Dense-choice Counter Machines revisited ☆

Florent Bouchy [a], Alain Finkel [a,*], Pierluigi San Pietro [b]

[a] *LSV, ENS Cachan, CNRS, Cachan, France*
[b] *Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Italy*

Received 25 October 2012
Received in revised form 22 December 2013
Accepted 24 April 2014
Communicated by O.H. Ibarra

1. Introduction

Discrete (i.e. integer-valued) Counter Machines have been well-studied and still receive a lot of attention. We can mention Minsky Machines [2], different kinds of counter systems (e.g., [3,4], which are Minsky Machines using affine functions instead of increment/decrements and zero-tests, or [5,6]), Petri nets (or equivalently, VASS) and their many extensions.

There are also extensions of discrete counter systems to real-valued systems, called *hybrid* systems, such as linear hybrid automata, real-counter systems [7], or dense counter systems [8]. Another subclass of hybrid systems is the well-known decidable model of Timed Automata [9], which has been linked to special classes of counter systems in [5] and [6]. Recently, some connections between Timed Automata and timed Petri nets have been made [10,11]. An extension of counter systems to timed counter systems has been defined and studied in [12].

Reachability is already undecidable in linear hybrid automata [13], as well as in Timed Automata extended with only one stopwatch. Other subclasses of hybrid systems, like hybrid Petri nets (which includes stochastic, continuous, differential, and timed Petri nets) are *dense*, i.e. real-valued, extensions of Petri nets, but they have not the same semantics and their comparison is not always easy or feasible (see [14] for a recent survey).

From our point of view, the natural extension of (discrete) counter systems to dense counter systems is quite recent; to the best of our knowledge, the first paper which introduces Dense Counter Machines (DCM) as a natural generalization of Counter Machines (CM) is [8]. Their Dense Counter Machine allows incrementing/decrementing each counter by a real value δ chosen non-deterministically between 0 and 1. The motivation of this extension is to model hybrid systems where a non-deterministic choice can be made (see for example the argumentation about the dense producer/consumer in [8], which

☆ This work presents an extended version of [1] and has been partly supported by the Agence Nationale de la Recherche through grant "REACHARD-ANR-11-BS02-001".

* Corresponding author.
E-mail addresses: florent.bouchy@gmail.com (F. Bouchy), finkel@lsv.ens-cachan.fr (A. Finkel), pierluigi.sanpietro@polimi.it (P. San Pietro).

neither Timed Automata nor hybrid automata can model in an easy way). However, what can we earn from extending CM (which have the total expression power of computability) into DCM? Non-trivial problems will remain, of course, undecidable. The direction followed by [8] is to find subclasses of DCM for which the binary reachability is still computable, such as reversal-bounded DCM.

Our contributions. We first give a general definition of Counter Systems containing all the variations of the nature of counters, such as discrete, dense-choice, purely dense-choice, etc. We then revisit the definition of "Dense Counter Machines" [8] into *Dense-choice Counter Machines* (shortly, also, *DCM*) so that it is now simpler, more precise and formal, and also more clearly understandable as a natural extension of Minsky Machines. Shortly, a DCM is a finite-state machine augmented with dense-choice counters, which can assume only non-negative real values. At each step, every dense-choice counter can be incremented/decremented by 0, 1, or by a non-deterministically-chosen δ, $0 < \delta < 1$ (which may be different at each step). We assume w.l.o.g. that for a given step, the *same* δ is used for all the counters. This δ increment/decrement is the essential difference between dense-choice and discrete counters. A DCM can also test a counter x_i against 0 (either $x_i = 0$ or $x_i > 0$).

Since dense-choice counters are (trivially) more general than discrete counters, we also study the model of *purely-DCM*, i.e. DCM in which counters lose the ability to increment/decrement by 1. We show that the restriction to *bounded* purely-DCM (i.e., there exists a constant bound b such that each counter is bounded by b) still produces an undecidable control-state reachability problem (even with four 1-bounded purely dense-choice counters).

We then consider an effective (i.e. whose binary reachability is computable) class of DCM: reversal-bounded DCM [8]. In order to model hybrid systems more easily, we wish to introduce the ability for a counter to be tested against an integer k (instead of 0): this is an easy, common extension for Minsky Machines and for Petri nets, but it produces new technical problems for reversal-bounded DCM. One of the reasons is that the usual simulation of a k-test (i.e., several decrements and 0-tests, followed by increments restoring the original counter value), does not preserve reversal-boundedness. We actually show that reversal-bounded DCM with k-tests are equivalent to reversal-bounded DCM. This allows us to obtain as a corollary that the reachability relation of a DCM with one free counter and a finite number of reversal-bounded k-testable counters is still effectively definable by a so-called *mixed formula*, which is a formula of a decidable logic equivalent to the first-order additive mixed theory of reals and integers, $\mathrm{FO}(\mathbb{R}, \mathbb{Z}, +, <)$. This extends a previous result of [8].

We give a logical characterization in $\mathrm{FO}(\mathbb{R}, \mathbb{Z}, +, <)$ of the sets of configurations reachable by reversal-bounded DCM, and we prove that any mixed formula is the reachability relation of a reversal-bounded DCM. This completes the initial result stating that the reachability relation of a reversal-bounded DCM is definable by a mixed formula.

Finally, we relate DCM to other models of systems that are more common in verification. In particular, we show that DCM are more expressive than one of the most powerful models of counter automata, and also than the standard timed automata model.

Outline. This paper is divided into four main sections. Section 2 introduces a new formal definition of DCM, and presents different classes of DCM along with some of their properties. In Section 3, known and new decidability results for reachability problems are presented for several classes of DCM. In Section 4, the main theorem gives a full characterization of reversal-bounded DCM by mixed formulae. Finally, Section 5 compares DCM to other common models of systems.

2. Dense-choice Counter Machines

Notations. We use $\mathbb{R}$ to denote the set of real numbers, $\mathbb{R}_+$ the set of non-negative real numbers, $\mathbb{Q}_+$ the set of non-negative rational numbers, $\mathbb{Z}$ the set of integers, and $\mathbb{N}$ the naturals. Capital letters (e.g. X) denote sets, and small letters (e.g. x) denote elements of sets. Bold-faced symbols (e.g. $\mathbf{x}$) denote vectors, and subscripted symbols (e.g. x_i) denote components of vectors. Sometimes, for the sake of readability, we use x instead of x_i (without ambiguity). Throughout this paper, $n \in \mathbb{N}$ is the number of counters.

2.1. Extending Minsky Machines

In this section, we motivate the use of Dense-choice Counter Machines, by arguing about possible ways to extend Minsky Machines [2]. Minsky Machines are indeed the most elementary definition of Counter Systems that we will consider here, and probably the most known. A Minsky Machine has a finite set of control states, and operates transitions between them by executing instructions on a finite set of integer-valued variables (the counters). Its possible instructions are (1) increment a counter value by 1, (2) test if a counter value is 0, and (3) if a counter value is greater than 0 then decrement it by 1.

Let $\mathscr{L}$ be a given logic, such as the Presburger logic $\mathrm{FO}(\mathbb{N}, +, =)$, the mixed linear arithmetic $\mathrm{FO}(\mathbb{R}, \mathbb{Z}, +, <)$, etc. A formula $\mathcal{F}(\mathbf{x}, \mathbf{x}')$ of $\mathscr{L}$, with $2n$ free variables, is interpreted as the transformation of counter values $\mathbf{x}$ into $\mathbf{x}'$: it defines the counter values *before* and *after* the firing of a transition labelled by the binary relation $\mathcal{F}(\mathbf{x}, \mathbf{x}')$. Throughout this paper, we will use several different classes of counter machines, each one based on the generic Definition 2.1. They all use a finite labelling alphabet $\Sigma \subseteq \mathscr{L}$ defining instructions on a vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$. The way the alphabet Σ is defined is what makes the difference between various Counter System classes.

Definition 2.1. A *Counter System* (CS for short) is a tuple $\mathscr{M} = \langle S, T \rangle$ such that S is a finite set of control states, and $T \subseteq S \times \Sigma \times S$ is a finite set of transitions.

A Minsky Machine is a CS in which formulae of Σ are of the following form, where x is a component of $\mathbf{x}$: $(x' = x + 1)$, i.e., counter x is incremented by one, $(x' = x = 0)$, i.e., counter x is 0 and it is not modified, $(x > 0 \wedge x' = x - 1)$, i.e., $x > 0$ is decremented by one, or *true*, i.e., counter values remain unchanged. Although the reachability problem is undecidable for Minsky Machines (with two or more counters), we would like to extend them for two reasons. First, if a Minsky Machine is reversal-bounded, then its reachability relation is computable; thus, we might be able to use a more expressive model than reversal-bounded Minsky Machines which still remains decidable. This first point will be detailed in Sections 2.2 and 2.3. Second, Minsky Machines are very basic and not very practical to be used for modelling or expressing high-level properties. For that matter, we add the possibility to use real-valued counters, and to choose non-deterministically the value of an increment/decrement for each transition. In the remainder of this section, we discuss and compare the two latter extensions.

In order to get real-valued counters, we first define *Dense* Minsky Machines, which are CS whose Σ is composed of formulae of the form $(x' = x + r)$, $(x' = x = 0)$, $((x - r > 0 \vee x - r = 0) \wedge x' = x - r)$, or *true*, with a given finite set of values[1] $r \in \mathbb{Q}_+$. Like in Minsky Machines, the semantics is as expected and the initial counter values are always 0. This first extension as Dense Minsky Machines is not really more powerful, since it can be simulated by a Minsky Machine:

Proposition 2.2. *Minsky Machines and Dense Minsky Machines are bisimilar.*

Proof. We show that each kind of machine may simulate the other. One way is trivial, by taking $r = 1$. The other way is a little more elaborate, but remains easy. We just have to simulate every Dense Minsky Machine instruction with a Minsky machine. There are four instructions, and two of them are obviously the same: *true* and $x' = x = 0$. For the other two instructions, $x' = x + r$ and $(x - r > 0 \vee x - r = 0) \wedge x' = x - r$, we just have to encode r by an integer. Each increment/decrement $r \in \mathbb{Q}_+$ can be written as $\frac{p}{q}$, with $p, q \in \mathbb{N}$. Then, since we know all the possible r in advance, we can compute for each r a $q' \in \mathbb{N}$ such that $r = \frac{pq'}{q_{lcm}}$, where q_{lcm} is the least common multiple of all q. Thus, each r can be represented by a non-negative integer $r' = pq'$, and the new counter values will all be multiplied by the same factor q_{lcm}. Using this simple encoding, we can simulate an instruction $x' = x + r$ by a sequence of r' instructions $x' = x + 1$. Likewise, $(x - r > 0 \vee x - r > 0) \wedge x' = x - r$ can be simulated by a sequence of r' instructions $x > 0 \wedge x' = x - 1$. □

Another way to extend Minsky Machines is to allow, on each transition, a non-deterministic choice of the increment/decrement. We call this extension a *Dense-choice* Minsky Machine, which is a CS whose Σ contains formulae f of the form $(x' = x + 1)$, $(x' = x = 0)$, $(x > 0 \wedge x' = x - 1)$, $(x' = x + \Delta)$, $((x - \Delta > 0 \vee x - \Delta = 0) \wedge x' = x - \Delta)$, or *true*, where Δ symbolizes a non-deterministically-chosen value $\delta \in \mathbb{R}_+$. The choice is made each time a transition is fired, so that two consecutive transitions labelled by $x' = x + \Delta$ may have different values for Δ: the choice is non-deterministic, and we have no knowledge of the chosen value (although it could be checked against other counter values). Given a transition label $\mathbf{f}$, we write $\mathbf{f}(\mathbf{x}, \mathbf{x}')$ to denote the application of formulas f in $\mathbf{f}$ to counters $\mathbf{x}$, resulting in $\mathbf{x}'$. Given $\mathbf{f} = (f_i)_{i \in [0,j]}$, with $j \geq 0$, each f_i is either a dense-choice increment/decrement (i.e., of the form $(x' = x + \Delta)$ or $((x - \Delta > 0 \vee x - \Delta = 0) \wedge x' = x - \Delta)$) or not (i.e., discrete increment, decrement, zero-test or *true*); then, we write $\mathbf{f}_\Delta = \{f_i \in \mathbf{f} \mid f_i \text{ is a dense-choice increment or decrement}\}$. The semantics of a Dense-choice Minsky Machine M is given by a transition system $\mathrm{TS}(M) = (C, \rightarrow)$, where $C = S \times \mathbb{R}_+^n$ is the set of configurations and $\rightarrow \subseteq C \times \Sigma \times C$ is the set of transitions defined by: $\langle s, \mathbf{x} \rangle \xrightarrow{\mathbf{f}} \langle s', \mathbf{x}' \rangle$ if and only if $(s, \mathbf{f}, s') \in T$ and $\exists \delta \in \mathbb{R}_+$ such that $\mathbf{f}(\mathbf{x}, \mathbf{x}')[\Delta \leftarrow \delta]$ holds. A *run* of a Dense-choice Minsky Machine is a sequence $\rho = \langle s^0, \mathbf{x}^0 \rangle \xrightarrow{\mathbf{f}^1} \langle s^1, \mathbf{x}^1 \rangle \xrightarrow{\mathbf{f}^2} \ldots \xrightarrow{\mathbf{f}^h} \langle s^h, \mathbf{x}^h \rangle$ of finite length $h \geq 0$, sometimes written $(c^0, \mathbf{f}^1, c^1, \mathbf{f}^2 \ldots \mathbf{f}^h, c^h)$. The set of all runs of a Dense-choice Minsky Machine M is denoted $\mathsf{Runs}(M)$.

We show that the δ value can be chosen in the open interval $]0, 1[$ rather than in $\mathbb{R}_+$. First, we need additional definitions. Let $M = \langle S, T \rangle$ and $M' = \langle S', T' \rangle$ be Dense-choice Minsky Machines. A run $\rho' \in \mathsf{Runs}(M')$ is said to *simulate* a run $\rho \in \mathsf{Runs}(M)$, denoted $\rho \preceq \rho'$, if and only if $\rho = (c^0, \mathbf{f}^1, c^1, \mathbf{f}^2, c^2, \ldots, \mathbf{f}^h, c^h)$ and $\rho' = (c^0, \ldots, \mathbf{f}^1, c^1, \ldots, \mathbf{f}^2, c^2, \ldots, \mathbf{f}^h, c^h)$; in other words, ρ' has to go through the same configurations as ρ but may go through additional configurations in between.

Definition 2.3. Two Dense-choice Minsky Machines M and M' are *equivalent*, denoted $M \equiv M'$, if and only if both:

(1) $\forall \rho \in \mathsf{Runs}(M), \exists \rho' \in \mathsf{Runs}(M')$ such that $\rho \preceq \rho'$;
(2) $\forall \rho' \in \mathsf{Runs}(M'), \exists \rho \in \mathsf{Runs}(M)$ such that $\rho' \preceq \rho$.

Proposition 2.4. *Dense-choice Minsky Machines with $\delta \in \mathbb{R}_+$ are equivalent to Dense-choice Minsky Machines with $\delta \in]0, 1[$.*

Proof. We first show that for any Dense-choice Minsky Machine with $\delta \in \mathbb{R}_+$, we can build an equivalent Dense-choice Minsky Machine with $\delta \in]0, 1[$. Let $M = \langle S, T \rangle$ be a Dense-choice Minsky Machine with $\delta \in \mathbb{R}_+$ and an initial configura-

[1] Note that here, we take these values in $\mathbb{Q}_+$ because the important properties are (1) density and (2) an effective representation of any rational number (this is not the case for reals, in general).

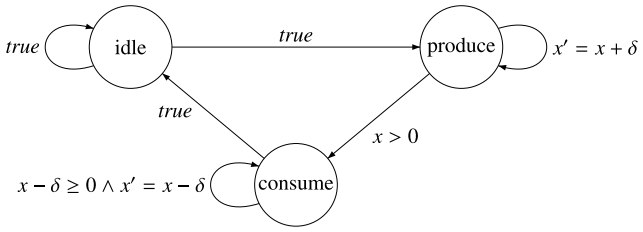


Fig. 1. Example of a producer–consumer system.

tion $\langle s^0, \mathbf{x}^0 \rangle$. We build a Dense-choice Minsky Machine $M = \langle S', T' \rangle$ with $\delta \in]0, 1[$ by replacing each transition t by new transitions going through a fresh state as follows:

$S' = S \cup S_T$, with $S_T = \{s_t \mid t \in T\}$; notice that $S \cap S_T = \emptyset$.
$T' = \{(s^i, true, s_t), (s_t, \mathbf{f}_\Delta, s_t), (s_t, \mathbf{f}, s^j) \mid t = (s^i, \mathbf{f}, s^j) \in T \wedge s_t \in S_T\}$.

We show that runs of M and M' are equivalent by proving that Parts (1) and (2) of Definition 2.3 hold.

Part (1) is shown by induction on the length of runs, measured as the number of configurations. When $|\rho| = 1$ the run is necessarily the initial configuration $\langle s^0, \mathbf{x}^0 \rangle$, which is also a run ρ' of M' such that $\rho \preceq \rho'$. Now, consider a run $\rho = (c^0, \mathbf{f}^1, c^1, \mathbf{f}^2 \dots \mathbf{f}^h, c^h) \in \mathsf{Runs}(M)$ simulated by a run $\rho' \in \mathsf{Runs}(M')$, with $h \geq 1$ and $\forall i \geq 0$, $c^i = \langle s^i, \mathbf{x}^i \rangle$. Let $\gamma = c^h \xrightarrow{\mathbf{f}} c^{h+1}$ be an additional step in TS(M), corresponding to a transition t in M; then a given $\delta \in \mathbb{R}_+$ is chosen when executing this step. In M', the three transitions corresponding to t can yield the following suffix of ρ' (omitting the counter values): $\gamma' = (s^h, true, s_t, (\mathbf{f}_\Delta, s_t)^{2|\delta|}, \mathbf{f}, s^{h+1})$, where $\delta' = \frac{1}{2}$ for $\mathbf{f}_\Delta$ and $\delta' = \delta - |\delta|$ for $\mathbf{f}$. Thus $\rho\gamma \preceq \rho'\gamma'$.

Part (2) is also shown by induction on the length of runs, with the same initial configuration when $|\rho'| = 1$. Now, consider a run $\rho' = (c^0, \mathbf{f}^1, c^1, \mathbf{f}^2 \dots \mathbf{f}^h, c^h) \in \mathsf{Runs}(M')$ simulated by a run $\rho \in \mathsf{Runs}(M)$, with $h \geq 1$ and $\forall i \geq 0$, $c^i = \langle s^i, \mathbf{x}^i \rangle$. Let $\gamma' = c^h \xrightarrow{\mathbf{f}} c^{h+1}$ be an additional step in TS(M'). Then $\rho'\gamma' \preceq \rho\gamma$, with either $\gamma = \varepsilon$ (i.e. $|\gamma| = 0$) if $s^h \notin S_T \vee s^h, s^{h+1} \in S_T$, or $\gamma = \gamma'$ if $s^h \in S_T \wedge s^{h+1} \notin S_T$, where increment $\delta \in \mathbb{R}_+$ can be the same as the $\delta \in]0, 1[$.

In order to complete the proof, just notice that any Dense-choice Minsky Machine, with $\delta \in]0, 1[$, is also a Dense-choice Minsky Machine with $\delta \in \mathbb{R}_+$. □

Therefore, there is no loss in generality in assuming that each increment is in the interval $]0, 1[$ rather than in $\mathbb{R}_+$, at least as long as finite runs are considered. Instead, a bounded increment can give a finer degree of control on counters. In fact, in many physical systems, physical variables are actually bounded (e.g., a water level in a reservoir, which is a non-negative real value that cannot exceed the height of the reservoir). It seems difficult to model or check this kind of behaviour with a CS where increments are unbounded reals.

Finally, we notice that allowing increments in the interval $]0, q[$, with a fixed $q \in \mathbb{N}$, does not give any gain in expressiveness with respect to the case of $q = 1$. For instance, to increment a counter x by any value δ with $0 < \delta < q$, it is enough to apply, in a Dense-choice Minsky Machine with non-deterministic increments in $]0, 1[$, a sequence of exactly q transitions (this is possible, since q is fixed), each of the form $x' = x + \delta$, for $0 < \delta < 1$.

In the next section, we generalize and formalize the definition of Dense-choice Minsky Machine that we just motivated. Before that, we illustrate the use of real-valued counters on a small example.

Example *(Producer–consumer system).* As a simple example of application of a machine with real-valued counters, consider the following version of a traditional producer–consumer system, described in [8]. A system may be in one of three states: *produce*, *consume* or *idle*. When in state *produce*, a resource is created, which may be stored and later used while in state *consume*. The resource is a real number, representing an available amount of a physical quantity, such as fuel or water. Production may be stored, and used up much later (or not used at all). This system may be easily modelled by a finite-state machine with one dense-choice counter, as shown on Fig. 1, where the resource is added when produced or subtracted when consumed.

By using a real-valued counter, there is an underlying assumption that a continuous variable, such as this resource, changes in discrete steps only; however, this is acceptable in many cases where a variable actually changes continuously, since the increments/decrements may be arbitrarily small. Since the counter may never decrease below zero, the specified system implements the physical constraint that consumption must never exceed production. More complex constraints are decidable, for instance if expressed by linear constraints on counter values. An example of a decidable query is whether total production never exceeds twice the consumption.

2.2. Definitions and properties of Dense-choice Counter Machines

Let $\mathbf{x}$ be a vector of n variables, called *dense-choice counters* (or simply *counters* if not specified otherwise). Dense-choice counters were called "dense counters" in [8]. A *counter valuation* is a function assigning a value in $\mathbb{R}_+$ to any component x_i of $\mathbf{x}$. In this paper, we write x_i (or $\mathbf{x}$) to denote both variable(s) and the image of counter valuation(s), since there is no ambiguity and the meaning is obvious. Let $G = \{(x = 0), (x > 0), true\}$ be the set of guards. We say that a counter valuation $\mathbf{x}$ satisfies a guard $\mathbf{g} \in G^n$, when each x_i satisfies guard g_i (in the obvious meaning); for example, if $n = 3$ and $\mathbf{g} = (true, x_2 > 0, x_3 = 0)$, then $(6, 2, 0)$ satisfies $\mathbf{g}$, while $(6, 2, 1)$ does not. Let $A = \{1, \Delta\}$ be the set of actions; intuitively, 1 stands for an integer increment/decrement, and Δ stands for a non-deterministically-chosen real increment/decrement.

Definition 2.5. A *Dense-choice Counter Machine* (shortly, a DCM) with $n > 0$ counters is a tuple $\mathscr{M} = \langle S, T \rangle$ where:

- S is a finite set of control states, with $s_{fin} \in S$ called the *final state* of $\mathscr{M}$;
- $T \subseteq S \times \Sigma \times S$ is a finite set of transitions, with $\Sigma = (G^n \times \mathbb{Z}^n \times A^n)$.

Intuitively, the integer component $\boldsymbol{\lambda} \in \mathbb{Z}^n$ of Σ is a factor determining whether the transition is incrementing or decrementing a counter, and of which value. Meanwhile, the action $\mathbf{a} \in A^n$ determines whether the increment or decrement is a real or integer value. For the sake of clarity, transitions are sometimes written in a more readable form; e.g., $x_i > 0 \wedge x_i := x_i + 3\delta$ means that the guard on counter x_i is $g_i = (x > 0)$, its factor is $\lambda_i = 3$, and its action is $a_i = \Delta$.

Notice that our transitions are equivalent to those of [8,7], in which the authors used the notion of *modes*. The modes `stay`, `unit increment`, `unit decrement`, `fractional increment`, and `fractional decrement` are here respectively represented by the cases $(\lambda_i = 0)$, $(\lambda_i > 0 \wedge a_i = 1)$, $(\lambda_i < 0 \wedge a_i = 1)$, $(\lambda_i > 0 \wedge a_i = \Delta)$, and $(\lambda_i < 0 \wedge a_i = \Delta)$. Also notice that transitions where $\lambda \in \{+1, -1\}^n$ are just a special case, and that they can simulate a linear combination of the form $x'_i = x_i + \sum_{j=1}^{m} \lambda_j \delta_j$, for a given m and a vector of different δ_j values in $]0, 1[$.

Interpreting a DCM is performed as usually in verification: we specify an initial valuation to each counter and an initial control state, and then we let the machine behave non-deterministically. The behaviour of a DCM mainly consists in choosing a transition whose guard $\mathbf{g} \in G^n$ is satisfied by the current counter valuations, and to update these valuations, while, of course, also going to a new control state.

Definition 2.6. The semantics of a DCM $\mathscr{M} = \langle S, T \rangle$ is given by a transition system $\mathrm{TS}(\mathscr{M}) = \langle C, \rightarrow \rangle$ where:

- $C = S \times \mathbb{R}_+^n$ is the set of configurations;
- $\rightarrow \subseteq C \times \Sigma \times C$ is the set of transitions, defined by:
  $\langle s, \mathbf{x} \rangle \xrightarrow{\mathbf{g}, \boldsymbol{\lambda}, \mathbf{a}} \langle s', \mathbf{x}' \rangle$ if and only if $(s, (\mathbf{g}, \boldsymbol{\lambda}, \mathbf{a}), s') \in T$ and $\exists \delta \in \mathbb{R}$ such that:
  $\mathbf{x}$ satisfies $\mathbf{g}$ and $0 < \delta < 1 \wedge \mathbf{x}' = \mathbf{x} + \boldsymbol{\lambda}\mathbf{u}$, with $\mathbf{u} = \mathbf{a}[\Delta \leftarrow \delta]$.

For a DCM $\mathscr{M} = \langle S, T \rangle$ and its transition system $\mathrm{TS}(\mathscr{M}) = \langle C, \rightarrow \rangle$, the reachability relation $\rightsquigarrow_{\mathscr{M}}$ is the reflexive and transitive closure $\rightarrow^*$; when the context is clear, we drop the subscript $_{\mathscr{M}}$. A *run* of $\mathscr{M}$ is a sequence $\langle s^0, \mathbf{x}^0 \rangle \rightarrow \langle s^1, \mathbf{x}^1 \rangle \rightarrow \ldots \rightarrow \langle s^h, \mathbf{x}^h \rangle$, of length $h \geq 0$. Because of the inherent non-determinism of a DCM, we are interested only in runs ending in final state $s_{fin} \in S$. The notions of simulation and equivalence are the same as for Dense Minsky Machines defined page 19. Roughly, a DCM $\mathscr{M}$ can be simulated by a DCM $\mathscr{M}'$ iff any run of $\mathscr{M}$ can be replicated in $\mathscr{M}'$, possibly going through additional intermediate configurations and using additional counters in $\mathscr{M}'$.

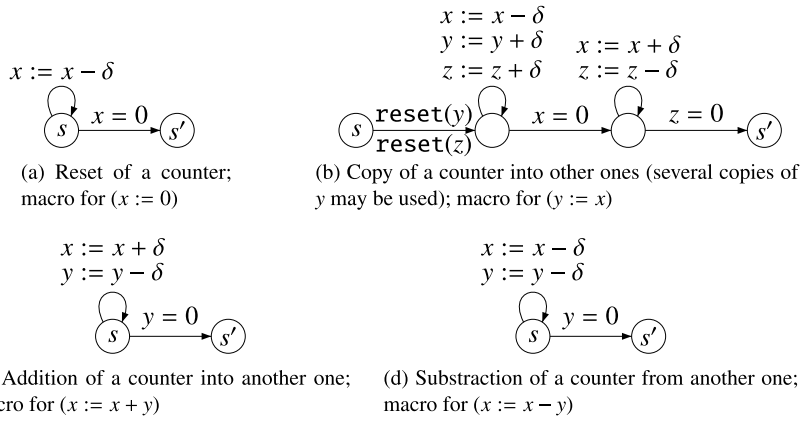
Formally, a run of $\mathscr{M}$ is *accepting* if it is of the form $\langle s, \mathbf{x} \rangle \rightarrow^* \langle s_{fin}, \mathbf{x}' \rangle$, for some $s \in S$ and $\mathbf{x}, \mathbf{x}' \in \mathbb{R}_+^n$; $\mathscr{M}$ is said to *accept* if there exists an accepting run of $\mathscr{M}$. $\mathscr{M}$ is said to *reject* if there is no accepting run of $\mathscr{M}$.

In the proofs and constructions that follow, it is often useful to consider the case of non-accepting runs that cannot continue further. For instance, a run reaching a configuration $\langle s', \mathbf{x}' \rangle$ may not proceed further if there is no outgoing transition from s', or if every outgoing transition from s' has either a guard not satisfied by $\mathbf{x}'$ or a decrement action on a counter $x'_j = 0$. Formally, $\mathscr{M}$ is said to *crash* during a run $\langle s, \mathbf{x} \rangle \rightarrow^* \langle s', \mathbf{x}' \rangle$ if $s' \neq s_{fin}$ and $\neg \exists \langle s'', x'' \rangle \in C$ such that $\langle s', \mathbf{x}' \rangle \rightarrow_{\mathscr{M}} \langle s'', \mathbf{x}'' \rangle$. Hence, a run making $\mathscr{M}$ crash may never be extended to be an accepting run.

The set of all pairs $(\langle s, \mathbf{x} \rangle, \langle s', \mathbf{x}' \rangle) \in C \times C$ such that $\langle s, \mathbf{x} \rangle \rightsquigarrow \langle s', \mathbf{x}' \rangle$ is the *binary reachability relation* of $\mathscr{M}$, sometimes called its *binary reachability* or *reachability relation*. The *binary reachability problem* consists in computing the binary reachability relation of a given DCM, i.e., in deciding whether $\langle s, \mathbf{x} \rangle \rightsquigarrow_{\mathscr{M}} \langle s', \mathbf{x}' \rangle$, given any $\langle s, \mathbf{x} \rangle, \langle s', \mathbf{x}' \rangle \in C$. An easier version is the *control-state reachability problem* (shortly, *state reachability problem*), which consists in deciding whether a given control state is reachable in some accepting run of a given DCM.

The example on Fig. 1 (page 20) is a DCM, since the guard $x - \delta \geq 0$ can be removed (the machine crashes anyway if the guard is not satisfied).

Although a DCM has only a restricted set of possible operations on counters, it can perform various higher-level operations, such as reset, copy, addition, subtraction, comparison, etc., that can be encoded as macros to be used as a shorthand. Some useful encodings are presented here.



(a) Reset of a counter; macro for $(x := 0)$

(b) Copy of a counter into other ones (several copies of y may be used); macro for $(y := x)$

(c) Addition of a counter into another one; macro for $(x := x + y)$

(d) Substraction of a counter from another one; macro for $(x := x - y)$

Fig. 2. Encodings of `reset`, `copy`, `add`, and `minus` operations.

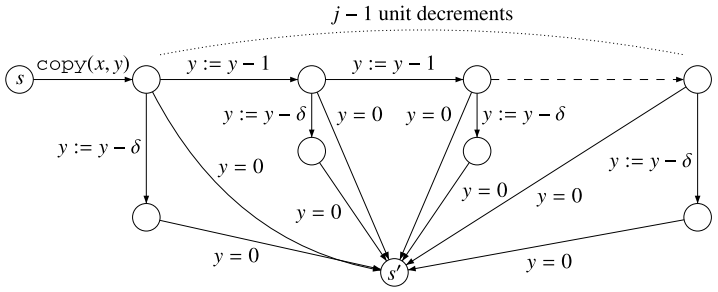


Fig. 3. Elimination of k-tests.

We denote by $s \xrightarrow{\texttt{reset}(x)} s'$, $s \xrightarrow{\texttt{copy}(x,y)} s'$, $s \xrightarrow{\texttt{add}(x,y)} s'$, and $s \xrightarrow{\texttt{minus}(x,y)} s'$ the DCM on Figs. 2a, 2b, 2c, and 2d (respectively).

Let $G'_k = \{(x = j), (x < j), (x > j), true \mid j \in \{0, \ldots, k\}\}$, for a given $k \in \mathbb{N}$. An (extended) DCM whose set of guards is included in G'_k, instead of G, is called a *k-DCM*. The counters of a k-DCM are said to be *k-testable*. The semantics of such extended guards is the obvious one, i.e., comparing a counter against an integer constant. Notice that a DCM is a 0-DCM, and that dense-choice counters are 0-testable if not specified otherwise. Moreover, every k-DCM is, by definition, a $(k+1)$-DCM, for all $k \in \mathbb{N}$. In fact, we prove that every k-DCM can be simulated by a DCM.

Proposition 2.7. *DCM are equivalent to k-DCM, for any given $k \in \mathbb{N}$.*

Proof. There are three different kinds of additional tests in k-DCM: $x < j$, $x > j$, and $x = j$, for any given $j \in \mathbb{N}$. We show how to encode each of them with only $x = 0$ and $x > 0$ tests.

A test $x < j$, with $j > 0$, represented by a transition $s \xrightarrow{x<j} s'$, can be simulated by the encoding depicted on Fig. 3.

Note that to avoid modifying the value of x, x is copied into another counter y by using the encoding of Fig. 2b, then y is decremented of $j - 1$ units, testing at each step whether $y - \delta = 0$ (i.e., whether $0 < y < 1$) or $y = 0$ (in case y was an integer). Since, in any run, y can never go below 0 (otherwise, the machine would crash), then if $x < j$ the machine would decrement y exactly of the integer part of x, and then decrement y once again exactly of the fractional part of x. The encoding of test $x > j$ (resp. $x = j$) is simpler, since the test can be simulated by a sequence of exactly j unit decrements of copy y followed by a test $y > 0$ (resp. $y = 0$).

Finally, to complete the proof, notice that any k-DCM contains a DCM. □

Definition 2.8. Let $\mathscr{M}$ be a DCM. A counter x_i of $\mathscr{M}$ is *purely dense-choice* if and only if $a_i = \Delta$ in every transition (i.e., it is never incremented/decremented by 1). Conversely, a counter x_i is *(purely) discrete* if and only if $a_i = 1$ in every transition (i.e., it is a classical discrete counter). If $\mathscr{M}$ contains only purely dense-choice counters, it is called a *purely-DCM*. If $\mathscr{M}$ contains only discrete counters, it is called a *(discrete) Counter Machine (CM)*, as defined in [15].

2.3. Reversal-bounded DCM

To extend the definition of reversal-boundedness from [15] to DCM, let $\mathscr{M} = \langle S, T \rangle$ be a DCM, $s, s' \in S$, and $r \in \mathbb{N}$. For every i, $1 \leq i \leq n$, a counter x_i is *r-reversal-bounded* on a run from s to s', if, along the transitions of the run, the factors

λ_i switch between positive and negative values at most r times. Counter x_i is *reversal-bounded* (shortly, *r.b.*) if there is an r such that x_i is r-reversal-bounded on every accepting run of $\mathscr{M}$. $\mathscr{M}$ is a *reversal-bounded Dense-choice Counter Machine*, denoted by *r.b. DCM*, if every counter in $\mathscr{M}$ is reversal-bounded.

A counter which is not necessarily reversal-bounded is called a *free* counter.

In this model, one can effectively check whether a counter is r-reversal-bounded, by making the control state check when transitions are incrementing ($\lambda_i > 0$) or decrementing ($\lambda_i < 0$) the counter x_i. Thus, one can use additional control states in order to remember each reversal and crash if the number of reversals exceeds constant bound r.

From [8], we know that the binary reachability of a reversal-bounded DCM with one free counter can be defined in a decidable logic: the logic of *mixed formulae*, which is equivalent to the well-known FO($\mathbb{R}, \mathbb{Z}, +, <$). Since the syntactical details of this logic are not relevant for now, its presentation is postponed to Section 4.1. The above decidability result in [8] can be restated as follows:

Proposition 2.9. *(See [8].) The binary reachability of a reversal-bounded DCM with one free 0-testable counter and a finite number of r.b. 0-testable counters is definable by a mixed formula.*

However, we have extended the guards of DCM to be able to test a counter against any given integer constant: we proved in Proposition 2.7 that this is not more powerful in the general case, but this is far from obvious when we consider r.b. DCM. Indeed, the encoding in the proof of Proposition 2.7 does not preserve reversal-boundedness (a k-test, which may be taken an unbounded number of times during a run, is replaced by reversals). Nevertheless, we now prove Theorem 2.12, which states that this extension is actually not more powerful even in the case of r.b. DCM, provided we can use many more counters. First, we need a technical lemma.

Lemma 2.10. *Let $\mathscr{M} = \langle S, T\rangle$ be a one-reversal-bounded (1-r.b.) k-DCM; let x be a k-testable 1-r.b. counter of $\mathscr{M}$, and j be an integer $0 < j \leq k$. Then there is a k-DCM $\mathscr{M}'$ with the same counters as $\mathscr{M}$ and 12 additional 0-testable r.b. counters, such that $\mathscr{M}'$ simulates $\mathscr{M}$ and there is no test of x against j in $\mathscr{M}'$.*

Proof. Let $\mathscr{M} = \langle S, T\rangle$ be a one-reversal-bounded k-DCM, x be a (k-testable 1-r.b.) counter of $\mathscr{M}$, and j be an integer such that $0 < j \leq k$. We encode tests of x against j in the state control of an associated k-testable machine $\mathscr{M}' = \langle S', T'\rangle$.

Since $\mathscr{M}$ is one-reversal-bounded, every counter of $\mathscr{M}$ is one-reversal-bounded; then there are at most two phases in each run of $\mathscr{M}$: each counter starts at 0, then it may go through an increasing phase, possibly followed by a decreasing one (actually, phases are "non-decreasing" then "non-increasing").

We may assume that state control can memorize the current phase of counter x. Then a configuration of machine $\mathscr{M}'$ will not only contain the configuration $(s, \mathbf{x})$ of $\mathscr{M}$ but also $\uparrow_x$ or $\downarrow_x$ for indicating that counter x is in its increasing phase or in its decreasing phase. One may find this explicit construction in [16].

First, notice that every guard crossed during a run must have been evaluated to true (if false, the corresponding transition is not taken). Hence, certain sequences of guards are simply not possible in a run. For instance, in the *increasing* phase, after crossing a guard of the form $(x = j)$ it is impossible to cross a guard $(x < j)$, while after crossing $(x > j)$ it is impossible to cross $(x < j)$ or $(x = j)$. Therefore, the sequence of guards testing x against j that are crossed during a run may be represented as a string of the form $(x < j)^*(x = j)^*(x > j)^*$ in the increasing phase, followed by $(x > j)^*(x = j)^*(x < j)^*$ in the decreasing phase. Second, notice that in a sequence of guards of the form $(x = j)^+$, only the leftmost guard must be tested, since the following tests whether $x = j$ may be left to the finite state control (by just storing whether x has changed). Third, using transitivity, only a few actual tests should be performed. In the increasing phase, in a sequence of guards $(x < j)$ only the rightmost guard needs to be tested, while in a sequence of guards $(x > j)$ only the leftmost guard needs to be tested. Symmetrically, in the decreasing phase it is enough to test the leftmost $(x > j)$ and the rightmost $(x < j)$.

For example, if the sequence of guards crossed during the increasing phase of a computation is:

$$(x < j)(x < j)(x < j)(x = j)(x = j)(x = j)(x > j)(x > j)(x > j)$$

then the sequence can be replaced by

$$(true)(true)(x < j)(x = j)(true)(true)(true)(x > j)(true)(true).$$

In the decreasing phase, the sequence:

$$(x > j)(x > j)(x > j)(x = j)(x = j)(x = j)(x < j)(x < j)(x < j)$$

can be replaced by

$$(true)(true)(x > j)(x = j)(true)(true)(true)(x < j)(true)(true).$$

Therefore, only at most three tests of x against j are needed in the increasing phase, followed by at most three tests in the decreasing phase, for a total of at most 6 tests.

The decision whether a test of x against j may be skipped (i.e., replaced by $true$), or actually executed, can be easily encoded in the finite-state control of an r.b. machine $\mathscr{M}''$ simulating $\mathscr{M}$. Thus, $\mathscr{M}''$ may be built to simulate $\mathscr{M}$ so that, in every run, $\mathscr{M}''$ makes at most 6 tests of x against j.

We may now define a new r.b. machine $\mathscr{M}'$, which replaces each test of x against j by tests on two new r.b. 0-testable counters, as in the proof of Proposition 2.7. The resulting machine $\mathscr{M}'$ simulates $\mathscr{M}''$, hence also simulates $\mathscr{M}$.

We show that each pair of new counters in $\mathscr{M}'$ only makes at most one reversal. The case of test $x < j$ may be simulated in $\mathscr{M}'$ by first copying x in two new, unused counters y and z, then by testing y against j (by repeatedly decreasing and testing against 0) as shown in Fig. 3; finally, the copy z can be used instead of x during the remainder of the run. The procedure makes one reversal for y and may cause a reversal for x (but only if x was still in the nondecreasing phase); both counters x and y are never used again in the same run, therefore they are one-reversal-bounded. Counter z is only increased, hence does not make any reversal during the procedure; when used instead of x in the remainder of the run, it will make at most one reversal. The other cases of tests, i.e., $x = j$ and $x > j$, are analogous (actually, simpler).

Since for every a run of $\mathscr{M}''$ there are at most 6 tests against j, $\mathscr{M}'$ simulates $\mathscr{M}''$ (and $\mathscr{M}$) with a total of 12 additional 1-r.b. counters. □

Lemma 2.11. *For every k-DCM $\mathscr{M}$ with a finite number n of k-testable r-reversal-bounded counters, there exists a 0-DCM $\mathscr{M}'$ that simulates $\mathscr{M}$, with $12 * k * n * r$ additional r.b. 0-testable counters.*

Proof. As in the case of discrete counters, one can always assume that reversal-bounded machines are *one*-reversal-bounded machines; indeed, each sequence of "increments, then decrements" can be simulated by a 1-r.b. counter, and thus a counter doing r reversals can be simulated by r 1-r.b. counters.

Then, the construction of $\mathscr{M}'$ is done as follows: for each counter x_i with $i \in [1, n]$ and for each constant j with $j \in [1, k]$, we remove each test of x_i against j by applying Lemma 2.10. This may produce at most $12 * k * n * r$ additional r.b. 0-testable counters. □

Now, we may deduce Theorem 2.12.

Theorem 2.12. *Reversal-bounded k-DCM are equivalent to reversal-bounded DCM, for any $k \geq 0$.*

This theorem immediately generalizes the main result of [8], recalled here as Proposition 2.9.

3. Decidability and undecidability results

The following table summarizes the results about DCM and their variations. The results in ***bold slanted*** characters are proved in this paper, and the others were proved (or inferable) from previous papers, namely [8] and [2]. There are four possible entries in this chart: "?" if we do not know whether the state reachability problem is decidable, "U" if it is undecidable, "D" if it is decidable, and "C" if the binary reachability relation is computable and definable in a decidable logic. The "+ r.b." (resp. "+ k-test. r.b.") means that the machine is extended with a finite number of reversal-bounded dense-choice counters (resp. reversal-bounded k-testable dense-choice counters).

Counters		DCM	Bounded k-DCM	DCM + r.b.	DCM + k-test. r.b.
Purely dense-choice	1	C	***C***	C	***C***
	2	D	?	?	?
	3	?	?	?	?
	4	U	***U***	U	***U***
Dense-choice	1	C	***C***	C	***C***
	2	U	U	U	U

In the remainder of this section, we prove two of these new results; the other new results are proved in the previous section or directly inferable. Notice that the seven open problems could be solved by only two or three proofs that subsume other results. However, the intuitions about 1 or 4 counters do not fit the case of 2 or 3 counters, and the proof techniques get even more complex when we use k-testable counters.

3.1. Undecidability for bounded purely dense-choice counters

Given a DCM $\mathscr{M}$, a counter x of $\mathscr{M}$ is *b-bounded*, $b \geq 0$, if $x \leq b$ along every run of $\mathscr{M}$. For instance, a 1-bounded counter can assume any non-negative value up to 1. A counter is *bounded* if it is b-bounded for some $b \geq 0$.

Given $b \geq 0$, if a machine $\mathscr{M}$ has a b-bounded b-testable dense-choice counter x, then one can assume that $\mathscr{M}$ must crash not only when trying to decrement x below 0, but also when trying to set $x > b$. Indeed, if x was not bounded,

$\mathscr{M}$ could be modified to test at each step whether $x \leq b$, crashing if this is not the case (which would force x to be bounded).

Bounded *integer* counters have a finite set of possible values, which can be encoded into the control states. However, bounded *dense-choice* counters have an infinite set of possible values: a DCM with several bounded counters is a powerful model, as shown next. In general, the state reachability problem is a simpler problem than computing the binary reachability. Nevertheless, the following proposition shows that even for state reachability, having only four 1-bounded counters implies undecidability.

Proposition 3.1. *The state reachability problem is undecidable for bounded purely-DCM.*

Proof. We show that the state reachability problem for a DCM with four purely dense-choice 1-bounded 1-testable counters is undecidable, which entails this proposition. The result follows the lines of the proof in [8] that 4 purely dense-choice counters are enough to simulate a Minsky machine.

The original proof was based on using two counters z_1, z_2, initially set to 0, to store a fixed value δ, chosen at the beginning of the computation. From the initial state q_0, the machine $\mathscr{M}$ goes into state q_1 by making a fractional increment to z_1, i.e., $z_1 := z_1 + \delta$ for some $0 < \delta < 1$; thus, in state q_1, $z_1 = \delta_0$ (the first δ value that has been chosen) and $z_2 = 0$. Then, $\mathscr{M}$ makes the following operations $z_2 := z_2 + \delta$ and $z_1 := z_1 - \delta$ (both with the same δ value) and $\mathscr{M}$ goes from q_1 to q_2; in state q_2, $z_1 = \delta_0 - \delta_1$ and $z_2 = \delta_1$. In order to force δ_1 to be equal to δ_0, $\mathscr{M}$ may go from q_2 to q_3 only through a transition guarded by $z_1 = 0$. Symmetrically, $\mathscr{M}$ may go from q_3 to q_4 by doing $z_1 := z_1 + \delta$ and $z_2 := z_2 - \delta$ (still, both with the same δ); thus, in q_4, $z_1 = \delta_2$ and $z_2 = \delta_0 - \delta_2$, and $\mathscr{M}$ cycles back from q_4 to q_1 by checking $z_2 = 0$, so that all δ values chosen during an execution are exactly δ_0 (provided the machine is either in q_1 or q_3, and is able to progress). All this process allows to save and use the same value δ_0.

The two remaining counters are then incremented or decremented only of this fixed value δ_0: any integer value k is encoded as $k\delta_0$. Hence, the two counters behave like two discrete counters without any restriction. If the 4 counters are 1-bounded, then they can encode only up to an integer $m = \lfloor \frac{1}{\delta_0} \rfloor$. However, m is unbounded, since δ_0, selected non-deterministically once at the beginning of a computation, can be chosen to be arbitrarily small: if δ_0 is not small enough then the DCM will crash trying to increase one of its counters beyond 1 (for example, by resetting to the initial configuration so that δ_0 can be chosen again until it is small enough). Nevertheless, in every halting computation of a Minsky machine, the values encoded in its two counters are bounded (with a bound depending on the computation). Therefore, the final state of the DCM is reachable if, and only if, the simulated Minsky machine has one halting computation. Hence, the state reachability problem is undecidable. □

3.2. Decidability with one k-testable counter

Proposition 3.1 does not rule out decidability if using less than four counters, since its proof is based on a four-counter purely-DCM. In particular, we show in Proposition 3.4 that for a DCM with only one counter, the binary reachability can effectively be computed even if the counter is k-testable. This extension to k-testability is indeed far from obvious.

In fact, the construction of the proof of Proposition 2.7 can be applied for tests of the form $x > j$ or $x = j$, for any $j \leq k$; this construction can be simulated by a sequence of j unit decrements followed by a test $x > 0$ or $x = 0$, and then followed by j unit increments to restore the original value. However, it cannot be applied for tests of the form $x < j$, since this would require a (non-existent) additional counter to be able to restore the original counter value. The proof of Proposition 3.4 also requires the counter to be bounded, in order to avoid an unbounded number of crossings of threshold k.

Before that, a few more technical definitions and lemma are needed.

Elimination of k-tests without introducing new counters. The crux of Definition 3.2 and Lemma 3.3 is a procedure reducing to zero the maximal constant integer against which the dense-choice counter will be tested. In other words, k-tests are eliminated. The idea of this elimination procedure is to build a 0-DCM $\mathscr{M}_k = \langle S_k, T_k \rangle$ to mimic the behaviour of the original k-DCM $\mathscr{M} = \langle S, T \rangle$, by reflecting the possible variations of the counter value into its finite-state control. $\mathscr{M}_k$ is called a *finite-test* one-counter DCM, as formalized in Definition 3.2. In essence, a finite-test DCM encodes the counter values in its control states using a rough partitioning (similar to the regions of a Timed Automaton [9]).

Note that such *finite-test* DCM can be built for a k-DCM with any finite number of counters, as in [1]. However, since we use only one counter now, a real value x is used instead of a vector $\mathbf{x}$ of counter values.

Intuitively, $\mathscr{M}_k$ is built as follows. A state of S_k is a triple $\langle\langle s, d, f \rangle\rangle$, where $s \in S$, d is an integer in $\{0, \ldots, k\}$ and f is a symbol in $\{0, ½\}$. Component d is a discrete counter from 0 up to k, intended to represent $\lfloor x \rfloor$. Component f is intended to represent the fractional part of x: $f = 0$ is for the case $\lfloor x \rfloor = x$, $f = ½$ otherwise.

Then, the set of transitions T_k is such that all tests of counter x against a constant $0 < j \leq k$ are eliminated and replaced by "finite-state tests" on d and f. For instance, a test $x > j$ is replaced by a test $d > j \vee (d = j \wedge f = ½)$. Only tests against 0 are replicated in T_k.

Define, for any real number y, $\mathrm{fr}(y) = 0$ if $\lfloor y \rfloor = y$ (i.e., y is an integer), and $\mathrm{fr}(y) = ½$ otherwise.

We formally detail the construction of the DCM $\mathscr{M}_k$. Since $\mathscr{M}_k$ has only one counter, it only contains guards that are either $(true)$ or testing the value of the single counter against 0, i.e., $(x = 0)$ or $(x > 0)$.

Definition 3.2. To any k-DCM $\mathscr{M} = \langle S, T\rangle$ with one counter, we associate a *finite-test* DCM $\mathscr{M}_k = \langle S_k, T_k\rangle$ also with one counter, where $S_k = S \times \{0, \ldots, k\} \times \{0, ½\}$ and T_k is defined as follows. For any transition $(s, (g, \lambda, a), s') \in T$, we build $(\langle\!\langle s, d, f\rangle\!\rangle, (g_k, \lambda, a), \langle\!\langle s', d', f'\rangle\!\rangle) \in T_k$ satisfying the following rules:

- g_k is the guard g if g only contains expressions of type $(x = 0)$ or $(x > 0)$; otherwise $g_k = (\text{true})$.
- if the following condition holds (for some j):
  - $g = (true)$, or
  - $g = (x < j)$ and $d < j$, or
  - $g = (x = j)$ and $d = j$ and $f = 0$, or
  - $g = (x > j)$ and $d > j$ or $(d = j$ and $f = ½)$,

  then $(\langle\!\langle s, d, f\rangle\!\rangle, (g_k, \lambda, a), \langle\!\langle s', d', f'\rangle\!\rangle) \in T_k$, where the values of d' and f' are obtained by satisfying one of the following five conditions:
  1. stay: $\lambda = 0 \wedge d' = d \wedge f' = f$;
  2. integer increment: $\lambda = 1 \wedge a = 1 \wedge ((d < k \wedge d' = d + 1 \wedge f' = f) \vee (d' = d = k \wedge f' = ½))$;
  3. integer decrement: $\lambda = -1 \wedge a = 1 \wedge ((0 < d \leq k \wedge d' = d - 1 \wedge f' = f) \vee (d' = d = k))$;
  4. fractional increment: $\lambda = 1 \wedge a = \Delta \wedge ((f = 0 \wedge d' = d \wedge f' = ½) \vee (f = ½ \wedge d' = d + 1 \wedge f' = 0) \vee (f = f' = ½ \wedge (d' = d + 1 \vee d' = d)))$;
  5. fractional decrement: $\lambda = -1 \wedge a = \Delta \wedge ((f = 0 \wedge d > 0 \wedge d' = d - 1 \wedge f' = ½) \vee (d > 0 \wedge d' = d - 1 \wedge f = f' = ½) \vee (f = ½ \wedge d' = d \wedge f' = 0) \vee (d' = d \wedge f = f' = ½))$.

Notice that when in cases (3), (4) and (5) of the above definition, more than one alternative may hold because of the disjunctions between parentheses, which correspond to non-deterministic choices of a δ by $\mathscr{M}_k$.

A special source of non-determinism also appears in case (3), when performing integer decrements: if the counter x went above k, then $d = k$ but one does not know whether $x - 1$ is above, equal to, or less than k; the right behaviour will be captured by the disjunctions in this rule.

Moreover, in case (5), it is implicit that if $f = 0 \wedge d = 0$ then $\mathscr{M}_k$ crashes, since there is no available alternative in T_k for a fractional decrement. A similar observation can be made in case (3) when $d = 0$, regardless of f since the decrement is integer.

The above definition of $\mathscr{M}_k$ entails, for example, that if at runtime there is a test $x = 0$ while $x = 0 \wedge (d > 0 \vee f = ½)$, then $\mathscr{M}_k$ crashes.

A configuration $\langle\langle\!\langle s, d, f\rangle\!\rangle, x\rangle$ of a finite-test DCM $\mathscr{M}_k$ is *consistent* if, either $(x \leq k \wedge d = \lfloor x \rfloor \ \wedge \ f = \mathrm{fr}(x))$ or $(x > k \wedge d = k \ \wedge \ f = ½)$ holds. Hence, in a consistent configuration, a test of a counter against a constant $j \leq k$ gives the same result as a test against the d and f components of the state.

In general, $\mathscr{M}_k$ may also reach non-consistent configurations. A run of $\mathscr{M}_k$ is *consistent* if it goes through consistent configurations only.

Now, we need a technical lemma showing that, for any k-DCM, we can build an "equivalent" finite-test DCM, with a notion of equivalence detailed in the lemma by two requirements.[2] This result will be used as a basis for the construction in the proof of Proposition 3.4.

Lemma 3.3. *Let $\mathscr{M} = \langle S, T\rangle$ be a DCM with one k-testable counter, given $k > 0$. Then the finite-test DCM $\mathscr{M}_k = \langle S_k, T_k\rangle$, with only one 0-testable counter, is such that:*

1. *Every run of $\mathscr{M}$ is also a run of $\mathscr{M}_k$, i.e.:*
   For every run of $\mathscr{M}$ of length $h \geq 0$: $\langle s^1, x^1\rangle \to^h_{\mathscr{M}} \langle s^h, x^h\rangle$, there exists a run of length h for $\mathscr{M}_k$: $\langle\langle\!\langle s^1, d^1, f^1\rangle\!\rangle, x^1\rangle \to^h_{\mathscr{M}_k} \langle\langle\!\langle s^h, d^h, f^h\rangle\!\rangle, x^h\rangle$;
2. *A consistent run of $\mathscr{M}_k$ is also a run of $\mathscr{M}$, i.e.:*
   For every consistent run of $\mathscr{M}_k$ of length $h \geq 0$: $\langle\langle\!\langle s^1, d^1, f^1\rangle\!\rangle, x^1\rangle \to^h_{\mathscr{M}_k} \langle\langle\!\langle s^h, d^h, f^h\rangle\!\rangle, x^h\rangle$, there exists a run of length h for $\mathscr{M}$ of the form: $\langle s^1, x^1\rangle \to^h_{\mathscr{M}} \langle s^h, x^h\rangle$.

Proof. Condition (1) of the lemma holds, since by construction, every transition in a run of $\mathscr{M}$ may be replicated in a run of $\mathscr{M}_k$. Hence, a run of $\mathscr{M}$ is also a run of $\mathscr{M}_k$, by adding suitable components to the state.

Condition (2) also follows, since in a consistent configuration every test of a counter against $j > 0$ (with $j \leq k$) is equivalent to a finite-state test, as shown above. Hence, a consistent run in $\mathscr{M}_k$ may be replicated also in $\mathscr{M}$. □

Notice that, in general, a run of a finite-test DCM $\mathscr{M}_k$ is *not* also a run of $\mathscr{M}$, since in $\mathscr{M}_k$ there is no test against constants (excepted 0), which are replaced by tests on state components d and f. Indeed, the fractional increments/decrements of the counter may lead to a non-consistent configuration where a counter value x is not compatible with the value of state

[2] See [1] for a version of this lemma generalized to n counters and preserving reversal-boundedness.

components d and f, e.g., $x \leq j$, for some $j > 0$, and on the other hand $d > j$. Therefore, the tests on d and f may not give the same results as a test on the actual value of x, and hence the run may be possible in $\mathscr{M}_k$ but not in $\mathscr{M}$. This problem is taken care of by consistency requirement 2 of Lemma 3.3.

Let us now state and prove Proposition 3.4. Again, we use the notion of mixed formula, which we develop in Section 4.1; remember that it is a decidable logic, equivalent to FO($\mathbb{R}, \mathbb{Z}, +, <$). Moreover, we know from Proposition 2.9 that the binary reachability of a DCM is definable by a mixed formula.

Proposition 3.4. *The binary reachability of a DCM with a single bounded k-testable counter is definable by a mixed formula, for every* $k \geq 0$.

Proof. Let $\mathscr{M} = \langle S, T \rangle$ be a one-counter DCM, such that its only counter is b-bounded. Since there is only one counter, a real value x is used instead of a vector $\mathbf{x}$ of counter values. We prove the case $b = k$, since if $b < k$ then all tests against $j > b$ are just false, while if $b > k$ then simply $\mathscr{M}$ will not use the tests against $j > k$.

Let $\mathscr{M}' = (S', T')$ be the finite-test DCM with one free 0-testable counter, as defined by Lemma 3.3, with $\langle\langle s, d, f \rangle\rangle \in S'$ for every $s \in S, d \in \{0, \ldots, k\}, f \in \{0, ½\}$.

We claim that for every $x^0, x^1 \in \mathbb{R}_+$ and $s^0, s^1 \in S$, with $s_{\text{init}}, s_{\text{final}} \in S'$,

$$\begin{aligned} &\langle s^0, x^0 \rangle \rightsquigarrow_{\mathscr{M}} \langle s^1, x^1 \rangle \quad \text{if, and only if,} \\ &\langle\langle s_{\text{init}}, \lfloor x^0 \rfloor, \text{fr}(x^0) \rangle\rangle, x^0 \rangle \rightsquigarrow_{\mathscr{M}'} \langle\langle s_{\text{final}}, \lfloor x^1 \rfloor, \text{fr}(x^1) \rangle\rangle, x^1 \rangle \end{aligned} \tag{1}$$

The main proposition follows then immediately, since relation (1) is decidable and can be described by a mixed formula.

"Only If": This part is guaranteed by Condition (2) of Lemma 3.3.

"If" part: Suppose that Formula (1) holds. We need to show that $\langle s^0, x^0 \rangle \rightsquigarrow_{\mathscr{M}} \langle s^1, x^1 \rangle$. Condition (3) of Lemma 3.3 only applies to consistent runs, and runs of $\mathscr{M}'$ are not necessarily consistent. However, each fractional increment/decrement in a run of $\mathscr{M}'$ is chosen non-deterministically. Hence the value of x can be adjusted for consistency with d and f. The proof of this claim requires some preliminary definitions and propositions.

A *consistent version* of a configuration $c = \langle\langle\langle s, d, f \rangle\rangle, x \rangle$ is any configuration $c' = \langle\langle\langle s, d, f \rangle\rangle, x' \rangle$, for some $x' \in \mathbb{R}_+$, which is consistent.

We claim that for every consistent configuration $c_0 = \langle\langle\langle s, d, f \rangle\rangle, x^0 \rangle$ and for every configuration $c_1 = \langle\langle\langle s_1, d_1, f_1 \rangle\rangle, x^1 \rangle$,

$$\begin{aligned} &\text{if } c_0 \xrightarrow{g', \lambda, a}_{\mathscr{M}'} c_1, \\ &\text{then there exists a consistent version } c_1' \text{ of } c_1 \text{ such that } c_0 \xrightarrow{g', \lambda, a}_{\mathscr{M}'} c_1', \\ &\text{defined by } c_1' = \langle\langle\langle s_1, d_1, f_1 \rangle\rangle, x'^1 \rangle, \text{ for some } x' \in \mathbb{R}_+. \end{aligned} \tag{2}$$

If $a = 1$, then c_1 is already consistent by definition of $\mathscr{M}'$. Hence, only a fractional increment/decrement (i.e. if $a = \Delta$ and $\lambda \neq 0$) may lead to an inconsistent configuration.

A special case of configuration is a *zero-conf*, i.e. any configuration of $\mathscr{M}'$ of the form $\langle\langle\langle s, (0,0)^n \rangle\rangle, \mathbf{0} \rangle$. We can further assume that, in a finite-test DCM $\mathscr{M}'$, every zero-conf is always consistent, since $\mathscr{M}'$ can test that every component of $\mathbf{x}$ is actually 0 (and crashes otherwise).

Assume first $x^0 = 0$ (i.e., c_0 is a zero-conf). Hence, $d_1 = d$, $f_1 = ½$ and $x = \delta$ for some δ such that $0 < \delta < 1$. Hence, c_1 is already consistent.

Assume now $x^0 > 0$. Hence, d_1 may differ from d at most by one.

To proceed, we need an additional observation. For all states $\langle\langle s, d, f \rangle\rangle$ and $\langle\langle s', d', f' \rangle\rangle$ of $\mathscr{M}'$, for all $(g, \lambda, a) \in \Sigma$, for all $x \in R_+$, $\forall \epsilon \in [-1, 1]$, with $0 \leq x + \epsilon < k + 1$, if

$$\langle\langle\langle s, d, f \rangle\rangle, x \rangle \xrightarrow{g', \lambda, a}_{\mathscr{M}'} \langle\langle\langle s', d', f' \rangle\rangle, x + \epsilon \rangle$$

then for every $x' \in \mathbb{R}_+$, such that $0 \leq x' + \epsilon < k + 1$ the same move can be repeated from x':

$$\langle\langle\langle s, d, f \rangle\rangle, x' \rangle \xrightarrow{g', \lambda, a}_{\mathscr{M}'} \langle\langle\langle s', d', f' \rangle\rangle, x' + \epsilon \rangle \tag{3}$$

Property (3) is obvious since $\mathscr{M}'$ can only test x for zero, hence it cannot differentiate x from x' before the move and it may apply the same increment.

By property (3), it is possible to make the same move from c_0 using a different increment (or decrement): x can be increased (or decreased) by a value (larger or smaller than δ, but always in the interval $]0, 1[$) which is enough to make up for the difference so that the configuration becomes consistent.

Let $c_0 \to_{\mathscr{M}'} c_1 \to_{\mathscr{M}'} \ldots \to_{\mathscr{M}'} c_l$ be a run of $\mathscr{M}'$, with $l \geq 0$. We now prove by induction on l that if c_0 is consistent, then there is another run of $\mathscr{M}'$ denoted by $c_0 \to_{\mathscr{M}'} c_1' \to_{\mathscr{M}'} \ldots \to_{\mathscr{M}'} c_l'$ where each c_i' is a consistent version of c_i, $1 \leq i \leq l$.

The case $l = 0$ is trivial (with $c_0 = c_0'$). Suppose $l > 0$. By induction hypothesis $c_0 \to_{\mathscr{M}'} c_1' \to_{\mathscr{M}'} \ldots \to_{\mathscr{M}'} c_{l-1}'$, each c_i' being a consistent version of c_i, $1 \leq i \leq l-1$. By Property (2), we can find a consistent version c_l' of c_l such that $c_{l-1}' \to_{\mathscr{M}'} c_l'$.

By Condition (2) of Lemma 3.3, every consistent run of $\mathscr{M}'$ is also a run of $\mathscr{M}$; hence the proof is completed. □

The proof is immediately extendable to the case where $\mathscr{M}$ has also discrete reversal-bounded counters, which are in no way influenced by the above construction. If the reversal-bounded counters are dense-choice, however, decidability is still open. The proof also holds for a DCM having the ability to test if a counter value is an integer: indeed, the finite-test DCM encodes the integer and fractional parts of the counter value, hence testing whether the fractional part is 0 is equivalent to testing whether the counter value is integer.

4. Logical characterization of DCM

4.1. Preliminary results about mixed formulae

We consider here the language of mixed formulae, defined in [17], and adapted from Presburger arithmetic. The language has two sorts of variables: real variables, denoted by $x, x', x_1, \ldots$ and integer variables, denoted by $y, y', y_1, \ldots$; the latter are a subsort of the former. The constants are 0 and 1, the operations are $+$, $-$, $\lfloor . \rfloor$ and the relations are equality $=$, ordering $<$ and congruences $\equiv_d$ for every constant $d \in \mathbb{N}$. Definition 4.1 formalizes this idea:

Definition 4.1. A *mixed formula* is inductively defined as follows. A *mixed linear expression* E is defined by the following grammar, where x is a real variable and y is an integer variable:

$$E ::= 0 \mid 1 \mid x \mid y \mid E + E \mid E - E \mid \lfloor E \rfloor$$

A *mixed linear constraint* C is defined by the following grammar, where d is a positive integer:

$$C ::= E = E \mid E < E \mid E \equiv_d E$$

A *mixed formula* F is defined by the following grammar, where $x \in \mathbb{R}$ and $y \in \mathbb{Z}$:

$$F ::= C \mid \neg F \mid F \wedge F \mid \exists x.F \mid \exists y.F$$

The semantics of a mixed formula is like in the reals, $\lfloor r \rfloor$ being the integer part of its real argument r, and $r_1 \equiv_d r_2$ holding iff $r_1 - r_2 = vd$ for some integer v.

Typically, one can use shorthands, e.g. writing $3x$ for $x + x + x$, or introducing other common operators (like $\geq$), etc.

Mixed formulae are equivalent to the well-known first-order additive theory of integers and reals $\mathrm{FO}(\mathbb{R}, \mathbb{Z}, +, <)$, since the floor operator $\lfloor x \rfloor = y$ can be rewritten as $\exists x'(0 < x' \wedge x' < 1 \wedge x - x' = y)$, and $x_1 \equiv_d x_2$ can be rewritten for a fixed $d > 0$ as $\exists y(x_1 - x_2 = \underbrace{y + \cdots + y}_{d})$. However, the main advantage of the richer syntax of mixed formulae is that it allows for quantifier elimination, which is not possible in $\mathrm{FO}(\mathbb{R}, \mathbb{Z}, +, <)$, as shown in Theorem 3.1 and Corollary 5.2 of [17].

4.2. Mixed formulae are definable by reversal-bounded DCM

It is well known that reversal-bounded discrete CM can define all Presburger formulae. Since Presburger logic admits effective quantifier elimination, the binary reachability of r.b. discrete CM can effectively define all Presburger relations. We prove that a similar result holds for r.b. DCM, using mixed formulae (and the effectiveness of quantifier elimination) instead of Presburger formulae.

Let $\mathbf{0}$ be the vector $(0, \ldots, 0)$ of size n, and let $F(z_1, \ldots, z_n)$ be a quantifier-free mixed formula in the free variables $z_1 \geq 0, \ldots, z_n \geq 0$. F is *definable* by a DCM $\mathscr{M}$ with at least n counters $x_1, \ldots, x_n$ (and possibly more) if $\mathscr{M}$, starting in a given initial configuration $\langle s, \mathbf{0} \rangle$, may reach all final configurations $\langle s_{fin}, \mathbf{x} \rangle$, and only them, such that $F(x_1/z_1, \ldots, x_n/z_n)$ holds (where x_i/z_i denotes a substitution of variable z_i with value x_i).

Proposition 4.2. *Let $F(x_1, \ldots, x_n, y_1, \ldots, y_p)$ be a quantifier-free mixed formula, with $x_1 \geq 0, \ldots, x_n \geq 0$ and $y_1 \geq 0, \ldots, y_p \geq 0$. Then F is definable by an r.b. DCM.*

The idea of this proof (detailed hereafter) involves several steps which can easily be understood. First, assume (w.l.o.g.) that F is in disjunctive normal form; then, transform it into a union of intersections of smaller formulae of the form $E \sim 0$, with $\sim \in \{>, =, <, \equiv_d, \not\equiv_d\}$. Every formula of the form $E \sim 0$ can be encoded using an r.b. DCM having n r.b. dense-choice counters x, p r.b. discrete counters y, and possibly also more r.b. counters: this machine makes an accepting run, with initial counter valuations equal to a non-deterministically-chosen assignment of the free variables of F, if and only if this assignment makes F true.

Then, we just have to connect each r.b. DCM as follows. Each machine has a final control state, which can be connected, by means of a transition, to the initial control state of another machine, thus forming another (larger) r.b. DCM. The intersection of formulae of the above form $E \sim 0$ can be obtained by a series of machines, where the final state of a first

machine is connected to the initial state of another machine: each component of the series verifies if the run is accepting, given an assignment encoded in additional r.b. counters. The last component is encoded by a machine whose final state leads to an accepting sink state (which is the final state of the overall r.b. DCM encoding F). The union of these "intersection machines" can be obtained by building an r.b. DCM that guesses an assignment, copies it into suitable r.b. counters and then non-deterministically enters the initial state of one of the intersection machines. The above sink state is reached if and only if formula F is satisfied.

Proof. Assume (w.l.o.g.) that F is in disjunctive normal form:

$$F = F_1 \vee F_2 \vee \ldots \vee F_m$$

Hence, F is the disjunction of clauses F_i of the form:

$$F_i = F_{i_1} \wedge F_{i_2} \wedge \cdots \wedge F_{i_{m_i}}$$

where each F_{i_j}, in the free variables $x_1, \ldots, x_n, y_1, \ldots, y_p$, can always be reduced, by pushing negation to the relational symbol and by elementary algebraic transformations, to the form:

$$E \sim 0$$

where $\sim \in \{>, =, <, \equiv_d, \not\equiv_d\}$. For instance, if F_{i_j} is $E_1 < E_2$ then one may check instead if $E_1 - E_2 < 0$, etc.

Below we show that for every F_{i_j}, there exists an r.b. DCM $\mathscr{M}_{i_j}$ with r.b. dense-choice counters $x_1, \ldots, x_n$ and r.b. discrete counters $y_1, \ldots, y_p$ (and possibly more r.b. counters $x_{n+1}, \ldots$ and $y_{p+1}, \ldots$) that accepts when relation F_{i_j} is verified on the initial values of the counters $x_1, \ldots, x_n, y_1, \ldots, y_p$.

This immediately entails that, for each clause F_i, there exists an r.b. DCM $\mathscr{M}_i$ that accepts if $F_{i_1} \wedge F_{i_2} \wedge F_{i_{m_i}}$ is verified on the initial values of its r.b. counters $x_1, \ldots, x_n, y_1, \ldots, y_p$. In fact, since F_i is the conjunction of all F_{i_j}, $\mathscr{M}_i$ is an r.b. DCM that first makes m_i copies of counters $x_1, \ldots, x_n, y_1, \ldots, y_p$ and then simulates each $\mathscr{M}_{i_j}$ started on one of the copies. $\mathscr{M}_i$ accepts if, and only if, all $\mathscr{M}_{i_j}$ accept.

Therefore, it is possible to build an r.b. DCM $\mathscr{M}'$ whose binary reachability describes relation F: $\mathscr{M}'$ starts with all counters equal to zero; first, it makes non-deterministic increments of each counter, guessing a tuple of values for $x_1, \ldots, x_n, y_1, \ldots, y_p$ such that at least one F_i (hence, also F) holds; second, it follows the computation of $\mathscr{M}_i$ described above, in order to verify that all guesses are correct. In order to end in a configuration in which the n first counters hold the values of the n variables making F true, we will copy these counters so that we do not modify them while verifying that they have been guessed right.

To show that for every i, j there actually exists an r.b. DCM $\mathscr{M}_{i_j}$ defining F_{i_j}, we first prove by induction on the structure of E that the value of E can be encoded by an r.b. dense-choice counter for $|E|$ and a flag in the control state for the sign of E. The following encodings are quite simple, but since there are several cases and their enumeration may seem tedious, we prefer detailing them to convince the reader.

Recall that a counter value can always be copied a fixed number of times, using the encoding of Fig. 2b.

The base steps of the induction are the cases when E is 0, 1, x_i, y_j, which are obvious.

Assume now that E is $(E_1 + E_2)$, with a copy of $|E_1|$ and $|E_2|$ stored in two suitable r.b. counters, with their sign flags in the finite state control. We can assume that E_1 and E_2 have the same sign (e.g., if $E_1 \geq 0$ and $E_2 < 0$ then $E_1 + E_2$ can be rewritten as $E_1 - |E_2|$). We only need to consider the case when both are positive (if both are negative, then compute $|E_1| + |E_2|$ and store the result in an r.b. counter with the sign flag being negative). The addition $E_1 + E_2$ can then be done using the encoding of Fig. 2c.

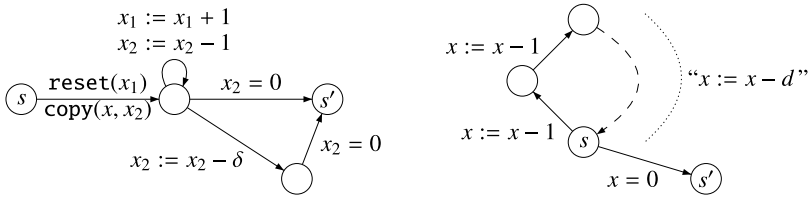
Assume now that E is $(E_1 - E_2)$, again with a copy of $|E_1|$ and $|E_2|$ stored in two suitable r.b. counters x_1 and x_2 (respectively), with sign flags. We only need to consider the case where both E_1 and E_2 are positive (the other cases can be easily eliminated or reduced to an application of $+$). We can also assume that $E_1 \geq E_2$, since if $E_2 > E_1$ then the machine will guess it and may compute $E_2 - E_1$ instead, changing the sign of the result. The computation of $E_1 - E_2$ can then be done using the encoding of Fig. 2d. Notice that if the machine made the wrong guess that $E_1 \geq E_2$, while instead $E_2 < E_1$, then this procedure will crash (hence, the non-deterministic choice has to be the correct one).

Finally, assume that E is $\lfloor E' \rfloor$; then the automaton on Fig. 4a can reach s' from s if and only if $y = \lfloor x \rfloor$.

We just showed how to encode a mixed linear expression E in an r.b. DCM. To complete the proof that there is an r.b. DCM $\mathscr{M}_{i_j}$ accepting F_{i_j}, it is enough to show that there exists an r.b. DCM $\mathscr{M}$ checking whether $E \sim 0$ (since F_{i_j} is of this form). Since the value of $|E|$ is stored in an r.b. counter x, with a flag in the control state for the sign of E, then tests $E < 0$ and $E > 0$ are immediate. Of course, $E = 0$ is trivial too, since it can be tested by a guard $(x = 0)$. The two remaining cases are zero-congruences modulo an integer d. The automaton on Fig. 4b can reach s' from s if and only if $E \equiv_d 0$, for a given integer d (the value of E being stored into counter x).

To accept if $E \not\equiv_d 0$, then $\mathscr{M}$ first checks if $x - \lfloor x \rfloor > 0$, accepting if this is the case. If $x - \lfloor x \rfloor = 0$ then $\mathscr{M}$ guesses the integer constant $v \in \{0, \ldots, d\}$ such that $E - v \equiv_d 0$. This can be computed as explained above.

Thus, we gave a constructive proof that there exists an r.b. DCM defining any mixed formula. □



(a) Storage of $\lfloor x \rfloor$ into x_1 (b) Test of $x \equiv_d 0$

Fig. 4. Encodings of "integer part" and "modulo".

Since the quantifier elimination of mixed formulae is effective, and since we can encode negative variables with a sign bit in the control states, then we can directly deduce the following theorem:

Theorem 4.3. *Any mixed formula can be defined by an r.b. DCM.*

This theorem is actually dual to the one in [8] (cited here as Proposition 2.9), which states that the binary reachability of an r.b. DCM (with an additional free counter) is a mixed formula. Hence, we get an exact characterization of r.b. DCM.

5. Comparing DCM with other models

In this section, we relate DCM to more familiar automata-based models. Indeed, the ability of a DCM to non-deterministically choose a value for incrementing or decrementing counters seems rather uncommon.

Hybrid automata [13] contain the class of DCM, since any predicate or formula of any logic could be used for labelling transitions. However, hybrid automata are so general that reachability problems remain undecidable even for very restricted subclasses. Although DCM contain Minsky machines by definition, reversal-bounded DCM have interesting decidability properties such as shown in Proposition 2.9 and Theorem 4.3. But how does the expressivity of a DCM compare to that of timed automata or counter automata?

As a matter of fact, Theorem 4.3 can be combined with other results about mixed formulae. For example, we know that the binary reachability of a flat counter automaton [5] or of a timed automaton [18] is definable by a mixed formula. Hence, we can construct an r.b. DCM that accepts exactly the binary reachability of a given flat counter automaton or timed automaton. On the other hand, a timed automaton can obviously not simulate a DCM (even reversal-bounded), since DCM have the ability to count with integers.

One of the models of counter systems has been defined by Comon and Jurski [5], with transitions labelled by relations instead of functions. We now show that DCM are more expressive than this model of counter systems that we will call CJ.

First, let us recall definitions from [5]. Let $D \in \{\mathbb{Z}, \mathbb{Q}, \mathbb{R}\}$. Let CJ_D be a class of counter systems, i.e. of tuples $\langle S, T \rangle$ such that S is a finite set of control states and $T \subseteq S \times \Sigma \times S$ is a finite set of transitions. In a CJ_D, the formulas of Σ are of the form $y_i \bowtie y_j + c_{i,j}$, where $\bowtie \in \{<, \leq, =, \geq, >\}$, $c_{i,j} \in D \cap \mathbb{Q}$, and y_i is either x_i or x'_i, with $i \in \{1, \ldots, n\}$ and counters values $\mathbf{x} = (x_1, \ldots, x_n) \in D^n$.

We show that for every $D \in \{\mathbb{Z}, \mathbb{Q}, \mathbb{R}\}$, any CJ_D can easily be encoded by a DCM. Three possible valuation domains D can be considered for CJ_D.

Assume $D = \mathbb{Z}$. A $CJ_{\mathbb{Z}}$ transition of the form $x \bowtie c$ and can trivially be simulated by the guards of a k-DCM. A $CJ_{\mathbb{Z}}$ transition of the form $y_i \bowtie y_j + c$ and can easily be simulated by a DCM. For example, $x_1 < x_2 + c$ is simulated by a DCM consisting of a sequence of c integer increments of x_2 followed by a loop labelled by a fractional decrement on both x_1 and x_2, with a single final transition guarded by $x_1 = 0 \wedge x_2 > 0$. Other $CJ_{\mathbb{Z}}$ transitions can be simulated by DCM using a similar mechanism.

The case $D = \mathbb{Q}$ can be identified with the case $D = \mathbb{Z}$ by using a least common multiple for constants c, like in the proof of Proposition 2.2.

Since the constants c are always in $\mathbb{Q}$ by definition, the case $D = \mathbb{R}$ is similar to the other two cases.

We claim that in the general case, CJ_D cannot simulate DCM. However, one-counter k-DCM are a special case of $CJ_{\mathbb{R}}$. Indeed, guards and integer increments/decrements of a k-DCM syntactically belong to the labels of transitions in any CJ_D. Moreover, fractional increments/decrements can surprisingly be encoded by transitions of CJ_D: $x' = x + \delta$ can be encoded by $x' > x \wedge x' < x + 1$, and $x' = x - \delta$ can be encoded by $x' < x \wedge x' > x - 1$.

The interesting property behind this latter encoding is that both models offer a non-deterministic choice each time a transition is fired. The main difference between these two sources of non-determinism is that in DCM, the value δ will be randomly chosen in $]0, 1[$ each time the transition is fired, whereas relations on the transitions of a CJ_D define intervals within D in which a specific value will be randomly chosen for a given execution.

Nevertheless, a DCM with two or more dense-choice counters has an ability that CJ_D cannot simulate: it can partially control the non-determinism, by incrementing or decrementing several counters of the same amount δ. Indeed, there would be a link between such counters that a CJ_D may not be able to reproduce.

6. Conclusions

The goal of this paper is to shed a more formal light on DCM, hence clarifying their relation with discrete counter machines and other models of systems. Although this papers presents DCM as an extension of Counter Machines, well-known, typical results of the latter model may not be extended to DCM or, when they may, they require a much more difficult proof.

An example of the former case, proved in this paper, is that restricting dense-choice counters to be bounded does not imply decidability, while with the same restrictions discrete counter machine are just finite automata.

An example of the latter case, considered in this paper, is allowing dense-choice counters to be compared against an integer constant k, and not only with 0. This paper shows that this extended dense-choice counters are not more powerful, even in the case of r.b. DCM, or of DCM with a single bounded counter, analogously to discrete counter machines.

Another example of extension, proved in this paper, is an exact characterization of r.b. DCM with the well-known first-order additive logic of integers and reals, analogous to the well-known characterization of discrete r.b. counter machines with Presburger logic.

Other lines of investigations are still open; for example, there are missing items in the table of page 24, which do not seem to be easily inferable from known results. One could also try to extend other rich properties of discrete counter machines to DCM, e.g. considering one-counter machines as well as machines without zero ($x = 0$) tests (leading to what could be called "Dense-choice Petri Nets").

References

[1] F. Bouchy, A. Finkel, P. San Pietro, Dense-choice Counter Machines revisited, in: INFINITY'09, in: Electronic Proceedings in Theoretical Computer Science, vol. 10, 2009, pp. 3–22.
[2] M.L. Minsky, Computation: Finite and Infinite Machines, Prentice-Hall, Inc., NJ, USA, 1967.
[3] S. Bardin, A. Finkel, J. Leroux, L. Petrucci, FAST: acceleration from theory to practice, Int. J. Softw. Tools Technol. Transf. 10 (5) (2008) 401–424.
[4] S. Demri, A. Finkel, V. Goranko, G. van Drimmelen, Towards a model-checker for counter systems, in: ATVA'06, in: Lecture Notes in Computer Science, vol. 4218, Springer, 2006, pp. 493–507.
[5] H. Comon, Y. Jurski, Multiple counters automata, safety analysis and Presburger arithmetic, in: CAV, in: Lecture Notes in Computer Science, vol. 1427, 1998, pp. 268–279.
[6] M. Bozga, R. Iosif, Y. Lakhnech, Flat parametric counter automata, in: ICALP (2), in: Lecture Notes in Computer Science, vol. 4052, 2006, pp. 577–588.
[7] Z. Dang, O.H. Ibarra, P. San Pietro, G. Xie, Real-counter automata and their decision problems, in: FSTTCS, in: Lecture Notes in Computer Science, vol. 3328, 2004, pp. 198–210.
[8] G. Xie, Z. Dang, O.H. Ibarra, P. San Pietro, Dense counter machines and verification problems, in: CAV, in: Lecture Notes in Computer Science, vol. 2725, 2003, pp. 93–105.
[9] R. Alur, D.L. Dill, A theory of timed automata, Theoret. Comput. Sci. 126 (2) (1994) 183–235.
[10] L. Recalde, S. Haddad, M. Silva, Continuous Petri nets: expressive power and decidability issues, in: ATVA'07, in: Lecture Notes in Computer Science, vol. 4762, Springer, 2007, pp. 362–377.
[11] P. Bouyer, S. Haddad, P.-A. Reynier, Timed Petri nets and timed automata: on the discriminating power of Zeno sequences, Inform. and Comput. 206 (1) (2008) 73–107.
[12] F. Bouchy, A. Finkel, A. Sangnier, Reachability in timed counter systems, in: INFINITY'08, in: Electronic Notes in Theoretical Computer Science, vol. 239C, Elsevier, 2009, pp. 167–178.
[13] R. Alur, C. Courcoubetis, T.A. Henzinger, P.-H. Ho, Hybrid automata: an algorithmic approach to the specification and verification of hybrid systems, in: Hybrid Systems, in: Lecture Notes in Computer Science, vol. 736, Springer, 1992, pp. 209–229.
[14] M. Dotoli, M.P. Fanti, A. Giua, C. Seatzu, First-order hybrid Petri nets. An application to distributed manufacturing systems, Nonlinear Anal. Hybrid Syst. 2 (2) (2008) 408–430.
[15] O.H. Ibarra, Reversal-bounded multicounter machines and their decision problems, J. ACM 25 (1) (1978) 116–133.
[16] A. Finkel, A. Sangnier, Reversal-bounded counter machines revisited, in: MFCS, in: Lecture Notes in Computer Science, vol. 5162, 2008, pp. 323–334.
[17] V. Weispfenning, Mixed real-integer linear quantifier elimination, in: ISSAC, ACM, 1999, pp. 129–136.
[18] Z. Dang, Pushdown timed automata: a binary reachability characterization and safety verification, Theoret. Comput. Sci. 302 (1–3) (2003) 93–121.