*..*

*...*

*..*

*...*

LTL()

Fig. 1. The hierarchy of basic LTL fragments with respect to the initial equivalence. The dashed line shows the decidability boundary of the model checking problem for wPRS.

# Main Result

In [[4](#_bookmark14)], we have shown that the model checking problem is decidable for LTL(Fs*,* Gs). Before we prove that the problem remains decidable even for a more expressive fragment LTL(Fs*,* Ps), we recall the basic structure of the proof for LTL(Fs*,* Gs).

First, the proof shows that every LTL(Fs*,* Gs) formula can be effectively trans- lated into an equivalent disjunction of so-called *α-formulae*, which are defined be- low. Note that LTL() denotes the fragment of formulae without any modality,

i.e. boolean combinations of actions. In what follows, we use *ϕ*1 U+ *ϕ*2 to abbreviate *ϕ*1 ∧ X(*ϕ*1 U *ϕ*2). Let *δ* = *θ*1*O*1*θ*2*O*2 *... θnOnθn*+1, where *n >* 0, each *θi* ∈ LTL(), *On* is ‘∧ Gs’, and, for each *i < n*, *Oi* is either ‘U’ or ‘U+’ or ‘∧ X’. Further, let B ⊆ LTL() be a finite set. An *α*-formula is defined as

*α*(*δ,* B)= *θ*1*O*1(*θ*2*O*2 *...* (*θnOnθn*+1) *.. .*) ∧ GsFs*ψ .*

*ψ*∈B

Hence, a word *u* satisfies *α*(*δ,* B) iff *u* can be written as a concatenation *v*1*.v*2 *... vn*+1

of words, where

* each word *vi* consists only of actions satisfying *θi* and
* |*vi*| ≥ 0 if *i* = *n* +1 or *Oi* is ‘U’,
* |*vi*| *>* 0 if *Oi* is ‘U+’,
* |*vi*| =1 if *Oi* is ‘∧ X’ or ‘∧ Gs’,
* and *vn*+1 satisfies GsFs*ψ* for every *ψ* ∈ B.

Second, decidability of the model checking problem for LTL(Fs*,* Gs) is then a di- rect consequence of the following theorem.

Theorem 3.1 ([[4](#_bookmark14)]) *The problem whether any given wPRS systems has a run sat- isfying any given α-formula is decidable.*

To prove decidability for LTL(Fs*,* Ps), we show that every LTL(Fs*,* Ps) formula can be effectively translated into a disjunction of P*α*-formulae. Intuitively, a P*α*-formula is a conjunction of an *α*-formula and a past version of the *α*-formula. A formal definition of a P*α*-formula makes use of *ϕ*1 S+ *ϕ*2 to abbreviate *ϕ*1 ∧ Y(*ϕ*1 S *ϕ*2).

Definition 3.2 Let *η* = *ι*1*P*1*ι*2*P*2 *... ιmPmιm*+1, where *m >* 0, each *ιj* ∈ LTL(), and, for each *j < m*, *Pj* is either ‘S’ or ‘S+’ or ‘∧ Y’. Further, let *α*(*δ,* B) be an *α*-formula. Then a P*α-formula* is defined as

P*α*(*η, δ,* B)= *ι*1*P*1(*ι*2*P*2 *...* (*ιmPmιm*+1) *.. .*) ∧ *α*(*δ,* B) *.*

Note that the definition of a P*α*-formula does not contain any past counterpart of ∧*ψ*∈BGsFs*ψ* as every history is finite — the semantics of LTL is given in terms of words with a fixed beginning.

Therefore, a pointed word (*u, k*) |= P*α*(*η, δ,* B) if and only if (*u, k*) satisfies *α*(*δ,* B) and *a*0 *... ak*−1*ak* can be written as a concatenation *vm*+1*.vm ... v*2*.v*1, where each word *vi* consists only of actions satisfying *ιi* and

* |*vi*| ≥ 0 if *i* = *m* +1 or *Pi* is ‘S’,
* |*vi*| *>* 0 if *Pi* is ‘S+’,
* |*vi*| =1 if *Pi* is ‘∧ Y’ or ‘∧ Hs’.

The proof of the following lemma is intuitively clear but it is quite a technical exercise, see [[18](#_bookmark28)] for some hints.

Lemma 3.3 *Let ϕ be a* P*α-formula and p* ∈ *LTL*()*. Formulae* X*ϕ,* Y*ϕ, p* U *ϕ, p* S *ϕ,* Fs*ϕ,* Ps(*ϕ*)*, as well as, a conjunction of* P*α-formulae can be effectively converted into a globally equivalent disjunction of* P*α-formulae.*

Theorem 3.4 *Every LTL*(Fs*,* Ps) *formula ϕ can be translated into a globally equiv- alent disjunction of* P*α-formulae.*

Proof. As Fs*,* Gs and Ps*,* Hs are dual modalities, we can assume that every LTL(Fs*,* Gs*,* Ps*,* Hs) formula contains negations only in front of actions. Given an LTL(Fs*,* Gs*,* Ps*,* Hs) formula *ϕ*, we construct a finite set *Aϕ* of *α*-formulae such that *ϕ* is equivalent to the disjunction of formulae in *Aϕ*. Although our proof looks like by induction on the structure of *ϕ*, it is in fact by induction on the length of *ϕ*. Thus, if *ϕ* /∈ LTL(), then we assume that for every LTL(Fs*,* Gs*,* Ps*,* Hs) formula *ϕ*' shorter

than *ϕ* we can construct the corresponding set *Aϕ*' . In this proof, *p* represents a formula of LTL(). The structure of *ϕ* fits into one of the following cases.

* + *p* Case *p*: In this case, *ϕ* is equivalent to *p* ∧ Gs*tt*. Hence *Aϕ* = {P*α*(*tt* ∧ Hs*tt,p* ∧

Gs*tt,* ∅)}.

•∨ Case *ϕ*1 ∨ *ϕ*2: Due to induction hypothesis, we can assume that we have sets

*Aϕ*1 and *Aϕ*2 . Clearly, *Aϕ* = *Aϕ*1 ∪ *Aϕ*2 .

•∧ Case *ϕ*1 ∧ *ϕ*2: Due to Lemma [3.3](#_bookmark6), *Aϕ* can be constructed from the sets *Aϕ*1

and *Aϕ*2 .

* + Fs Case Fs*ϕ*1: Due to Lemma [3.3](#_bookmark6), the set *Aϕ* can be constructed from the set *Aϕ*1 .
  + Ps Case Ps*ϕ*1: Due to Lemma [3.3](#_bookmark6), the set *Aϕ* can be constructed from the set

*Aϕ*1 .

* + Gs Case Gs*ϕ*1 is divided into the following subcases according to the structure of *ϕ*1 :
    - *p* Case Gs*p*: As Gs*p* is equivalent to *tt* ∧ Gs*p*, we set *Aϕ* = {P*α*(*tt* ∧ Hs*tt, tt* ∧

Gs*p,* ∅)}.

◦∧ Case Gs(*ϕ*2 ∧ *ϕ*3): As Gs(*ϕ*2 ∧ *ϕ*3) ≡ (Gs*ϕ*2) ∧ (Gs*ϕ*3), the set *Aϕ* can be constructed from *A*Gs *ϕ*2 and *A*Gs *ϕ*3 using Lemma [3.3](#_bookmark6). Note that *A*Gs *ϕ*2 and *A*Gs *ϕ*3 can be constructed because Gs*ϕ*2 and Gs*ϕ*3 are shorter than Gs(*ϕ*2 ∧ *ϕ*3).

* + - Fs Case GsFs*ϕ*2: This case is again divided into the following subcases.

−*p* Case GsFs*p*: As *p* ∈ LTL(), we directly set *Aϕ* = {P*α*(*tt* ∧ Hs*tt, tt* ∧

Gs*tt,* {*p*})}.

−∨ Case GsFs(*ϕ*3 ∨ *ϕ*4): As GsFs(*ϕ*3 ∨ *ϕ*4) ≡ (GsFs*ϕ*3) ∨ (GsFs*ϕ*4), we set *Aϕ* =

*A*Gs Fs *ϕ*3 ∪ *A*Gs Fs *ϕ*4 .

−∧ Case GsFs(*ϕ*3 ∧ *ϕ*4): This case is also divided into subcases depending on the formulae *ϕ*3 and *ϕ*4.

∗*p* Case GsFs(*p*3 ∧ *p*4): As *p*3 ∧ *p*4 ∈ LTL(), this subcase has already been covered by Case GsFs*p*.

∗∨ Case GsFs(*ϕ*3 ∧ (*ϕ*5 ∨ *ϕ*6)): As GsFs(*ϕ*3 ∧ (*ϕ*5 ∨ *ϕ*6)) ≡ GsFs(*ϕ*3 ∧ *ϕ*5) ∨

GsFs(*ϕ*3 ∧ *ϕ*6), we set *Aϕ* = *A*Gs Fs (*ϕ*3 ∧*ϕ*5) ∪ *A*Gs Fs (*ϕ*3 ∧*ϕ*6).

∗Fs Case GsFs(*ϕ*3 ∧ Fs*ϕ*5): As GsFs(*ϕ*3 ∧ Fs*ϕ*5) ≡ (GsFs*ϕ*3) ∧ (GsFs*ϕ*5), the set

*Aϕ* can be constructed from *A*Gs Fs *ϕ*3 and *A*Gs Fs *ϕ*5 using Lemma [3.3](#_bookmark6).

∗Ps Case GsFs(*ϕ*3 ∧ Ps*ϕ*5): As GsFs(*ϕ*3 ∧ Ps*ϕ*5) ≡ (GsFs*ϕ*3) ∧ (GsFsPs*ϕ*5), the set *Aϕ* can be constructed from *A*Gs Fs *ϕ*3 and *A*Gs Fs Ps*ϕ*5 using Lemma [3.3](#_bookmark6).

∗Gs Case GsFs(*ϕ*3 ∧ Gs*ϕ*5): As GsFs(*ϕ*3 ∧ Gs*ϕ*5) ≡ (GsFs*ϕ*3) ∧ (GsFsGs*ϕ*5), the

set *Aϕ* can be constructed from *A*Gs Fs *ϕ*3 and *A*Gs Fs Gs*ϕ*5 using Lemma [3.3](#_bookmark6).

∗Hs Case GsFs(*ϕ*3 ∧ Hs*ϕ*5): As GsFs(*ϕ*3 ∧ Hs*ϕ*5) ≡ (GsFs*ϕ*3) ∧ (GsFsHs*ϕ*5), the set *Aϕ* can be constructed from *A*Gs Fs *ϕ*3 and *A*Gs Fs Hs*ϕ*5 using Lemma [3.3](#_bookmark6).

−Fs Case GsFsFs*ϕ*3: As GsFsFs*ϕ*3 ≡ GsFs*ϕ*3, we set *Aϕ* = *A*Gs Fs *ϕ*3 .

−Ps Case GsFsPs*ϕ*3: A pointed word (*u, i*) satisfies GsFsPs*ϕ*3 iff *i* = |*u*|− 1 or *u* is an infinite word satisfying F*ϕ*3. Note that Gs¬*tt* is satisfied only by finite words at their last position. Further, a word *u* satisfies (Fs*tt*) ∧ (GsFs*tt*) iff *u* is infinite. Thus, GsFsPs*ϕ*3 ≡ (Gs¬*tt*) ∨ *ϕ*' where *ϕ*' = (Fs*tt*) ∧ (GsFs*tt*) ∧ (*ϕ*3 ∨ Ps*ϕ*3 ∨ Fs*ϕ*3). Hence, *Aϕ* = *A*Gs¬*tt* ∪ *Aϕ*' where *Aϕ*' is constructed from *A*Fs *tt*,

*A*Gs Fs *tt*, and *Aq*3 ∪ *A*Ps *q*3 ∪ *A*Fs *q*3 using Lemma [3.3](#_bookmark6).

−Gs Case GsFsGs*ϕ*3: A pointed word (*u, i*) satisfies GsFsGs*ϕ*3 iff *i* = |*u*| − 1 or *u* is an infinite word satisfying FsGs*ϕ*3. Thus, GsFsGs*ϕ*3 ≡ (Gs¬*tt*) ∨ *ϕ*' where *ϕ*' = (Fs*tt*) ∧ (GsFs*tt*) ∧ (FsGs*ϕ*3). Hence, *Aq* = *A*G ¬*tt* ∪ *Aq*' where *Aq*' is constructed from *A*Fs *tt*, *A*Gs Fs *tt*, and *A*Fs Gs *q*3 using Lemma [3.3](#_bookmark6).

s

−Hs Case GsFsHs*ϕ*3: A pointed word (*u, i*) satisfies GsFsHs*ϕ*3 iff *i* = |*u*|− 1 or

*u* is an infinite word satisfying G*ϕ*3. Thus, GsFsHs*ϕ*3 ≡ (Gs¬*tt*) ∨ *ϕ*' where *ϕ*' = (Fs*tt*)∧(GsFs*tt*)∧(*ϕ*3 ∧Hs*ϕ*3 ∧Gs*ϕ*3). Hence, *Aq* = *A*G ¬*tt* ∪*Aq*' where *Aq*' is constructed from *A*Fs *tt*, *A*Gs Fs *tt*, *Aq*3 , *A*Hs *q*3 , and *A*Gs *q*3 using Lemma [3.3](#_bookmark6).

s

* Ps Case GsPs*ϕ*2: A pointed word (*u, i*) satisfies GsPs*ϕ*2 iff *i* = |*u*| − 1 or (*u, i*)

satisfies P*ϕ*2. Hence, *Aq* = *A*Gs¬*tt* ∪ *Aq*2 ∪ *A*Ps *q*2 .

◦∨ Case Gs(*ϕ*2 ∨ *ϕ*3): According to the structure of *ϕ*2 and *ϕ*3, there are the following subcases.

−*p* Case Gs(*p*2 ∨*p*3): As *p*2∨*p*3 ∈ LTL(), this subcase has already been covered by Case Gs*p*.

−∧ Case Gs(*ϕ*2∨(*ϕ*4∧*ϕ*5)): As Gs(*ϕ*2∨(*ϕ*4∧*ϕ*5)) ≡ Gs(*ϕ*2∨*ϕ*4)∧Gs(*ϕ*2∨*ϕ*5), the set *Aq* can be constructed from *A*Gs(*q*2 ∨*q*4) and *A*Gs(*q*2 ∨*q*5) using Lemma [3.3](#_bookmark6).

−Fs Case Gs(*ϕ*2 ∨ Fs*ϕ*4): It holds that Gs(*ϕ*2 ∨ Fs*ϕ*4) ≡ (Gs*ϕ*2) ∨ Fs(Fs*ϕ*4 ∧

Gs*ϕ*2) ∨ GsFs*ϕ*4. Therefore, the set *Aq* can be constructed as *A*Gs*q*2 ∪ *A*Fs(Fs *q*4∧Gs *q*2) ∪ *A*GsFs *q*4 , where *A*Fs(Fs *q*4 ∧Gs*q*2 ) is obtained from *A*Fs *q*4 and *A*Gs*q*2 using Lemma [3.3](#_bookmark6).

−Hs Case Gs(*ϕ*2∨Hs*ϕ*4): As Gs(*ϕ*2∨Hs*ϕ*4) ≡ (Gs*ϕ*2)∨Fs(Hs*ϕ*4∧Gs*ϕ*2)∨GsHs*ϕ*4. Hence, *Aq* = *A*Gs*q*2 ∪ *A*Fs(Hs*q*4∧Gs*q*2) ∪ *A*(GsHs*q*4) where *A*Fs(Hs*q*4∧Gs*q*2) can be obtained from *A*Hs*q*4 and *A*Gs *q*2 using Lemma [3.3](#_bookmark6).

−Gs*,* Ps There are only the following six subcases (the others fit to some of the

previous cases).

1. Case Gs( *q*'∈*G* Gs*ϕ*'): It holds that Gs( *q*'∈*G* Gs*ϕ*') ≡ (Gs¬*tt*) ∨

*q*'∈*G* XGs*q* XGs*q* Gs*q*

*q*'∈*G*

s

(XGs*ϕ*'). Therefore, the set *Aq* can be constructed as *A*G ¬*tt* ∪

*A* ' where each *A* ' is obtained from *A* ' using Lemma [3.3](#_bookmark6).

1. Case Gs(*p*2 ∨ *q*'∈*G* Gs*ϕ*'): As Gs(*p*2 ∨ *q*'∈*G* Gs*ϕ*') ≡ (Gs*p*2) ∨

*q*'∈*G*

(X(*p*2 U (Gs*ϕ*'))), the set *Aq* can be constructed as *A*G *p* ∪

s 2

*q*'∈*G A*X(*p*2 U (Gs *q*')) where each *A*X(*p*2 U (Gs *q*')) is obtained from *A*Gs *q*' us- ing Lemma [3.3](#_bookmark6).

1. Case Gs( *q*''∈*P* Ps*ϕ*''): It holds that Gs( *q*''∈*P* Ps*ϕ*'') ≡ (Gs¬*tt*) ∨

*q*''∈*P* XPs*q* XPs*q* Ps*q*

*q*''∈*P*

s

(XPs*ϕ*''). Therefore, the set *Aq* can be constructed as *A*G ¬*tt* ∪

*A* '' where each *A* '' is obtained from *A* '' using Lemma [3.3](#_bookmark6).

1. Case Gs(*p*2 ∨ *q*''∈*P* Ps*ϕ*''): As Gs(*p*2 ∨ *q*''∈*P* Ps*ϕ*'') ≡ (Gs*p*2) ∨

*q*''∈*P*

s 2

(X(*p*2 U (Ps*ϕ*''))), the set *Aq* can be constructed as *A*G *p* ∪

*q*''∈*P A*X(*p*2 U (Ps *q*'')) where each *A*X(*p*2 U (Ps *q*'')) is obtained from *A*Ps *q*'' using

Lemma [3.3](#_bookmark6).

1. Case Gs( *q*'∈*G* Gs*ϕ*' ∨ *q*''∈*P* Ps*ϕ*''): As Gs( *q*'∈*G* Gs*ϕ*' ∨ *q*''∈*P* Ps*ϕ*'') ≡ (G ¬*tt*) ∨ (XG *ϕ*') ∨ (XP *ϕ*''), the set *A* can be constructed as *A*Gs¬*tt* ∪ *q*'∈*G A*XGs *q*' ∪ *q*''∈*P A*XPs *q*'' where each *A*XGs *q*' is obtained from *A*Gs *q*' and each *A*XPs *q*'' is obtained from *A*Ps *q*'' using Lemma [3.3](#_bookmark6).

s *q*'∈*G* s *q*''∈*P* s *q*

1. Case G (*p* ∨ G *ϕ*' ∨ P *ϕ*''): As G (*p* ∨ G *ϕ*' ∨

s 2 *q*'∈*G* s *q*''∈*P* s s 2 *q*'∈*G* s

*q*''∈*P* Ps*ϕ*'') ≡ (Gs*p*2) ∨ *q*'∈*G*(X(*p*2 U (Gs*ϕ*'))) ∨ *q*''∈*P* (X(*p*2 U (Ps*ϕ*''))), the set can be constructed as

*q* Gs*p*2 *q*'∈*G* X(*p*2 U (Gs*q* ))

*A A* ∪ *A* ' ∪

*q*''∈*P A*X(*p*2 U (Ps *q*'')) where each *A*X(*p*2 U (Gs *q*')) is obtained from *A*Gs *q*' and each *A*X(*p*2 U (Ps *q*'')) is obtained from *A*Ps *q*'' using Lemma [3.3](#_bookmark6).

* + Gs Case GsGs*ϕ*2: As Gs(Gs*ϕ*2) ≡ (Gs¬*tt*) ∨ (XGs*ϕ*2), the set *Aq* can be

constructed as *A*Gs¬*tt* ∪ *A*XGs *q*2 where *A*XGs*q*2 is obtained from *A*Gs*q*2 using Lemma [3.3](#_bookmark6).

* + Hs Case GsHs*ϕ*2: A pointed word (*u, i*) satisfies Gs(Hs*ϕ*2) iff *i* = |*u*| − 1 or (*u,* |*u*|− 1) satisfies Hs*ϕ*2 or *u* is infinite and all its positions satisfy *ϕ*2. Hence, *Aq* = *A*Gs¬*tt* ∪ *A*Fs((Gs¬*tt*)∧(Hs*q*2)) ∪ *A*(Hs*q*2)∧*q*2∧(Gs*q*2) where *A*Fs((Gs¬*tt*)∧(Hs*q*2)) and *A*(Hs *q*2 )∧*q*2 ∧(Gs *q*2) are obtained from *A*Gs¬*tt*, *A*Hs *q*2 , *Aq*2 , and *A*Gs *q*2 using Lemma [3.3](#_bookmark6).
  + Hs Case Hs*ϕ*1: This case is divided into the following subcases according to the structure of *ϕ*1.
    - *p* Case Hs*p*: As Hs*p* is globally equivalent to *tt* ∧ Hs*p*, we set *Aq* = {P*α*(*tt* ∧

Hs*p, tt* ∧ Gs*tt,* ∅)}.

◦∧ Case Hs(*ϕ*2 ∧ *ϕ*3): As Hs(*ϕ*2 ∧ *ϕ*3) ≡ (Hs*ϕ*2) ∧ (Hs*ϕ*3), the set *Aq* can be constructed from *A*Hs*q*2 and *A*Hs*q*3 using Lemma [3.3](#_bookmark6).

* + - Fs Case HsFs*ϕ*2: A pointed word (*u, i*) satisfies HsFs*ϕ*2 iff *i* =0 or (*u, i*) satisfies

F*ϕ*2. Note that Hs¬*tt* is satisfied by (*u, i*) only if *i* = 0. Therefore, *Aq* =

*A*Hs¬*tt* ∪ *Aq*2 ∪ *A*Fs*q*2 .

* + - Ps Case HsPs*ϕ*2: A pointed word (*u, i*) satisfies HsPs*ϕ*2 iff *i* = 0. Therefore,

*Aq* = *A*Hs¬*tt*.

◦∨ Case Hs(*ϕ*2 ∨ *ϕ*3): According to the structure of *ϕ*2 and *ϕ*3, there are the following subcases.

−*p* Case Hs(*p*2∨*p*3): As *p*2∨*p*3 ∈ LTL(), this subcase has already been covered by Case Hs*p*.

−∧ Case Hs(*ϕ*2 ∨ (*ϕ*4 ∧ *ϕ*5)): As Hs(*ϕ*2 ∨ (*ϕ*4 ∧ *ϕ*5)) ≡ Hs(*ϕ*2 ∨ *ϕ*4) ∧ Hs(*ϕ*2 ∨ *ϕ*5), the set *Aq* can be constructed from *A*Hs(*q*2 ∨*q*4) and *A*Hs(*q*2 ∨*q*5 ) using Lemma [3.3](#_bookmark6).

−Ps Case Hs(*ϕ*2 ∨Ps*ϕ*4): It holds that Hs(*ϕ*2 ∨Ps*ϕ*4) ≡ (Hs*ϕ*2)∨Ps(Ps*ϕ*4 ∧Hs*ϕ*2). Therefore, the set *Aq* can be constructed as *A*Hs *q*2 ∪ *A*Ps(Ps *q*4∧Hs *q*2), where *A*Ps(Ps *q*4 ∧Hs*q*2 ) is obtained from *A*Ps*q*4 and *A*Hs *q*2 using Lemma [3.3](#_bookmark6).

−Gs Case Hs(*ϕ*2 ∨ Gs*ϕ*4): As Hs(*ϕ*2 ∨ Gs*ϕ*4) ≡ (Hs*ϕ*2) ∨ Ps(Gs*ϕ*4 ∧ Hs*ϕ*2), *Aq* is

constructed as *A*Hs *q*2 ∪ *A*Ps(Gs *q*4∧Hs *q*2) where *A*Ps(Gs *q*4∧Hs *q*2) is obtained from

*A*Gs *q*4 and *A*Hs *q*2) using Lemma [3.3](#_bookmark6).

−Fs*,* Hs There are only the following six subcases (the others fit to some of the previous cases).

1. Case Hs( *q*'∈*F* Fs*ϕ*'): It holds that Hs( *q*'∈*F* Fs*ϕ*') ≡ (Hs¬*tt*) ∨

*q*'∈*F* YFs*q* YFs*q* Fs*q*

*q*'∈*F*

s

(YFs*ϕ*'). Therefore, the set *Aq* can be constructed as *A*H ¬*tt* ∪

*A* ' where each *A* ' is obtained from *A* ' using Lemma [3.3](#_bookmark6).

1. Case Hs(*p*2 ∨ *q*'∈*F* Fs*ϕ*'): As Hs(*p*2 ∨ *q*'∈*F* Fs*ϕ*') ≡ (Hs*p*2) ∨

*q*'∈*F*

s 2

(Y(*p*2 S (Fs*ϕ*'))), the set *Aq* can be constructed as *A*H *p* ∪

*q*'∈*F A*Y(*p*2 S (Fs *q*')) where each *A*Y(*p*2 S (Fs *q*')) is obtained from *A*Fs *q*' using Lemma [3.3](#_bookmark6).

1. Case Hs( *q*''∈*H* Hs*ϕ*''): It holds that Hs( *q*''∈*H* Hs*ϕ*'') ≡ (Hs¬*tt*) ∨

*q*''∈*H*

(YHs*ϕ*''). Therefore, the set *Aq* can be constructed as *A*H ¬*tt* ∪

s

*q*''∈*H A*YHs*q*'' where each *A*YHs *q*'' is obtained from *A*Hs *q*'' using Lemma [3.3](#_bookmark6).

1. Case Hs(*p*2 ∨ *q*''∈*H* Hs*ϕ*''): As Hs(*p*2 ∨ *q*''∈*H* Hs*ϕ*'') ≡ (Hs*p*2) ∨

*q*''∈*H*

s 2

(Y(*p*2 S (Hs*ϕ*''))), the set *Aq* can be constructed as *A*H *p* ∪

*q*''∈*H A*Y(*p*2 S (Hs *q*'')) where each *A*Y(*p*2 S (Hs *q*'')) is obtained from *A*Hs *q*'' us-

ing Lemma [3.3](#_bookmark6).

1. Case Hs( *q*'∈*F* Fs*ϕ*' ∨ *q*''∈*H* Hs*ϕ*''): As Hs( *q*'∈*F* Fs*ϕ*' ∨ *q*''∈*H* Hs*ϕ*'') ≡ (H ¬*tt*) ∨ (YF *ϕ*') ∨ (YH *ϕ*''), the set *A* can be constructed as *A*Hs¬*tt* ∪ *q*'∈*F A*YFs *q*' ∪ *q*''∈*H A*YHs *q*'' where each *A*YFs *q*' is obtained from *A*Fs *q*' and each *A*YHs*q*'' is obtained from *A*Hs*q*'' using Lemma [3.3](#_bookmark6).

s 2 *q*'∈*F* s *q*''∈*H* s s 2 *q*'∈*F* s

s *q*'∈*F* s *q*''∈*H* s *q*

1. Case H (*p* ∨ F *ϕ*' ∨ H *ϕ*''): As H (*p* ∨ F *ϕ*' ∨

*q*''∈*H* Hs*ϕ*'') ≡ (Hs*p*2) ∨ *q*'∈*F* (Y(*p*2 S (Fs*ϕ*'))) ∨ *q*''∈*H* (Y(*p*2 S (Hs*ϕ*''))), the set *A* can be constructed as *A* ∪ *A* ' ∪

*q* Hs*p*2 *q*'∈*F* Y(*p*2 S (Fs*q* ))

*q*''∈*H A*Y(*p*2 S (Hs *q*'')) where each *A*Y(*p*2 S (Fs *q*')) is obtained from *A*Fs *q*' and each *A*Y(*p*2 S (Hs*q*'')) is obtained from *A*Hs *q*'' using Lemma [3.3](#_bookmark6).

* Gs Case HsGs*ϕ*2: A pointed word (*u, i*) satisfies Hs(Gs*ϕ*2) iff *i* = 0 or (*u,* 0)

satisfies Gs*ϕ*2. Hence, *Aq* = *A*Hs¬*tt* ∪ *A*Ps((Hs¬*tt*)∧(Gs*q*2)) where *A*Ps((Hs¬*tt*)∧(Gs*q*2))

is obtained from *A*Hs¬*tt* and *A*Gs *q*2 using Lemma [3.3](#_bookmark6).

* Hs Case HsHs*ϕ*2: As Hs(Hs*ϕ*2) ≡ (Hs¬*tt*) ∨ (YHs*ϕ*2), the set *Aq* can be constructed as *A*Hs¬*tt* ∪ *A*YHs*q*2 where *A*YHs *q*2 is obtained from *A*Hs *q*2 using Lemma [3.3](#_bookmark6).

Remark 3.5 In other words, we have just shown that LTL(Fs*,* Ps) is a semantic subset (with respect to global equivalence) of every formalism that is (i) able to express *p*, Gs*p*, Hs*p*, and GsFs*p*, where *p* ∈ LTL(); and (ii) is closed under disjunction, conjunction, and applications of X , Y , *p* U , and *p* S , where *p* ∈ LTL().

Now, using Theorem [3.1](#_bookmark4), we can easily solve the problem dual to the model checking problem, i.e. given any wPRS system and any P*α*-formula, to decide whether the system has a run satisfying the formula.

Theorem 3.6 *The problem whether any given wPRS system has a run satisfying a given* P*α-formula is decidable.*

Proof. A run over a nonempty (finite or infinite) word *u* = *a*0*a*1*a*2 *...* satisfies a formula *ϕ* iff (*u,* 0) |= *ϕ*. Moreover, (*u,* 0) |= P*α*(*η, δ,* B) iff (*a*0*,* 0) |= *η* and (*u,* 0) |= *α*(*δ,* B). Let *η* = *ι*1*P*1*ι*2*P*2 *... ιmPmιm*+1. It follows from the semantics of LTL that (*a*0*,* 0) |= *η* if and only if (*a*0*,* 0) |= *ιm* and *Pi* = S for all *i < m*. Therefore, the problem is to check whether *Pi* = S for all *i < m* and whether the given wPRS system has a run satisfying *ιm* ∧ *α*(*δ,* B). As *ιm* ∧ *α*(*δ,* B) can be easily translated into a disjunction of *α*-formulae, Theorem [3.1](#_bookmark4) finishes the proof.

As LTL(Fs*,* Ps) is closed under negation, Theorem [3.4](#_bookmark7) and Theorem [3.6](#_bookmark8) give us the following.

Corollary 3.7 *The model checking problem for wPRS and LTL*(Fs*,* Ps) *is decidable.*

Moreover, we can show that the pointed model checking problem is decidable for wPRS and LTL(Fs*,* Ps) as well. Again, we solve the dual problem.

Theorem 3.8 *Let* Δ *be a wPRS and pt be a reachable nonterminal state of* Δ*. The problem whether L*(*pt,* Δ) *contains a pointed word* (*u, i*) *satisfying any given* P*α-formula is decidable.*

Proof. Let Δ = (*M,* ≥*, R, p*0*, t*0) be a wPRS and *pt* be a reachable nonterminal state of Δ. We construct a wPRS Δ' = (*M,* ≥*, R*'*, p*0*, t*0*.X*) where *X* /∈ *Const* (Δ) is a fresh process constant, *f* /∈ *Act* (Δ) is a fresh action,

*R*' = *R* ∪ {(*p*(*t.X*) *‹a pX* )*,* (*pX ‹f pY* )*,* (*pY ‹a p*'*t*') | *pt* —*a*→ *p*'*t*'},

→ *a a* → *a a* →

and *Xa, Ya* /∈ *Const* (Δ) are fresh process constants for each *a* ∈ *Act* (Δ).

It is easy to see that (*u, i*) is in *L*(*pt,* Δ) iff *u* = *a*0*a*1 *... ai*−1*ai.f.ai.ai*+1 *...* is in *L*(Δ'). Hence, for any given P*α*-formula *ϕ* = P*α*(*η, δ,* У) we construct a P*α*-formula *ϕ*' = P*α*(*η, tt* Λ X*f* Λ X*δ,* У). We get that

*L*(*pt,* Δ) |= P*α*(*η, δ,* У) ⇐⇒ *L*(Δ') |= F(P*α*(*η, tt* Λ X*f* Λ X*δ,* У))

and due to Lemma [3.3](#_bookmark6) and Theorem [3.6](#_bookmark8) the proof is done.

As LTL(Fs*,* Ps) is closed under negation and Theorem [3.4](#_bookmark7) works with global equivalence, Theorem [3.8](#_bookmark9) give us the following.

Corollary 3.9 *The pointed model checking problem is decidable for wPRS and LTL*(Fs*,* Ps)*.*

# Conclusion

We have examined the model checking problem for basic LTL fragments with both future and past modalities and the PRS class, i.e. the class of infinite state system generated by Process Rewrite Systems (PRS), possibly enriched with a weak finite control unit (weakly extended PRS – wPRS). We have proved that the problem is decidable for wPRS and LTL(Fs*,* Ps), i.e. the fragment with modalities *strict even- tually*, *eventually in the strict past*, and derived modalities *strict always* and *always* *in the strict past*. [8](#_bookmark10) However, both these problems are at least as hard as the reach- ability problem for PN [[6](#_bookmark16)] (EXPSPACE-hard without any elementary upper bound known).

Note that the expressive power of the fragment LTL(Fs*,* Ps) semantically coincides with formulae of First-Order Monadic Logic of Order containing at most 2 variables

8 In fact, we have shown that the problem is decidable even for a more expressive fragment containing negations of disjunctions of so-called P*α*-formulae (see Definition [3.2](#_bookmark5)).

and no successor predicate (FO2[*<*]), and that First-Order Monadic Logic of Order containing at most 2 variables (FO2) coincides with an LTL(F*,* X*,* P*,* Y) fragment [[8](#_bookmark18)]. Further, let us recall our undecidability results for model checking of PA systems

∞

(a subclass of PRS) and fragments LTL( F *,* X) and LTL( U ), respectively (the former

with modalities *inﬁnitely often* and *next* only, the latter with *until* as the only modality), see [[4](#_bookmark14)].

Thus, we have located the borderline between decidability and undecidability of the problem for wPRS and the LTL fragments, as well as for wPRS and First-Order Monadic Logic of Order: it is decidable for FO2[*<*] and undecidable for FO2. For the sake of completeness, we note that the First-Order Monadic Logic of Order containing at most 3 variables (FO3) coincides with the set of all LTL formulae as well as with the full First-Order Monadic Logic of Order [[11](#_bookmark21),[10](#_bookmark20)]. Finally, we note that the decidability results are new for the PRS class too and they are illustrated by the decidability border in Figure [1](#_bookmark3).

# References

1. Bouajjani, A., J. Esparza and O. Maler, *Reachability Analysis of Pushdown Automata: Application to Model-Checking*, in: *Proc. of CONCUR’97*, LNCS 1243, 1997, pp. 135–150.
2. Bouajjani, A. and P. Habermehl, *Constrained Properties, Semilinear Systems, and Petri Nets*, in:

*Proc. of CONCUR’96*, LNCS 1119 (1996), pp. 481–497.

1. Bozzelli, L., *Model checking for process rewrite systems and a class of action-based regular properties*, in: *Proc. of VMCAI’05*, LNCS 3385 (2005), pp. 282–297.
2. Bozzelli, L., M. Kˇret´ınsky´, V. Rˇeha´k and J. Strejˇcek, *On decidability of LTL model checking for process rewrite systems*, in: *FSTTCS 2006*, LNCS 4337 (2006), pp. 248–259.
3. Burkart, O., D. Caucal, F. Moller and B. Steffen, *Verification on infinite structures*, in: *Handbook of Process Algebra* (2001), pp. 545–623.
4. Esparza, J., *On the Decidability of Model Checking for Several mu-calculi and Petri Nets*, in: *CAAP*, LNCS 787 (1994), pp. 115–129.
5. Esparza, J., *Grammars as Processes*, in: *Formal and Natural Computing*, LNCS 2300 (2002), pp. 277–297.
6. Etessami, K., M. Y. Vardi and T. Wilke, *First-order logic with two variables and unary temporal logic*, Information and Computation 179 (2002), pp. 279–295.
7. Gabbay, D., *The Declarative Past and Imperative Future: Executable Temporal Logic for Interactive Systems*, in: *Temporal Logic in Specification*, LNCS 398, 1987, pp. 409–448.
8. Gabbay, D., A. Pnueli, S. Shelah and J. Stavi, *On the Temporal Analysis of Fairness*, in: *Conference Record of the 7th Annual ACM Symposium on Principles of Programming Languages (POPL’80)* (1980), pp. 163–173.
9. Kamp, J. A. W., “Tense Logic and the Theory of Linear Order,” Ph.D. thesis, University of California, Los Angeles (1968).
10. Kˇret´ınsky´, M., V. Rˇeha´k and J. Strejˇcek, *Extended Process Rewrite Systems: Expressiveness and Reachability*, in: *Proceedings of CONCUR’04*, LNCS 3170 (2004), pp. 355–370.
11. Kˇret´ınsky´, M., V. Rˇeha´k and J. Strejˇcek, *On Extensions of Process Rewrite Systems: Rewrite Systems with Weak Finite-State Unit*, in: *Proceedings of INFINITY’03*, Electr. Notes Theor. Comput. Sci. 98 (2004), pp. 75–88.
12. Lichtenstein, O., A. Pnueli and L. D. Zuck, *The Glory of The Past*, in: *Logic of Programs*, LNCS 193, 1985, pp. 196–218.
13. Mayr, R., “Decidability and Complexity of Model Checking Problems for Infinite-State Systems,” Ph.D. thesis, Technische Universit¨at Mu¨nchen (1998).
14. Mayr, R., *Process rewrite systems*, Information and Computation 156 (2000), pp. 264–286.
15. Pnueli, A., *The Temporal Logic of Programs*, in: *Proc. 18th IEEE Symposium on the Foundations of Computer Science*, 1977, pp. 46–57.
16. Rˇeh´ak, V., “On Extensions of Process Rewrite Systems,” Ph.D. thesis, Faculty of Informatics, Masaryk University, Brno (2007).
17. Srba, J., “Roadmap of Infinite Results,” Current Trends In Theoretical Computer Science, The Challenge of the New Century Vol 2: Formal Models and Semantics, World Scientific Publishing Co., 2004, 337–350 pp.
18. Strejˇcek, J., “Linear Temporal Logic: Expressiveness and Model Checking,” Ph.D. thesis, Faculty of Informatics, Masaryk University, Brno (2004).