Revisiting Ackermann-Hardness for Lossy Counter Machines and Reset Petri Nets*

Ph. Schnoebelen

LSV, ENS Cachan, CNRS
61, av. Pdt. Wilson, F-94230 Cachan, France
www.lsv.ens-cachan.fr/~phs

Abstract. We prove that coverability and termination are not primitive-recursive for lossy counter machines and for Reset Petri nets.

1 Introduction

Lossy counter machines [16, 19] and Reset Petri nets [8] are two computational models that can be seen as weakened versions of Minsky counter machines. This weakness explains why some problems (e.g., termination) are decidable for these two models, while they are undecidable for the Turing-powerful Minsky machines.

While these positive results have been used in the literature, there also exists a negative side that has had much more impact. Indeed, we showed in [18] that decidable verification problems for lossy channel systems are Ackermann-hard and hence cannot be answered in primitive-recursive time or space. We also claimed that the construction used for lossy channels could be adapted for lossy counters and Reset Petri nets.

Hardness Theorem (in the Introduction of [18]). *Reachability, termination and coverability for lossy counter machines are Ackermann-hard.*
Termination and coverability for Reset Petri nets are Ackermann-hard.

These hardness results turned out to be relevant in several other areas. Using lossy counter machines, hardness results relying on the first half of the Hardness Theorem have been derived for a variety of logics and automata dealing with data words or data trees [6, 7, 14, 12, 20]. Ackermann-hardness has also been shown by reductions from Reset and Transfer nets, relying on the second half of the Hardness Theorem. Examples can be found in, e.g., [1, 13]. We refer to [3, 4] and the references therein for hardness inherited from lossy channel systems.

Our contribution. In this paper we prove the Hardness Theorem with a simplified construction. Compared to [18], we introduce three main simplifications:
1. We use counter machines and not channel systems, which is more direct since the crux of the construction is the computation of numerical functions.
2. We use a tail-recursive presentation of the F_m functions from the Fast-Growing Hierarchy. Thus we do not build our counter machines in nested stages like in [18]. As a

* Work supported by the Agence Nationale de la Recherche, grant ANR-06-SETIN-001.

consequence, the correctness of the numerical computations is obvious and we obtain a clearer view of how many counters are really used.
3. We do not define, nor compute, inverses of the F_m functions as done in [18]. Instead, the tail-recursive definition is a simple rewrite loop that can easily be run backwards.

In addition, we strove for extra simplicity. E.g., we use counter machines extended with simple primitives that make computing Ackermann's function less cumbersome.

There are several reasons for providing a new proof of an old result. First, the results are important and influential as demonstrated by the number and the variety of applications we listed above: they definitely deserve being revisited, polished, advertised, etc. Second, our original proof has already been adapted to yet other computational models (e.g., in [15]) and a simplified proof will probably be easier to adapt to further models. Finally, we note that the main contents of [18] is now obsolete since Ackermann-hardness is not optimal for lossy channel systems [3]. However, for lossy counter machines and Reset nets, the Hardness Theorem is optimal (see [17, 11]) and will not become obsolete.

Outline of the paper. Section 2 defines counter machines, both reliable and lossy. Section 3 builds counter machines that compute Ackermann's function. Section 4 puts Minsky machines *on a budget*, a gadget that is essential in Section 5 where the main reduction is given and the hardness of reachability and coverability for lossy counter machines is proved. We then show how to deal with reset nets in Section 6 and how to prove hardness of termination in Section 7. Some proofs have been omitted for lack of space: they can be found in the full version of this paper.

2 Counter machines, reliable and lossy

Counter machines are a model of computation where a finite-state control acts upon a finite number of *counters*, i.e., storage locations that hold a natural number. The computation steps are usually restricted to simple tests and updates. For Minsky machines, the tests are zero-tests and the updates are increments and decrements.

For our purposes, it will be convenient to use a slightly extended model that allows more concise constructions, and that will let us handle Reset nets smoothly.

2.1 Extended counter machines and Minsky machines

Formally, an *extended counter machine with n counters*, often just called a "counter machine" (a CM), is a tuple $M = (Loc, C, \Delta)$ where $Loc = \{\ell_1, \ldots, \ell_p\}$ is a finite set of *locations*, $C = \{\mathtt{c}_1, \ldots, \mathtt{c}_n\}$ is a finite set of *counters*, and $\Delta \subseteq Loc \times OP(C) \times Loc$ is a finite set of transition rules. The transition rules are depicted as directed edges (see Fig. 1, 2, and 3 below) between control locations labeled with an instruction $op \in OP(C)$ that is either a *guard* (a condition on the current contents of the counters for the rule to be firable), or an *update* (a method that modifies the contents of the counters), or both.

For CM's, the instruction set $OP(C)$ is given by the following abstract grammar:

$$\begin{array}{rlll} OP(C) \ni op ::= & \texttt{c=0?} & \text{/* zero test */} & \mid \texttt{c:=0} \quad \text{/* reset */} \\ \mid & \texttt{c>0? c--} & \text{/* decrement */} & \mid \texttt{c=c'?} \quad \text{/* equality test */} \\ \mid & \texttt{c++} & \text{/* increment */} & \mid \texttt{c:=c'} \quad \text{/* copy */} \end{array}$$

where $\texttt{c}, \texttt{c}'$ are any two counters in C. (We also allow a `no_op` instruction that does not test or modify the counters.)

A *Minsky machine* is a CM that only uses instructions among zero tests, decrements and increments (the first three types). Petri nets and Vector Addition Systems with States (VASS's) can be seen as counter machines that only use decrements and increments (i.e., Minsky machines without zero-tests).

2.2 Operational semantics

The operational semantics of a CM $M = (Loc, C, \Delta)$ is given under the form of transitions between its configurations. Formally, a *configuration* (written $\sigma, \theta, \ldots$) of M is a tuple $(\ell, \boldsymbol{a})$ with $\ell \in Loc$ representing the "current" control location, and $\boldsymbol{a} \in \mathbb{N}^C$, a C-indexed vector of natural numbers representing the current contents of the counters. If C is some $\{\texttt{c}_1, \ldots, \texttt{c}_n\}$, we often write $(\ell, \boldsymbol{a})$ under the form $(\ell, a_1, \ldots, a_n)$. Also, we sometimes use labels in vectors of values to make them more readable, writing, e.g., $\boldsymbol{a} = (0, \ldots, 0, \texttt{c}_k : 1, 0, \ldots, 0)$.

Regarding the behavior induced by the rules from Δ, there is a *transition* (also called a *step*) $\sigma \xrightarrow{\delta}_{\text{std}} \sigma'$ if, and only if, σ is some $(\ell, a_1, \ldots, a_n)$, σ' is some $(\ell', a'_1, \ldots, a'_n)$, $\Delta \ni \delta = (\ell, op, \ell')$ and either:
– op is $\texttt{c}_k\texttt{=0?}$ (zero test): $a_k = 0$, and $a'_i = a_i$ for all $i = 1, \ldots, n$, or
– op is $\texttt{c}_k\texttt{>0? c}_k\texttt{--}$ (decrement): $a'_k = a_k - 1$, and $a'_i = a_i$ for all $i \neq k$, or
– op is $\texttt{c}_k\texttt{++}$ (increment): $a'_k = a_k + 1$, and $a'_i = a_i$ for all $i \neq k$, or
– op is $\texttt{c}_k\texttt{:=0}$ (reset): $a'_k = 0$, and $a'_i = a_i$ for all $i \neq k$, or
– op is $\texttt{c}_k\texttt{=c}_p\texttt{?}$ (equality test): $a_k = a_p$, and $a'_i = a_i$ for all $i = 1, \ldots, n$, or
– op is $\texttt{c}_k\texttt{:=c}_p$ (copy): $a'_k = a_p$, and $a'_i = a_i$ for all $i \neq k$.
(The steps carry a "std" subscript to emphasize that we are considering the usual, standard, operational semantics of counter machines, where the behavior is *reliable*.)

As usual, we write $\sigma \xrightarrow{\Delta}_{\text{std}} \sigma'$, or just $\sigma \rightarrow_{\text{std}} \sigma'$, when $\sigma \xrightarrow{\delta}_{\text{std}} \sigma'$ for some $\delta \in \Delta$. Chains $\sigma_0 \rightarrow_{\text{std}} \sigma_1 \rightarrow_{\text{std}} \cdots \rightarrow_{\text{std}} \sigma_m$ of consecutive steps, also called *runs*, are denoted $\sigma_0 \rightarrow^*_{\text{std}} \sigma_m$, and also $\sigma_0 \rightarrow^+_{\text{std}} \sigma_m$ when $m > 0$. Steps may also be written more precisely under the form $M \vdash \sigma \rightarrow_{\text{std}} \sigma'$ when several counter machines are at hand and we want to be explicit about where the steps take place.

For a vector $\boldsymbol{a} = (a_1, \ldots, a_n)$, or a configuration $\sigma = (\ell, \boldsymbol{a})$, we let $|\boldsymbol{a}| = |\sigma| = \sum_{i=1}^n a_i$ denote its *size*. For $N \in \mathbb{N}$, we say that a run $\sigma_0 \rightarrow \sigma_1 \rightarrow \cdots \rightarrow \sigma_m$ is N-bounded if $|\sigma_i| \leq N$ for all $i = 0, \ldots, n$.

2.3 Lossy counter machines

Lossy counter machines are counter machines where the contents of the counters can decrease non-deterministically (the machine can "leak", or "lose data").

Technically, it is more convenient to see lossy machines as counter machines with a different operational semantics (and not as a special class of machines): thus it is possible to use simultaneously the two semantics and relate them.

Formally, this is defined via the introduction of a partial ordering between the configurations of M:

$$(\ell, a_1, ..., a_n) \leq (\ell', a'_1, ..., a'_n) \stackrel{\text{def}}{\Leftrightarrow} \ell = \ell' \wedge a_1 \leq a'_1 \wedge \cdots \wedge a_n \leq a'_n.$$

$\sigma \leq \sigma'$ can be read as "σ is σ' after some losses (possibly none)".

Now "lossy" steps, denoted $M \vdash \sigma \xrightarrow{\delta}_{\text{lossy}} \sigma'$, are given by the following definition:

$$\sigma \xrightarrow{\delta}_{\text{lossy}} \sigma' \stackrel{\text{def}}{\Leftrightarrow} \exists \theta, \theta' : (\sigma \geq \theta \wedge \theta \xrightarrow{\delta}_{\text{std}} \theta' \wedge \theta' \geq \sigma'). \qquad (*)$$

Note that reliable steps are a special case of lossy steps:

$$M \vdash \sigma \rightarrow_{\text{std}} \sigma' \text{ implies } M \vdash \sigma \rightarrow_{\text{lossy}} \sigma'. \qquad (\dagger)$$

2.4 Behavioral problems on counter machines

We consider the following decision problems:

Reachability: given a CM M and two configurations σ_{ini} and σ_{goal}, is there a run $M \vdash \sigma_{\text{ini}} \rightarrow^* \sigma_{\text{goal}}$?

Coverability: given a CM M and two configurations σ_{ini} and σ_{goal}, is there a run $M \vdash \sigma_{\text{ini}} \rightarrow^* \sigma$ for some configuration $\sigma \geq \sigma_{\text{goal}}$ that covers σ_{goal}?

(Non-)Termination: given a CM M and a configuration σ_{ini}, is there an infinite run $M \vdash \sigma_{\text{ini}} \rightarrow \sigma_1 \rightarrow \cdots \rightarrow \sigma_n \rightarrow \cdots$?

These problems are parameterized by the class of counter machines we consider and, more importantly, by the operational semantics that is assumed. Recall that reachability and termination are decidable for lossy counter machines, i.e., counter machines assuming lossy steps [16, 19]. Observe that, for lossy machines, reachability and coverability coincide (except for runs of length 0). For the standard semantics, the same problems are undecidable for Minsky machines but become decidable for VASS's and, except for reachability, for Reset nets (see Section 6).

3 The Fast-Growing Hierarchy

The *Fast-Growing Hierarchy* [10] turns the class of all primitive-recursive functions into a strict cumulative hierarchy built from a sequence $(F_k)_{k=0,1,2,...}$ of number-theoretic functions. The functions $F_k : \mathbb{N} \rightarrow \mathbb{N}$ are defined by induction over $k \in \mathbb{N}$:

$$F_0(n) \stackrel{\text{def}}{=} n+1, \qquad F_{k+1}(n) \stackrel{\text{def}}{=} F_k^{n+1}(n) = \overbrace{F_k(F_k(\ldots F_k}^{n+1 \text{ times}}(n)\ldots)). \qquad (D)$$

This induces $F_1(n) = 2n-1$ and $F_2(n) = (n+1)2^{n+1} - 1$, hence F_2 is not polynomial. Writing down an expression for $F_3(n)$ needs a tower of n exponents and F_3 is non-elementary. Note that, for all k and n, $F_k(n+1) > F_k(n)$ and that F_{k+1} dominates F_k.

Each F_k is primitive-recursive. A classic result is that every primitive-recursive function $f : \mathbb{N} \to \mathbb{N}$ is eventually dominated by some F_k.

It is possible to define a variant of Ackermann's function by a diagonalisation process: $Ack(n) \stackrel{\text{def}}{=} F_n(n)$. The *Ack* function is recursive but it eventually dominates any F_k, so it is not primitive-recursive.

A tail-recursive definition. The functions $(F_k)_{k\in\mathbb{N}}$ can be defined in a convenient tail-recursive way via the introduction of a generalized, so-called "*vectorial*", function F with two arguments. Formally, for a vector $\boldsymbol{a} = (a_m, \ldots, a_0) \in \mathbb{N}^{m+1}$, we define

$$F(\boldsymbol{a}; n) = F(a_m, \ldots, a_0; n) \stackrel{\text{def}}{=} F_m^{a_m}(\ldots F_1^{a_1}(F_0^{a_0}(n))). \tag{V}$$

Hence $Ack(m)$ is $F(1, \mathbf{0}; m)$, i.e., $F(1, 0, \ldots, 0; n)$ with m zeroes, and (D) can be reformulated in vectorial form:

$$F(\mathbf{0}; n) = F(0, \ldots, 0; n) = n, \tag{D$_0$}$$

$$F(a_m, \ldots, a_0 + 1; n) = F(a_m, \ldots, a_0; n+1), \tag{D$_1$}$$

$$F(a_m, \ldots, a_k + 1, \overbrace{0, \ldots, 0}^{k>0 \text{ zeroes}}; n) = F(a_m, \ldots, a_k, n+1, \overbrace{0, \ldots, 0}^{k-1 \text{ zeroes}}; n). \tag{D$_2$}$$

Fact 3.1 (Monotonicity) *If $\boldsymbol{a} \leq \boldsymbol{a}'$ and $n \leq n'$ then $F(\boldsymbol{a}; n) \leq F(\boldsymbol{a}'; n')$.*

Reading (D$_{0-2}$) as left-to-right rewrite rules turns them into a functional program for evaluating F: Write $\langle \boldsymbol{a}; n\rangle \xrightarrow{D} \langle \boldsymbol{a}'; n'\rangle$ when (D$_1$) or (D$_2$) transforms the term $F(\boldsymbol{a}; n)$ into $F(\boldsymbol{a}'; n')$. Clearly, $\langle \boldsymbol{a}; n\rangle \xrightarrow{D} \langle \boldsymbol{a}'; n'\rangle$ implies $F(\boldsymbol{a}; n) = F(\boldsymbol{a}'; n')$.

Now, $\xrightarrow{D}$ terminates since $\langle \boldsymbol{a}; n\rangle \xrightarrow{D} \langle \boldsymbol{a}'; n'\rangle$ implies $\boldsymbol{a} >_{\text{lexico}} \boldsymbol{a}'$ (recall that the lexicographical ordering is a linear extension of $\leq$, hence a well-ordering of $\mathbb{N}^{m+1}$). Furthermore, if $\boldsymbol{a} \neq \mathbf{0}$, one of the rules among (D$_1$) and (D$_2$) can be applied to $\langle \boldsymbol{a}; n\rangle$. Hence for all $\boldsymbol{a}$ and n there exists some n' such that $\langle \boldsymbol{a}; n\rangle \xrightarrow{D}{}^{*} \langle \mathbf{0}; n'\rangle$, and then $n' = F(\boldsymbol{a}; n)$ since $F(\boldsymbol{a}; n)$ and $F(\mathbf{0}; n')$, i.e., n', must coincide. (The reverse relation $\xrightarrow{D}{}^{-1}$ terminates too since, in a step $\langle \boldsymbol{a}'; n'\rangle \xrightarrow{D}{}^{-1} \langle \boldsymbol{a}; n\rangle$, either n' is decreased, or it stays constant and the number of zeroes in $\boldsymbol{a}'$ is increased.)

A counter machine evaluating F vectorially. Being tail-recursive, the vectorial F can be evaluated via a simple while-loop that implements the $\xrightarrow{D}$ rewriting. Fix a level $m \in \mathbb{N}$: we need $m+2$ counters, one for the n argument, and $m+1$ for the $\boldsymbol{a} \in \mathbb{N}^{m+1}$ argument.

We define a counter machine $M_{evalF}(m) = (Loc_{evalF}, C, \Delta_{evalF})$, or M_{evalF} for short, with $C = \{\mathtt{a}_0, \mathtt{a}_1, ..., \mathtt{a}_m, \mathtt{n}\}$. Its rules are defined pictorially in Fig. 1: they implement $\xrightarrow{D}$ as a loop around a central location ℓ_{evalF}, as captured by the following lemma.

Lemma 3.2 (Behavior of M_{evalF}). *For all $\boldsymbol{a}, \boldsymbol{a}' \in \mathbb{N}^{m+1}$ and $n, n' \in \mathbb{N}$:*
a. If $\langle \boldsymbol{a}; n\rangle \xrightarrow{D} \langle \boldsymbol{a}'; n'\rangle$ then $M_{evalF} \vdash (\ell_{evalF}, \boldsymbol{a}, n) \to^*_{std} (\ell_{evalF}, \boldsymbol{a}', n')$.
b. If $M_{evalF} \vdash (\ell_{evalF}, \boldsymbol{a}, n) \to^*_{std} (\ell_{evalF}, \boldsymbol{a}', n')$ then $F(\boldsymbol{a}; n) = F(\boldsymbol{a}'; n')$.
c. If $M_{evalF} \vdash (\ell_{evalF}, \boldsymbol{a}, n) \to^*_{lossy} (\ell_{evalF}, \boldsymbol{a}', n')$ then $F(\boldsymbol{a}; n) \geq F(\boldsymbol{a}'; n')$.

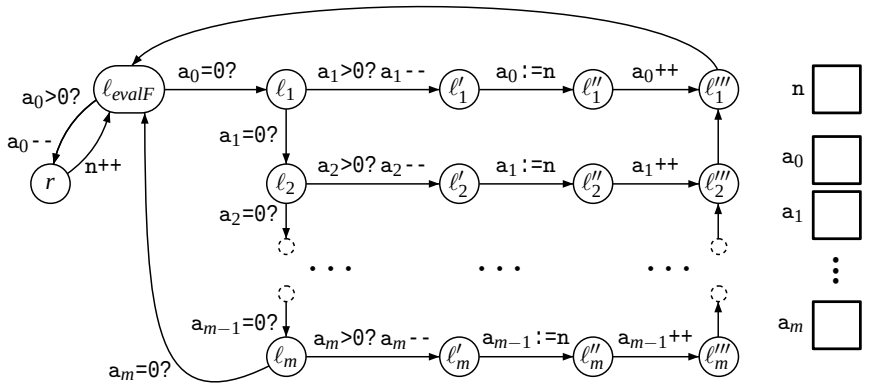

Fig. 1. $M_{evalF}(m)$, a counter machine evaluating F vectorially on $\mathbb{N}^{m+1}$

A counter machine inverting F. The rules (D_0–D_2) can also be used from right to left. Used this way, they *invert* F. This is implemented by another counter machine, $M_{backF}(m) = (Loc_{backF}, C, \Delta_{backF})$, or M_{backF} for short, defined pictorially in Fig. 2.

M_{backF} implements $\xrightarrow{D}{}^{-1}$ as a loop around a central location ℓ_{backF}, as captured by Lemma 3.3. Note that M_{backF} may deadlock if it makes the wrong guess as whether $\mathtt{a}_i$ contains $n+1$, but this is not a problem with the construction.

Lemma 3.3 (Behavior of M_{backF}). *For all $\boldsymbol{a}, \boldsymbol{a}' \in \mathbb{N}^{m+1}$ and $n, n' \in \mathbb{N}$:*
a. If $\langle \boldsymbol{a}; n\rangle \xrightarrow{D} \langle \boldsymbol{a}'; n'\rangle$ then $M_{backF} \vdash (\ell_{backF}, \boldsymbol{a}', n') \to^*_{std} (\ell_{backF}, \boldsymbol{a}, n)$.
b. If $M_{backF} \vdash (\ell_{backF}, \boldsymbol{a}, n) \to^*_{std} (\ell_{backF}, \boldsymbol{a}', n')$ then $F(\boldsymbol{a}; n) = F(\boldsymbol{a}'; n')$.
c. If $M_{backF} \vdash (\ell_{backF}, \boldsymbol{a}, n) \to^*_{lossy} (\ell_{backF}, \boldsymbol{a}', n')$ then $F(\boldsymbol{a}; n) \geq F(\boldsymbol{a}'; n')$.

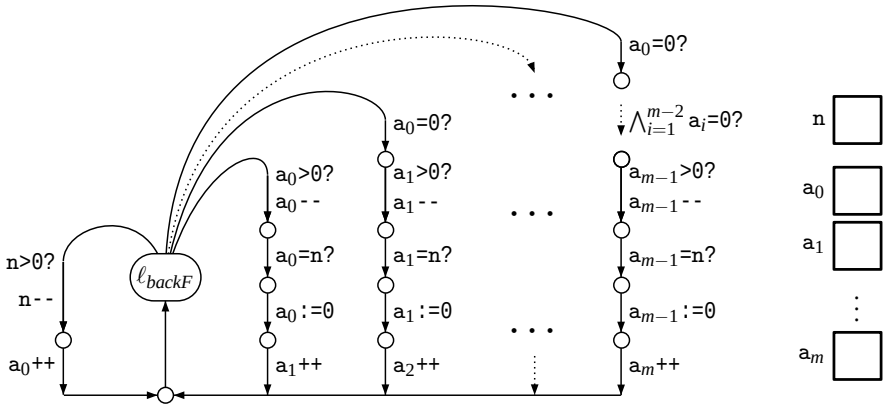

Fig. 2. $M_{backF}(m)$, a counter machine inverting F vectorially on $\mathbb{N}^{m+1}$

4 Minsky machines on a budget

With a Minsky machine $M = (Loc, C, \Delta)$ we associate a Minsky machine $M^{\text{on\_budget}} = (Loc_b, C_b, \Delta_b)$, called M^b for short. (Note that we are only considering Minsky machines here, and not the extended counter machines from earlier sections.)

$M^{\text{on\_budget}}$ is obtained by adding to M an extra "budget" counter B and by adapting the rules of Δ so that any increment (resp. decrement) in the original counters is balanced by a corresponding decrement (resp. increment) on the new counter B, so that the sum of the counters remains constant. This is a classic idea in Petri nets. The construction is described on a schematic example (Fig. 3) that is clearer than a formal definition. Observe that extra intermediary locations (in gray) are used, and that a rule in M that increments some c_i will be forbidden in M^b when the budget is exhausted.

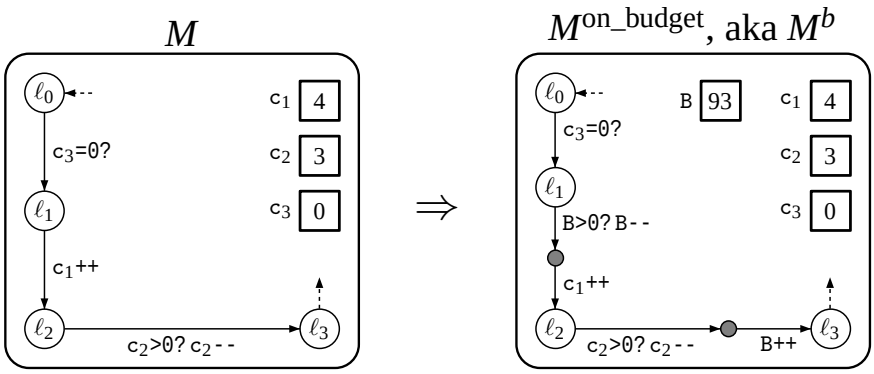

Fig. 3. From M to M^b (schematically)

We now collect the properties of this construction that will be used later. The fact that M^b faithfully simulates M is stated in Lemmas 4.2 and 4.3. There and at other places, the restriction to "$\ell, \ell' \in Loc$" ensures that we only relate behavior anchored at the original locations in M (locations that also exist in M^b) and not at one of the new intermediary locations introduced in M^b.

First, the sum of the counters in M^b is a numerical invariant (that is only temporarily disrupted while in the new intermediary locations).

Lemma 4.1. *If* $M^b \vdash (\ell, B, \boldsymbol{a}) \to^*_{std} (\ell', B', \boldsymbol{a}')$ *and* $\ell, \ell' \in Loc$, *then* $B + |\boldsymbol{a}| = B' + |\boldsymbol{a}'|$.

Observe that M^b can only do what M would do:

Lemma 4.2. *If* $M^b \vdash (\ell, B, \boldsymbol{a}) \to^*_{std} (\ell', B', \boldsymbol{a}')$ *and* $\ell, \ell' \in Loc$ *then* $M \vdash (\ell, \boldsymbol{a}) \to^*_{std} (\ell', \boldsymbol{a}')$.

Reciprocally, everything done by M can be mirrored by M^b *provided that a large enough budget is allowed*. More precisely:

Lemma 4.3. *If* $M \vdash (\ell, \boldsymbol{a}) \to^*_{std} (\ell', \boldsymbol{a}')$ *is an* N*-bounded run of* M, *then* M^b *has an* N*-bounded run* $M^b \vdash (\ell, B, \boldsymbol{a}) \to^*_{std} (\ell', B', \boldsymbol{a}')$ *for* $B \stackrel{def}{=} N - |\boldsymbol{a}|$ *and* $B' \stackrel{def}{=} N - |\boldsymbol{a}'|$.

Now, the point of the construction is that M^b can distinguish between lossy and non-lossy runs in ways that M cannot. More precisely:

Lemma 4.4. *Let $M^b \vdash (\ell, B, \boldsymbol{a}) \to^*_{lossy} (\ell', B', \boldsymbol{a}')$ with $\ell, \ell' \in Loc$. Then $M^b \vdash (\ell, B, \boldsymbol{a}) \to^*_{std} (\ell', B', \boldsymbol{a}')$ if, and only if, $B + |\boldsymbol{a}| = B' + |\boldsymbol{a}'|$.*

Proof (Idea). The "($\Leftarrow$)" direction is an immediate consequence of (†).
For the "($\Rightarrow$)" direction, we consider the hypothesized run $M^b \vdash (\ell, B, \boldsymbol{a}) = \sigma_0 \to_{\text{lossy}} \sigma_1 \to_{\text{lossy}} \cdots \to_{\text{lossy}} \sigma_n = (\ell', B', \boldsymbol{a}')$. Coming back to definition ($*$), these lossy steps require, for $i = 1, \ldots, n$, some reliable steps $\theta_{i-1} \to_{\text{std}} \theta'_i$ with $\sigma_{i-1} \geq \theta_{i-1}$ and $\theta'_i \geq \sigma_i$, and hence $|\theta'_i| \geq |\theta_i|$ for $i < n$. Combining with $|\theta_{i-1}| = |\theta'_i|$ (by Lemma 4.1), and $|\sigma_0| = |\sigma_n|$ (from the assumption that $B + |\boldsymbol{a}| = B' + |\boldsymbol{a}'|$), proves that all these configurations have same size. Hence $\theta'_i = \sigma_i = \theta_i$ and the lossy steps are also reliable steps. □

Corollary 4.5. *Assume $M^b \vdash (\ell, B, \mathbf{0}) \to^*_{lossy} (\ell', B', \boldsymbol{a})$ with $\ell, \ell' \in Loc$. Then:*
1. $B \geq B' + |\boldsymbol{a}|$, and
2. $M \vdash (\ell, \mathbf{0}) \to^*_{std} (\ell', \boldsymbol{a})$ if, and only if, $B = B' + |\boldsymbol{a}|$. Furthermore, this reliable run of M is B-bounded.

5 Ackermann-hardness for lossy counter machines

We now collect the ingredients that have been developed in the previous sections.

Let M be a Minsky machine with two fixed "initial" and "final" locations ℓ_{ini} and ℓ_{fin}. With M and a level $m \in \mathbb{N}$ we associate a counter machine $M(m)$ obtained by stringing together $M_{evalF}(m)$, $M^{\text{on\_budget}}$, and $M_{backF}(m)$ and fusing the extra budget counter B from $M^{\text{on\_budget}}$ with the accumulator n of $M_{evalF}(m)$ and $M_{backF}(m)$ (these two share their counters). The construction is depicted in Fig. 4.

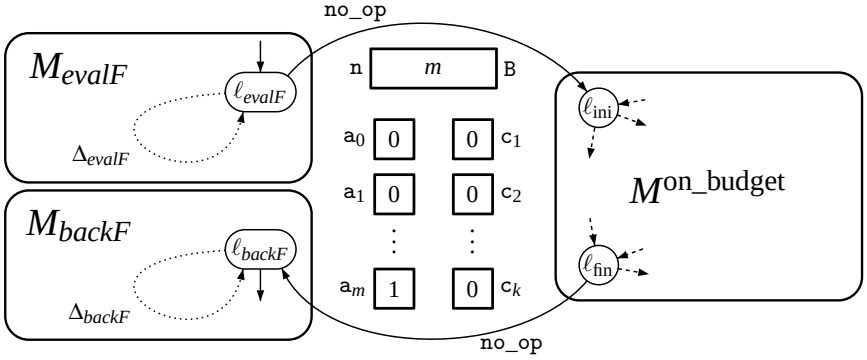

Fig. 4. Constructing $M(m)$ from M^b, M_{evalF} and M_{backF}

Theorem 5.1. *The following are equivalent:*
1. $M(m)$ has a lossy run $(\ell_{evalF}, \mathtt{a}_m : 1, \mathbf{0}, \mathtt{n} : m, \mathbf{0}) \to^*_{lossy} \theta$ for some $\theta \geq (\ell_{backF}, 1, \mathbf{0}, m, \mathbf{0})$.
2. M^{on_budget} has a lossy run $(\ell_{\text{ini}}, \mathtt{B} : Ack(m), \mathbf{0}) \to^*_{lossy} (\ell_{\text{fin}}, Ack(m), \mathbf{0})$.

3. M^{on_budget} *has a reliable run* $(\ell_{\text{ini}}, Ack(m), \mathbf{0}) \to^*_{std} (\ell_{\text{fin}}, Ack(m), \mathbf{0})$.
4. $M(m)$ *has a reliable run* $(\ell_{evalF}, 1, \mathbf{0}, m, \mathbf{0}) \to^*_{std} (\ell_{backF}, 1, \mathbf{0}, m, \mathbf{0})$.
5. M *has a reliable run* $(\ell_{\text{ini}}, \mathbf{0}) \to^*_{std} (\ell_{\text{fin}}, \mathbf{0})$ *that is* $Ack(m)$*-bounded.*

Proof (Sketch).
— For "*1* $\Rightarrow$ *2*", and because coverability implies reachability by $(*)$, we may assume w.l.o.g. that $M(m)$ has a run $(\ell_{evalF}, 1, \mathbf{0}, m, \mathbf{0}) \to^*_{\text{lossy}} (\ell_{backF}, 1, \mathbf{0}, m, \mathbf{0})$. This run must go through $M^{\text{on\_budget}}$ and be in three parts of the following form:

$$(\ell_{evalF}, 1, \mathbf{0}, m, \mathbf{0}) \xrightarrow{\Delta_{evalF}}{}^*_{\text{lossy}} (\ell_{evalF}, \boldsymbol{a}, \mathtt{n} : x, \mathbf{0}) \qquad \text{(starts in } M_{evalF})$$

$$\to_{\text{lossy}} (\ell_{\text{ini}}, \ldots, B, \mathbf{0}) \xrightarrow{\Delta_b}{}^*_{\text{lossy}} (\ell_{\text{fin}}, \ldots, B', \boldsymbol{c}) \qquad \text{(goes through } M^{\text{on\_budget}})$$

$$\to_{\text{lossy}} (\ell_{backF}, \boldsymbol{a}', x', \ldots) \xrightarrow{\Delta_{backF}}{}^*_{\text{lossy}} (\ell_{backF}, 1, \mathbf{0}, m, \mathbf{0}). \qquad \text{(ends in } M_{backF})$$

The first part yields $F(1, \mathbf{0}; m) \geq F(\boldsymbol{a}; x)$ (by Lemma 3.2.*c*), the third part $F(\boldsymbol{a}'; x') \geq F(1, \mathbf{0}; m)$ (by Lemma 3.3.*c*), and the middle part $B \geq B' + |\boldsymbol{c}|$ (by Coro. 4.5.*1*). Lossiness further implies $x \geq B$, $B' \geq x'$ and $\boldsymbol{a} \geq \boldsymbol{a}'$. Now, the only way to reconcile $F(\boldsymbol{a}; x) \leq F(1, \mathbf{0}; m) = Ack(m) \leq F(\boldsymbol{a}'; x')$, $\boldsymbol{a}' \leq \boldsymbol{a}$, $x' \leq x$, and the monotonicity of F (Fact 3.1) is by concluding $x = B = B' = x' = Ack(m)$ and $\boldsymbol{c} = \mathbf{0}$. Then the middle part of the run witnesses $M^{\text{on\_budget}} \vdash (\ell_{\text{ini}}, Ack(m), \mathbf{0}) \to^*_{\text{lossy}} (\ell_{\text{fin}}, Ack(m), \mathbf{0})$.
— "*2* $\Rightarrow$ *5*" is Coro. 4.5.*2*.
— "*5* $\Rightarrow$ *3*" is given by Lemma 4.3.
— "*3* $\Rightarrow$ *4*" is obtained by stringing together reliable runs of the components, relying on Lemmas 3.2.*a* and 3.3.*a* for the reliable runs of M_{evalF} and M_{backF}.
— Finally "*3* $\Rightarrow$ *2*" and "*4* $\Rightarrow$ *1*" are immediate from (†). □

With Theorem 5.1, we have a proof of the Hardness Theorem for reachability and coverability in lossy counter machines: Recall that, for a Minsky machine M, the existence of a run between two given configurations is undecidable, and the existence of a run bounded by $Ack(m)$ is decidable but not primitive-recursive when m is part of the input. Therefore, Theorem 5.1, and in particular the equivalence between its points 1 and 5, states that our construction reduces a nonprimitive-recursive problem to the reachability problem for lossy counter machines.

6 Handling Reset Petri nets

Reset nets [2, 5] are Petri nets extended with special reset arcs that empty a place when a transition is fired. They can equally be seen as special counter machines, called "reset machines", where actions are restricted to decrements, increments, and resets. This is the view we adopt in this paper. *Note that zero-tests are not allowed in reset machines.*

It is known that termination and coverability are decidable for reset machines while other properties like reachability of a given configuration, finiteness of the reachability set, or recurrent reachability, are undecidable [8, 9].

Our purpose is to prove the Ackermann-hardness of termination and coverability for reset machines. We start with coverability and refer to section 7 for termination.

6.1 $R(M)$: replacing zero-tests with resets

For a counter machine M, we let $R(M)$ be the counter machine obtained by replacing every zero-test instruction `c=0?` with a corresponding reset `c:=0`. Note that $R(M)$ is a reset machine when M is a Minsky machine.

Clearly, the behavior of M and $R(M)$ are related in the following way:

Lemma 6.1.
1. $M \vdash \sigma \to_{std} \sigma'$ *implies* $R(M) \vdash \sigma \to_{std} \sigma'$.
2. $R(M) \vdash \sigma \to_{std} \sigma'$ *implies* $M \vdash \sigma \to_{lossy} \sigma'$.

In other words, the reliable behavior of $R(M)$ contains the reliable behavior of M and is contained in the lossy behavior of M.

We now consider the counter machine $M(m)$ defined in Section 5 and build $R(M(m))$.

Theorem 6.2. *The following are equivalent:*
1. $R(M(m))$ *has a reliable run* $(\ell_{evalF}, \mathtt{a}_m : 1, \mathbf{0}, \mathtt{n} : m, \mathbf{0}) \to^*_{std} (\ell_{backF}, 1, \mathbf{0}, m, \mathbf{0})$.
2. $R(M(m))$ *has a reliable run* $(\ell_{evalF}, 1, \mathbf{0}, m, \mathbf{0}) \to^*_{std} \theta$ *for some* $\theta \geq (\ell_{backF}, 1, \mathbf{0}, m, \mathbf{0})$.
3. M *has a reliable run* $(\ell_{\text{ini}}, \mathbf{0}) \to^*_{std} (\ell_{\text{fin}}, \mathbf{0})$ *that is* $Ack(m)$*-bounded.*

Proof. *1* $\Rightarrow$ *3*: The reliable run in $R(M(m))$ gives a lossy run in $M(m)$ (Lemma 6.1.*2*), and we conclude using "*1*$\Rightarrow$*5*" in Theorem 5.1.
3 $\Rightarrow$ *2*: We obtain a reliable run in $M(m)$ ("*5*$\Rightarrow$*4*" in Theorem 5.1) which gives a reliable run in $R(M(m))$ (Lemma 6.1.*1*) which in particular witnesses coverability.
2 $\Rightarrow$ *1*: The covering run in $R(M(m))$ gives a lossy covering run in $M(m)$ (Lemma 6.1.*2*), hence also a lossy run in $M(m)$ that reaches exactly $(\ell_{backF}, 1, \mathbf{0}, m, \mathbf{0})$ (e.g., by losing whatever is required at the last step). From there we obtain a reliable run in $M(m)$ ("*1*$\Rightarrow$*4*" in Theorem 5.1) and then a reliable run in $R(M(m))$ (Lemma 6.1.*1*). □

We have thus reduced an Ackermann-hard problem (point *3* above) to a coverability question (point *2* above).

This almost proves the Hardness Theorem for coverability in Reset nets, except for one small ingredient: $R(M(m))$ is not a reset machine properly because $M(m)$ is an extended counter machine, not a Minsky machine. I.e., we proved hardness for "extended" reset machines. Before tackling this issue, we want to point out that something as easy as the proof of Theorem 6.2 will prove Ackermann-hardness of reset machines by reusing the hardness of lossy counter machines.

In order to conclude the proof of the Hardness Theorem for Reset nets, we only need to provide versions of M_{evalF} and M_{backF} in the form of Minsky machines (M and M^b already are Minsky machines) and plug these in Figure 4 and Theorem 5.1. This is an easy and unsurprising exercise that we only tackle in the full version of this paper.

7 Hardness for termination

We can prove hardness for termination by a minor adaptation of the proof for coverability. This adaptation, sketched in Fig. 5, is similar to the one used in [18]. It applies to both lossy counter machines and reset machines.

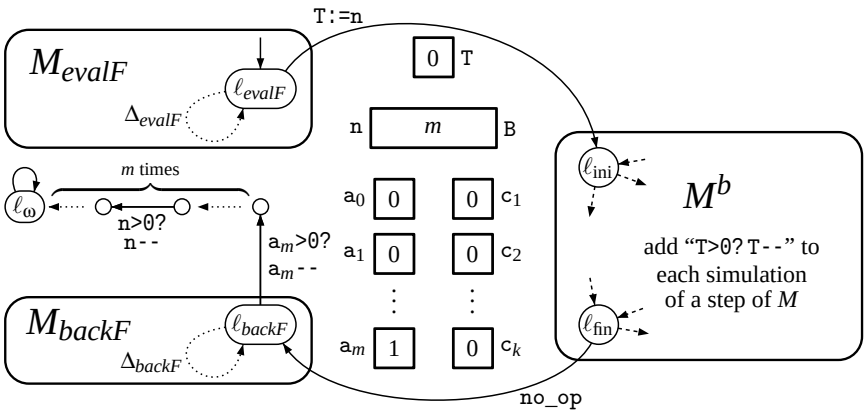

Fig. 5. Hardness for termination: A new version of $M(m)$

Basically, M^b now uses two copies of the initial budget. One copy in B works as before: its purpose is to ensure that *losses will be detected by a budget imbalance* as in Lemma 4.4. The other copy, in a new counter T, is a time limit that is initialized with n and is decremented with every simulated step of M: its purpose is to ensure that the new M^b always terminates. Since M_{evalF} and M_{backF} cannot run forever (because $\xrightarrow{D}$ and $\xrightarrow{D}^{-1}$ terminate, see Section 3), we now have a new $M(m)$ that always terminate when started in ℓ_{evalF} and that satisfies the following variant of Theorems 5.1 and 6.2:

Theorem 7.1. *The following are equivalent:*
1. $M(m)$ *has a lossy run* $(\ell_{evalF}, \mathtt{a}_m : 1, \mathbf{0}, \mathtt{n} : m, \mathbf{0}) \rightarrow^*_{lossy} \theta \geq (\ell_{backF}, 1, \mathbf{0}, m, \mathbf{0})$.
2. $R(M(m))$ *has a reliable run* $(\ell_{evalF}, 1, \mathbf{0}, \mathtt{n} : m, \mathbf{0}) \rightarrow^*_{lossy} \theta \geq (\ell_{backF}, 1, \mathbf{0}, m, \mathbf{0})$.
3. M *has a reliable run* $(\ell_{\text{ini}}, \mathbf{0}) \rightarrow^*_{std} (\ell_{\text{fin}}, \mathbf{0})$ *of length at most* $Ack(m)$.

Finally, we add a series of $m+1$ transitions that leave from ℓ_{backF}, and check that $\sigma_{\text{goal}} \stackrel{\text{def}}{=} (\ell_{backF}, 1, \mathbf{0}, m, \mathbf{0})$ is covered, i.e., that $\mathtt{a}_m$ contains at least 1 and n at least m. If this succeeds, one reaches a new location ℓ_ω, *the only place where infinite looping is allowed unconditionally*. This yields a machine $M(m)$ that has an infinite lossy run if, and only if, it can reach a configuration that covers σ_{goal}, i.e., if, and only if, M has a reliable run of length at most $Ack(m)$, which is an Ackermann-hard problem.

8 Concluding remarks

We proved Ackermann-hardness for lossy counter machines and, with very minor adaptations to the proof, for Reset Petri nets. These results are important in the field of algorithmic verification. Indeed, they have been abundantly cited in recent years even though they were only claimed in the introduction of [18]. The proof we present has several simplifications over the one that was given in [18] for channel systems instead of counter machines. We hope that these improvements will facilitate the wider dissemination of these results.

Acknowledgements. We thank Pierre Chambart and Sylvain Schmitz who greatly helped by proof-reading this paper at various stages.

References

1. R. Amadio and Ch. Meyssonnier. On decidability of the control reachability problem in the asynchronous π-calculus. *Nordic Journal of Computing*, 9(2):70–101, 2002.
2. T. Araki and T. Kasami. Some decision problems related to the reachability problem for Petri nets. *Theoretical Computer Science*, 3(1):85–104, 1977.
3. P. Chambart and Ph. Schnoebelen. The ordinal recursive complexity of lossy channel systems. In *Proc. LICS 2008*, pages 205–216. IEEE Comp. Soc. Press, 2008.
4. P. Chambart and Ph. Schnoebelen. Pumping and counting on the Regular Post Embedding Problem. In *Proc. ICALP 2010*, Lect. Notes Comp. Sci. Springer, 2010.
5. G. Ciardo. Petri nets with marking-dependent arc cardinality: Properties and analysis. In *Proc. ICATPN '94*, volume 815 of *Lect. Notes Comp. Sci.*, pages 179–198. Springer, 1994.
6. S. Demri. Linear-time temporal logics with Presburger constraints: An overview. *J. Applied Non-Classical Logics*, 16(3-4):311–347, 2006.
7. S. Demri and R. Lazić. LTL with the freeze quantifier and register automata. In *Proc. LICS 2006*, pages 17–26. IEEE Comp. Soc. Press, 2006.
8. C. Dufourd, A. Finkel, and Ph. Schnoebelen. Reset nets between decidability and undecidability. In *Proc. ICALP '98*, volume 1443 of *Lect. Notes Comp. Sci.*, pages 103–115. Springer, 1998.
9. C. Dufourd, P. Jančar, and Ph. Schnoebelen. Boundedness of Reset P/T nets. In *Proc. ICALP '99*, volume 1644 of *Lect. Notes Comp. Sci.*, pages 301–310. Springer, 1999.
10. M. V. Fairtlough and S. S. Wainer. Hierarchies of provably recursive functions. In *Handbook of Proof Theory*, volume 137 of *Studies in Logic*, chapter 3, pages 149–207. North-Holland, 1998.
11. D. Figueira, S. Figueira, S. Schmitz, and Ph. Schnoebelen. Ackermann and primitive-recursive upper bounds with Dickson's lemma. In preparation, 2010.
12. D. Figueira and L. Segoufin. Future-looking logics on data words and trees. In *Proc. MFCS 2009*, volume 5734 of *Lect. Notes Comp. Sci.*, pages 331–343. Springer, 2009.
13. A. Finkel, J.-F. Raskin, M. Samuelides, and L. Van Begin. Monotonic extensions of Petri nets: Forward and backward search revisited. In *Proc. INFINITY 2002*, volume 68(6) of *Electronic Notes in Theoretical Computer Science*, pages 121–144, 2003.
14. M. Jurdziński and R. Lazić. Alternation-free modal mu-calculus for data trees. In *Proc. LICS 2007*, pages 131–140. IEEE Comp. Soc. Press, 2007.
15. T. Jurdziński. Leftist grammars are nonprimitive recursive. In *Proc. ICALP 2008*, volume 5126 of *Lect. Notes Comp. Sci.*, pages 51–62. Springer, 2008.
16. R. Mayr. Undecidable problems in unreliable computations. *Theoretical Computer Science*, 297(1–3):337–354, 2003.
17. K. McAloon. Petri nets and large finite sets. *Theoretical Computer Science*, 32(1–2):173–183, 1984.
18. Ph. Schnoebelen. Verifying lossy channel systems has nonprimitive recursive complexity. *Information Processing Letters*, 83(5):251–261, 2002.
19. Ph. Schnoebelen. Lossy counter machines decidability cheat sheet. In *Proc. RP 2010*, volume 6227 of *Lect. Notes Comp. Sci.* Springer, 2010.
20. T. Tan. On pebble automata for data languages with decidable emptiness problem. In *Proc. MFCS 2009*, volume 5734 of *Lect. Notes Comp. Sci.*, pages 712–723. Springer, 2009.