Noname manuscript No.
(will be inserted by the editor)

Clause-Reachability is Undecidable in Legal Contracts

Giorgio Delzanno · Cosimo Laneve · Arnaud Sangnier · Gianluigi Zavattaro

Received: date / Accepted: date

Abstract *Micro-Stipula* is a stateful calculus in which clauses can be activated either through interactions with the external environment or by the evaluation of time expressions. Despite the apparent simplicity of its syntax and operational model, the combination of state evolution, time reasoning, and nondeterminism gives rise to significant analytical challenges. In particular, we show that determining whether a clause is never executed is undecidable. We formally prove that this undecidability result holds even for syntactically restricted fragments: namely, the *time-ahead* fragment, where all time expressions are strictly positive, the *instantaneous* fragment, where all time expressions evaluate to zero, and the *determinate* fragment, where the initial states of functions and events are disjoint. On the other hand, we identify a decidable subfragment: at the intersection of the instantaneous and determinate fragments reachability becomes decidable.

Keywords Reachability analysis · legal contracts · *Stipula* · Minsky machines · well-structured transition systems

1 Introduction

Micro-Stipula, noted μ*Stipula*, is a basic calculus defining *legal contracts*, namely sets of clauses that are either (*a*) *parameterless functions*, to be invoked by the external environment (e.g. by the parties involved in the contract), or (*b*) *events* that are triggered at given times (e.g. actions to be executed as soon as a deadline expires). The calculus has been devised to study the presence of clauses in legal contracts written in *Stipula* [11, 12] that can never be applied because of unreachable circumstances or of wrong time constraints – so-called *unreachable clauses*. In the legal contract domain, removing such clauses when the contract is drawn up is substantial because they might be considered too oppressive by parties and make the legal relationship fail.

While dropping unreachable code is a very common optimization in compiler construction of programming languages [6,9], the presence of time expressions in μ*Stipula* events makes the optimization complex. In particular, when the time expressions are logically inconsistent with the contract behaviour, the corresponding event (and its continuation) becomes unreachable.

In [25] we have defined an *unreachable clauses* analyzer that uses symbolic expressions to approximate time expressions at static-time and that computes an over-approximation of reachable clauses by means of a closure operation based on a fixpoint technique. The analyzer, whose prototype is at [18], is sound (every clause it spots is unreachable) but not complete (there may be unreachable clauses that are not recognized). The definition of a complete algorithm for unreachable clauses was left as an open problem.

In this paper we study the foregoing problem for three fragments of μ*Stipula* that are used to model distinctive elements of legal contracts:

- μ*Stipula*$^{\mathtt{TA}}$: time expressions are always positive – the corresponding events allow one to define *future obligations* such as the power to exercise an option may only last until a deadline is met. For example, a standard clause in a legal contract for renting a device is

G. Delzanno · A. Sangnier
DIBRIS, University of Genova, Italy
E-mail: {giorgio.delzanno, arnaud.sangnier}@unige.it

C. Laneve · G. Zavattaro
DISI, University of Bologna, Italy
E-mail: {cosimo.laneve, gianluigi.zavattaro}@unibo.it

– *The Borrower shall return the device* `k` *hours after the rental and will pay Euro* `cost` *in advance where half of the amount is of surcharge for late return.*

This clause is transposed in μ*Stipula* with an event like

```
now + k ≫ @Using {
    // send money cost to the Lender
} ⇛ @End
```

which is affirming that the money `cost` is sent to the Lender if the Borrower is `Using` the device when the deadline expires.

μ*Stipula*$^{\mathtt{I}}$: time expressions are always `now` – the corresponding events permit to define *judicial enforcements* such as the *immediate* activation of a dispute resolution mechanism by an authority when a party challenges the content or the execution of the contract. For example, a contract for renting a device might also contain a clause for resolution of problems:

– *If the Lender detects a problem, he may trigger a resolution process that is managed by the Authority. The Authority will immediately enforce resolution to either the Lender or the Borrower.*

In this case, the μ*Stipula* clause is the event

```
now ≫ @Problem {
    // immediately enforce resolution to Lender
    // or Borrower
} ⇛ @Solved
```

where the *immediate resolution* is implemented by a time expression `now`. In this case, if the Lender and the Borrower have not yet resolved the issue – the contract is in a state `Problem` – then the Authority enforces a resolution by, for example, sending a part of `cost` to Lender and the remaining part to Borrower.

μ*Stipula*$^{\mathtt{D}}$: functions and events do not have initial states in common; events in this fragment allow one to model *exceptional behaviours* that must be performed *before* any party may invoke a function. This type of restriction is quite common in legal contracts, particularly when formalizing a clause that outlines the consequences of a party breaching a condition. Section A contains a legal contract written in μ*Stipula*$^{\mathtt{D}}$.

We demonstrate that, even for the aforesaid fragments of μ*Stipula*, there is no complete algorithm for determining unreachable clauses. The proof technique is based on reducing the halting problem for Minsky machines (finite state machines with two registers) [28] to the clause-unreachability problem. The μ*Stipula* encodings of Minsky machines model registers' values with multiplicities of events and extensively use the features that (a) events preempt both the invocation of functions and the progression of time and (b) events that are not executed in the current time slot are garbage-collected when the time progresses. The encoding of μ*Stipula*$^{\mathtt{TA}}$ is particularly complex because we need to decouple in two different time slots the events corresponding to the two registers and recreate them when the time progresses.

We then restrict to μ*Stipula*$^{\mathtt{DI}}$, a fragment of μ*Stipula* that is the intersection of μ*Stipula*$^{\mathtt{I}}$ and μ*Stipula*$^{\mathtt{D}}$, and prove that the corresponding contracts are an instance of well-structured transition systems (WSTS) [19], whereby the reachability of clause is decidable as it is an instance of the *coverability* problem (which is decidable in WSTS [19]). To achieve this result, we had to modify the semantics of μ*Stipula* by restricting the application of the time progression to states in which functions can be invoked (thus it is disabled in the states where events are executed). Hereafter, the correspondence between the models using the two different progression rules has been analyzed to demonstrate reachability results for μ*Stipula*$^{\mathtt{DI}}$.

The rest of this paper is structured as follows. Section 2 presents the calculus μ*Stipula* with examples and the semantics. Section 3 contains the undecidability results for μ*Stipula*$^{\mathtt{TA}}$, μ*Stipula*$^{\mathtt{I}}$ and μ*Stipula*$^{\mathtt{D}}$. Section 4 contains the decidability results about μ*Stipula*$^{\mathtt{DI}}$. Section 5 reports and discusses related work and Section 6 presents general conclusions and future work. Appendix A highlights the relevance in a legal context of the fragments of μ*Stipula* studied in this paper. The proofs of our propositions, lemmas and theorems are reported in Appendix B.

This paper is a revised and extended version of [14] which includes more explanations of the results and the details of the (un)decidability proofs.

2 The calculus μ*Stipula*

μ*Stipula* is a calculus of contracts. A contract is declared by the term

$$\texttt{stipula C \{ init Q} \quad F \ \}$$

where `C` is the name of the contract, `Q` is the *initial state* and a F is a sequence of *functions*. We use a set of *states*, ranged over `Q`, `Q'`, $\cdots$; and a set of *function names* `f`, `g`, $\cdots$. The above contract is defined by the keyword `stipula` and is initially in the state specified by the `init` keyword. The syntax of functions F, events

W and time expressions `t` is the following:

Functions	F	::=	`_` | `@Q f { W } ⇒ @Q′ F`	
Events	W	::=	`_` | `t ≫ @Q ⇒ @Q′ W`	
Time expressions	`t`	::=	`now + k`	(`k` ∈ Nat)

Contracts transit from one state to another either by invoking a *function* or by running an *event*. Functions `@Q f { W } ⇒ @Q′` are invoked by the *external environment* and define the state `Q` when the invocation is admitted and the state `Q′` when the execution of `f` terminates.

Events W are sequences of *timed continuations* that are created by functions and schedule a transition in future execution. More precisely, the term `t ≫ @Q ⇒ @Q′` schedules a transition from `Q` to `Q′` at `t` time slot ahead the current time if the contract will be in the state `Q`. The time expressions are additions `now+k`, where `k` is a constant (a natural number representing, for example, *minutes*); `now` is a place-holder that will be replaced by 0 during the execution, see rule [Function] in Table 1. We always shorten `now + 0` into `now`.

Restriction and notations. We write `@Q f { W } ⇒ @Q′ ∈ C` when the function `@Q f { W } ⇒ @Q′` is in the contract `C`. Similarly for events. We assume that *a function is uniquely determined by the tuple* $\texttt{Q}\cdot\texttt{f}\cdot\texttt{Q}'$, that is the initial and final states and the function name. In the same way, an event is uniquely determined by the tuple $\texttt{Q}\cdot\texttt{ev}_\texttt{n}\cdot\texttt{Q}'$, where `n` is the line-code of the event.[1] Functions and events are generically called *clauses* and, since tuples $\texttt{Q}\cdot\texttt{f}\cdot\texttt{Q}'$ and $\texttt{Q}\cdot\texttt{ev}_\texttt{n}\cdot\texttt{Q}'$ uniquely identify functions and events, we will also call them clauses and write $\texttt{Q}\cdot\texttt{f}\cdot\texttt{Q}' \in \texttt{C}$ and $\texttt{Q}\cdot\texttt{ev}_\texttt{n}\cdot\texttt{Q}' \in \texttt{C}$.

2.1 Examples

As a first, simple example consider the `PingPong` contract:

```
1  stipula PingPong {
2      init Q0
3      @Q0 ping {
4          now + 1 ≫ @Q1 ⇒ @Q2
5      } ⇒ @Q1
6      @Q2 pong {
7          now + 2 ≫ @Q3 ⇒ @Q0
8      } ⇒ @Q3
9  }
```

The contract contains two functions: `ping` and `pong`. In particular `ping` is invoked if the contract is in the state `Q0`, `pong` when the contract is in `Q2`. Functions (i) make the contract transit in the state specified by the term "`⇒ @Q`" (see lines `5` and `8`) and (ii) make the events in their body to be scheduled. In particular, an event `now + k ≫ @Q ⇒ @Q′` (see lines `4` and `7`) is a timed continuation that can run when the time is `k` *time slots ahead to the clock value when the function is called* and the state of the contract is `Q`. The only effect of executing an event is the change of the state. When no event can be executed in a state either *the time progresses* (a tick occurs) or *a function is invoked*. The progression of time does not modify a state.

In the `PingPong` contract, the initial state is `Q0` where only `ping` may be invoked; no event is present because they are created by executing functions. The invocation of `ping` makes the contract transit to `Q1` and creates the event at line `4`, noted $\texttt{ev}_4$. In `Q1` there is still a unique possibility: executing $\texttt{ev}_4$. However, to execute it, it is necessary to wait `1` minute (assuming that the clock period is `1` minute) due to the time expression `now + 1`. Then the state becomes `Q2` indicating that `pong` may be invoked, thus letting the contract transit to `Q3` where, after `2` minutes (the expression `now + 2`), the event at line `7` can be executed and the contract returns to `Q0`. In `PingPong`, every clause is reachable.

The following `Sample` contract has an event that is unreachable:

```
1  stipula Sample {
2      init Init
3      @Init f {
4          now + 0 ≫ @Go ⇒ @End
5      } ⇒ @Run
6      @Init g { } ⇒ @Go
7  }
```

Let us discuss the issue. `Sample` has two functions at lines `3` and `6`, called `f` and `g`, respectively. The two functions may be invoked in `Init`, however the invocation of one of them excludes the other because their final states are not `Init`. Therefore the event at line `4`, which is inside `f`, is unreachable since it can run only if `g` is executed.

2.2 The operational semantics

The meaning of *μStipula* primitives is defined operationally by a transition relation between configurations. A configuration, ranged over by $\mathbb{C}$, $\mathbb{C}'$, $\cdots$, is a tuple $\texttt{C}(\texttt{Q}, \Sigma, \Psi)$ where

- `C` is the contract name;
- `Q` is the current state of the contract;
- Σ is either `_` or a term $\Psi \Rightarrow \texttt{Q}$. Σ represents either an empty body (hence, a clause can be executed or the time may progress) or a continuation where a set of events Ψ must be evaluated;

[1] We assume the code of *μStipula* contracts to be organized in lines of code, and each line contains at most one event definition.

– Ψ is a (possibly empty) multiset of *pending events* that have been already scheduled for future execution but not yet triggered. In particular, Ψ is either $\_$, when there are no pending events, or it is $\mathtt{k}_1 \gg_{\mathtt{n}_1} \mathtt{Q}_1 \Rightarrow \mathtt{Q}'_1 \mid \cdots \mid \mathtt{k}_h \gg_{\mathtt{n}_h} \mathtt{Q}_h \Rightarrow \mathtt{Q}'_h$ where "$\mid$" is commutative and associative with identity $\_$. In every term $\mathtt{k}_i \gg_{\mathtt{n}_i} \mathtt{Q}_i \Rightarrow \mathtt{Q}'_i$, the constant $\mathtt{k}_i$ is obtained from the time expression $\mathtt{t}_i$ of the corresponding event by dropping `now`. The index $\mathtt{n}_i$ is the *line-code* of the event.
The function that turns a *sequence of events* $\mathtt{now} + \mathtt{k} \gg \mathtt{@Q} \Rightarrow \mathtt{@Q}'$ into a *multiset of terms* $\mathtt{k} \gg_{\mathtt{n}} \mathtt{Q} \Rightarrow \mathtt{Q}'$ is $\mathtt{LC}_{\mathtt{Q}\cdot\mathtt{f}\cdot\mathtt{Q}'}(W)$ (see rule [Function], the trivial definition of this function is omitted). This function also drops the "`@`" from the states.

The transition relation of μ*Stipula* is $\mathbb{C} \xrightarrow{\mu} \mathbb{C}'$, where μ is either *empty* $\_$ or $\mathtt{f}$ or $\mathtt{ev}_{\mathtt{n}}$ (the label $\mathtt{ev}_{\mathtt{n}}$ indicates the event at line $\mathtt{n}$). The formal definition of $\mathbb{C} \xrightarrow{\mu} \mathbb{C}'$ is given in Table 1 using

– the predicate $\Psi, \mathtt{Q} \nrightarrow$, whose definition is

$$\Psi, \mathtt{Q} \nrightarrow \stackrel{\text{def}}{=} \begin{cases} \mathit{true} & \text{if } \Psi = \_ \\ \mathit{false} & \text{if } \Psi = 0 \gg_{\mathtt{n}} \mathtt{Q} \Rightarrow \mathtt{Q}' \mid \Psi' \\ \Psi', \mathtt{Q} \nrightarrow & \text{if } \Psi = \mathtt{k} \gg_{\mathtt{n}} \mathtt{Q}' \Rightarrow \mathtt{Q}'' \mid \Psi' \\ & \quad \text{and } (\mathtt{k} \neq 0 \text{ or } \mathtt{Q}' \neq \mathtt{Q}) \end{cases}$$

– the function $\Psi\downarrow$, whose definition is

$$\begin{aligned} (\Psi \mid \Psi')\downarrow &= \Psi\downarrow \mid \Psi'\downarrow \\ (\mathtt{k}+1 \gg_{\mathtt{n}} \mathtt{Q}' \Rightarrow \mathtt{Q}'')\downarrow &= \mathtt{k} \gg_{\mathtt{n}} \mathtt{Q}' \Rightarrow \mathtt{Q}'' \\ (0 \gg_{\mathtt{n}} \mathtt{Q}' \Rightarrow \mathtt{Q}'')\downarrow &= \_ \end{aligned}$$

A discussion about the four rules follows. Rule [Function] defines invocations: the label specifies the function name `f`. The transition may occur provided (i) the contract is in the state `Q` that admits invocations of `f` and (ii) no event can be triggered – *cf.* the premise $\Psi, \mathtt{Q} \nrightarrow$ (event's execution preempts function invocation). Rule [State-Change] says that a contract changes state by adding the sequence of events W to the multiset of pending events once `now` has been dropped from time expressions. Rule [Event-Match] specifies that an event handler may run provided Σ is $\_$, the time guard of the event has value 0 and the initial state of the event is the same of the contract's state. Rule [Tick] defines the progression of time. This happens when the contract has an empty Σ and no event can be triggered. In this case, the events with time value 0 are garbage-collected and the time values of the other events are decreased by one. The rules [Function] and [State-Change] might have been squeezed in one rule only. We have preferred to keep them apart for compatibility with *Stipula* (where functions' and events' bodies may also contain statements).

The *initial configuration* of a μ*Stipula* contract

$$\mathtt{stipula\ C\ \{\ init\ Q} \quad F\ \}$$

is $\mathbb{C}_{\mathtt{init}} = \mathtt{C}(\mathtt{Q}, \_, \_)$; the *set of configurations* of `C` are denoted by $\mathcal{C}_{\mathtt{C}}$.

We write $\mathbb{C} \longrightarrow \mathbb{C}'$ if there is a μ (as said above, μ may be either $\_$ or a function name or an event) such that $\mathbb{C} \xrightarrow{\mu} \mathbb{C}'$. We also write $\mathbb{C} \longrightarrow^* \mathbb{C}'$, called *computation*, if there are $\mu_1, \cdots, \mu_h$ such that $\mathbb{C} \xrightarrow{\mu_1} \cdots \xrightarrow{\mu_h} \mathbb{C}'$. Labels are useful in the examples (and proofs) to highlight the clauses that are executed. However, while in [25], they were introduced to ease the formal reasonings, labels may be overlooked in this work. Let $\mathbb{TS}(\mathtt{C}) = (\mathcal{C}_{\mathtt{C}}, \longrightarrow)$ be the transition system associated to `C`.

Remark 1 Few issues about the semantics in Table 1 are worth to be remarked.

– μ*Stipula* has three *causes for nondeterminism*: (i) two distinct functions can be invoked in a state, (ii) either a function is invoked or the time progresses (in this case the function may be invoked at a later time), and (iii) if two events may be executed at the same time, then one is chosen nondeterministically and is executed.
– Rule [Tick] defines the progression of time. This happens when the contract has no event to trigger. Henceforth, the complete execution of a function or of an event cannot last more than a single time unit. It is worth to notice that this semantics admits the paradoxical phenomenon that an endless sequence of function invocations does not make time progress (*cf.* the encoding of the Minsky machines in μ*Stipula*$^{\mathtt{I}}$). This paradoxical behaviour, which is also present in process calculi with time [22,29], might be removed by adjusting the semantics so to progress time when a maximal number of functions has been invoked. To ease the formal arguments we have preferred to stick to the simpler semantics.
– The semantics of μ*Stipula* in this paper is different from, yet equivalent to, [25,12]. In the literature, configurations have clock values that are incremented by the tick-rule. Then, in the [Function] rule, the variable `now` is replaced by the current clock value (and not dropped, as in our rule). We have chosen the current presentation because it eases the reasoning about expressivity.

To illustrate μ*Stipula* semantics, we discuss the computations of the `PingPong` contract. Consider $\mathtt{Ev}_4 = 1 \gg_4 \mathtt{Q1} \Rightarrow \mathtt{Q2}$ and $\mathtt{Ev}_7 = 2 \gg_7 \mathtt{Q3} \Rightarrow \mathtt{Q0}$. We write $\mathtt{Ev}_i{}^{(-j)}$ to indicate the $\mathtt{Ev}_i$ where the time guard has been decreased by j (time) units.

The contract may initially perform a number of [Tick] transitions, say k, and then a [Function] one. Therefore we have (on the right we write the rule that is used):

$$\frac{@\mathtt{Q}\ \mathtt{f}\{\,W\,\} \Rightarrow @\mathtt{Q}' \in \mathtt{C} \quad \Psi' = \mathtt{LC}_{\mathtt{Q}\cdot\mathtt{f}\cdot\mathtt{Q}'}(W) \quad \Psi, \mathtt{Q} \nrightarrow}{\mathtt{C}(\mathtt{Q}\,,\,\_\,,\,\Psi) \xrightarrow{\mathtt{f}} \mathtt{C}(\mathtt{Q}\,,\,\Psi' \Rightarrow \mathtt{Q}'\,,\,\Psi)}\ [\textsc{Function}]$$

$$\frac{\Psi = \mathtt{0} \gg_{\mathtt{n}} \mathtt{Q} \Rightarrow \mathtt{Q}' \mid \Psi'}{\mathtt{C}(\mathtt{Q}\,,\,\_\,,\,\Psi) \xrightarrow{\mathtt{ev_n}} \mathtt{C}(\mathtt{Q}\,,\,\_ \Rightarrow \mathtt{Q}'\,,\,\Psi')}\ [\textsc{Event-Match}]$$

$$\mathtt{C}(\mathtt{Q}\,,\,\Psi' \Rightarrow \mathtt{Q}'\,,\,\Psi) \longrightarrow \mathtt{C}(\mathtt{Q}'\,,\,\_\,,\,\Psi' \mid \Psi)\ [\textsc{State-Change}]$$

$$\frac{\Psi, \mathtt{Q} \nrightarrow}{\mathtt{C}(\mathtt{Q}\,,\,\_\,,\,\Psi) \longrightarrow \mathtt{C}(\mathtt{Q}\,,\,\_\,,\,\Psi\downarrow)}\ [\textsc{Tick}]$$

Table 1 The operational semantics of μ*Stipula*

$$\begin{array}{lll}
\mathtt{PingPong}(\mathtt{Q0}\,,\,_\,,\,_) & & \\
\quad \longrightarrow^{k} \mathtt{PingPong}(\mathtt{Q0}\,,\,_\,,\,_) & & [\textsc{Tick}] \\
\quad \xrightarrow{\mathtt{ping}} \mathtt{PingPong}(\mathtt{Q0}\,,\,\mathtt{Ev_4} \Rightarrow \mathtt{Q1}\,,\,_) & & [\textsc{Function}] \\
\quad \longrightarrow \mathtt{PingPong}(\mathtt{Q1}\,,\,_\,,\,\mathtt{Ev_4}) & & [\textsc{State-Change}] \\
\quad \longrightarrow \mathtt{PingPong}(\mathtt{Q1}\,,\,_\,,\,\mathtt{Ev_4}^{(-1)}) & & [\textsc{Tick}] \\
\quad \xrightarrow{\mathtt{ev_4}} \mathtt{PingPong}(\mathtt{Q1}\,,\,_ \Rightarrow \mathtt{Q2}\,,\,_) & & [\textsc{Event-Match}] \\
\quad \longrightarrow \mathtt{PingPong}(\mathtt{Q2}\,,\,_\,,\,_) & & [\textsc{State-Change}]
\end{array}$$

In $\mathtt{PingPong}(\mathtt{Q2}\,,\,\_\,,\,\_)$, the contract may perform a [Function] executing `pong`. Then the computation continues as follows:

$$\begin{array}{lll}
\xrightarrow{\mathtt{pong}} \mathtt{PingPong}(\mathtt{Q2}\,,\,\mathtt{Ev_7} \Rightarrow \mathtt{Q3}\,,\,_) & & [\textsc{Function}] \\
\longrightarrow \mathtt{PingPong}(\mathtt{Q3}\,,\,_\,,\,\mathtt{Ev_7}) & & [\textsc{State-Change}] \\
\longrightarrow^{2} \mathtt{PingPong}(\mathtt{Q3}\,,\,_\,,\,\mathtt{Ev_7}^{(-2)}) & & [\textsc{Tick}] \\
\xrightarrow{\mathtt{ev_7}} \mathtt{PingPong}(\mathtt{Q3}\,,\,_ \Rightarrow \mathtt{Q0}\,,\,_) & & [\textsc{Event-Match}] \\
\longrightarrow \mathtt{PingPong}(\mathtt{Q0}\,,\,_\,,\,_) & & [\textsc{State-Change}]
\end{array}$$

Definition 1 (State reachability)
Let $\mathtt{C}$ be a μ*Stipula* contract with initial configuration $\mathbb{C}_{\mathtt{init}}$. A state $\mathtt{Q}$ is reachable in $\mathtt{C}$ if and only if there exists a configuration $\mathtt{C}(\mathtt{Q}\,,\,\_\,,\,\Psi)$ such that $\mathbb{C}_{\mathtt{init}} \longrightarrow^{*} \mathtt{C}(\mathtt{Q}\,,\,\_\,,\,\Psi)$.

Our notion of *state reachability* closely relates to several well-established properties in the verification literature. In particular, it aligns with the concept of *control state reachability*, first introduced by Alur and Dill in the context of timed automata [7]. There, the problem is defined as determining whether, given an automaton A and a control state q, there exists a run of A that reaches q. Control state reachability has been extensively studied across various computational models. For instance, it has been examined in the setting of lossy Minsky machines [26], as well as in lossy FIFO channel systems [3,10]. In the framework of Petri nets, a related problem is the *coverability of a target marking*, as formulated in [24].

As discussed in the Introduction, a corresponding and central property in the context of μ*Stipula* is what we refer to as *clause reachability* as defined below.

Definition 2 (Clause reachability)
Let $\mathtt{C}$ be a μ*Stipula* contract with initial configuration $\mathbb{C}_{\mathtt{init}}$. A *clause* $\mathtt{Q}\cdot\mathtt{f}\cdot\mathtt{Q}'$ (or $\mathtt{Q}\cdot\mathtt{ev_n}\cdot\mathtt{Q}'$) in $\mathtt{C}$ is reachable if there exists a computation $\mathbb{C} \longrightarrow^{*} \mathbb{C}' \xrightarrow{\mu} \mathbb{C}''$ with $\mu = \mathtt{f}$ (or $\mu = \mathtt{ev_n}$).

Clause reachability and state reachability are closely related in the fragments of μ*Stipula* considered in this paper. Their equivalence is established in the next subsection.

2.3 Relevant sublanguages

We will consider the following fragments of μ*Stipula* whose relevance has been already discussed in the Introduction:

- $\mu\mathit{Stipula}^{\mathtt{I}}$, called instantaneous μ*Stipula*, is the fragment where the time expression of all the events is always `now + 0`;
- $\mu\mathit{Stipula}^{\mathtt{TA}}$, called time-ahead μ*Stipula*, is the fragment where every time expression of the events is `now + k`, with $\mathtt{k} > \mathtt{0}$;
- $\mu\mathit{Stipula}^{\mathtt{D}}$, called determinate μ*Stipula*, is the fragment where the sets of initial states of functions and of events have empty intersection; *i.e.*, for each function $@\mathtt{Q}\ \mathtt{f}\,\{\,W\,\} \Rightarrow @\mathtt{Q}'$ and event $\mathtt{t} \gg @\mathtt{Q}'' \Rightarrow @\mathtt{Q}'''$ in a contract, we impose $\mathtt{Q} \neq \mathtt{Q}''$;
- $\mu\mathit{Stipula}^{\mathtt{DI}}$, called determinate-instantaneous μ*Stipula*, is the intersection between $\mu\mathit{Stipula}^{\mathtt{D}}$ and $\mu\mathit{Stipula}^{\mathtt{I}}$; *i.e.*, for each function $@\mathtt{Q}\ \mathtt{f}\,\{\,W\,\} \Rightarrow @\mathtt{Q}'$ and event $\mathtt{t} \gg @\mathtt{Q}'' \Rightarrow @\mathtt{Q}'''$ in a contract, we impose $\mathtt{Q} \neq \mathtt{Q}''$ and $\mathtt{t} = \mathtt{now} + \mathtt{0}$.

The focus of this paper is about the (un)decidability of clause reachability in the above fragments, because our the ultimate goal is to investigate the possibility to implement a sound and complete unreachable-clause analyser. We now prove that clause reachability is decidable if and only if state reachability is decidable, for all the considered fragments. This will allow us to resort to state reachability that, as previously commented, is a more standard property in the literature.

Theorem 1 *Clause reachability is decidable in μStipula$^{\mathtt{I}}$ (respectively, μStipula$^{\mathtt{TA}}$, μStipula$^{\mathtt{D}}$ and μStipula$^{\mathtt{DI}}$) if and only if state reachability is decidable in μStipula$^{\mathtt{I}}$ (respectively, μStipula$^{\mathtt{TA}}$, μStipula$^{\mathtt{D}}$ and μStipula$^{\mathtt{DI}}$).*

Proof We simply show that, in all the considered fragments, each of the two problems can be reduced to the other one.

Consider a contract $\mathtt{C}$, expressed in any of the above four fragments of μStipula, and one of its clauses. Consider now a modified contract $\mathtt{C}'$ obtained from $\mathtt{C}$ by modifying the considered clause in such a way that its effect is that of entering a new fresh state $\mathtt{Q}'$ (non present in $\mathtt{C}$). Such contract $\mathtt{C}'$ belongs to the same fragment of $\mathtt{C}$ and, moreover, we have that the considered clause is reachable in $\mathtt{C}$ if and only if the state $\mathtt{Q}'$ is reachable in $\mathtt{C}'$.

Consider now a contract $\mathtt{C}$, expressed in any of the above four fragments of μStipula, and one of its states $\mathtt{Q}$. It is not restrictive to consider $\mathtt{Q}$ different from the initial state, otherwise state reachability becomes a trivial property that always holds. Consider now the set S of clauses in $\mathtt{C}$ whose effect is that of entering in state $\mathtt{Q}$. We have that $\mathtt{Q}$ is reachable in $\mathtt{C}$ if and only if at least one of the clauses in S is reachable. Notice that the set S is finite, hence checking the reachability of one clause in S corresponds to the conjunction of finitely many instances of the clause reachability problem. □

3 Undecidability results

To show the undecidability of state reachability, we rely on a reduction technique from a Turing-complete computational model to μStipula$^{\mathtt{I}}$, μStipula$^{\mathtt{TA}}$, and μStipula$^{\mathtt{D}}$. As Turing-powerful formalism we consider Minsky machines [28]. A Minsky machine is an automaton with two registers $\mathtt{R}_1$ and $\mathtt{R}_2$ holding arbitrary large natural numbers, a finite set of states $\mathtt{Q}, \mathtt{Q}', \cdots$, and a program $\mathtt{P}$ consisting of a finite sequence of numbered instructions of the following type:

- $\mathtt{Q} : \mathit{Inc}(\mathtt{R}_i, \mathtt{Q}')$: in the state $\mathtt{Q}$, increment $\mathtt{R}_i$ and go to the state $\mathtt{Q}'$;
- $\mathtt{Q} : \mathit{DecJump}(\mathtt{R}_i, \mathtt{Q}', \mathtt{Q}'')$: in the state $\mathtt{Q}$, if the content of $\mathtt{R}_i$ is zero then go to the state $\mathtt{Q}'$, else decrease $\mathtt{R}_i$ by 1 and go to the state $\mathtt{Q}''$.

A configuration of a Minsky machine is given by a tuple $(\mathtt{Q}, v_1, v_2)$ where $\mathtt{Q}$ indicates the state of the machine and v_1 and v_2 are the contents of the two registers. A transition of a Minsky machine is denoted by $\longrightarrow_{\mathtt{M}}$. We assume that the machine has an *initial state* $\mathtt{Q}_0$ and a *final state* $\mathtt{Q}_F$ that has no instruction starting at it. The *halting problem* of a Minsky machine is assessing whether there exist v_1, v_2 such that $(\mathtt{Q}_F, v_1, v_2)$ is reachable starting from $(\mathtt{Q}_0, 0, 0)$. This problem is undecidable [28]. In the rest of the section, we will demonstrate the undecidability of state reachability for μStipula$^{\mathtt{I}}$, μStipula$^{\mathtt{TA}}$, and μStipula$^{\mathtt{D}}$ by providing encodings of Minsky machines. The encodings we use are increasingly complex. Therefore, we will discuss the undecidability results starting from the simplest one. Here, we present the Minsky machines encodings and informally discuss their correctness while the details of the proofs can be found in Appendix B.

3.1 Undecidability for μStipula$^{\mathtt{I}}$

Table 2 defines the encoding of a Minsky machine M into a μStipula$^{\mathtt{I}}$ contract $\mathtt{I}_M$. The relevant invariant of the encoding is that, every time M transits to $(\mathtt{Q}, v_1, v_2)$ then $\mathtt{I}_M$ may transit to $\mathtt{I}_M(\mathtt{Q}, \_, \Psi)$, where the number of events

$$0 \gg \mathtt{@dec}_1 \Rightarrow \mathtt{@ackdec}_1 \quad \text{and} \quad 0 \gg \mathtt{@dec}_2 \Rightarrow \mathtt{@ackdec}_2$$

in Ψ are v_1 and v_2, respectively. Additionally, a transition $(\mathtt{Q}, v_1, v_2) \longrightarrow_{\mathtt{M}} (\mathtt{Q}', v_1', v_2')$ corresponds to a sequence of transitions $\mathtt{I}_M(\mathtt{Q}, \_, \Psi) \longrightarrow^* \mathtt{I}_M(\mathtt{Q}', \_, \Psi')$ with either (i) a `fincQ` function, if the Minsky machine performs an *Inc* instruction, or (ii) either a `fdecQ` or a `fzeroQ` function, if the Minsky machine performs a *DecJump* instruction. In particular, the function `fincQ` has the ability to add one instance of the event $0 \gg \mathtt{@dec}_1 \Rightarrow \mathtt{@ackdec}_1$ (or $0 \gg \mathtt{@dec}_2 \Rightarrow \mathtt{@ackdec}_2$). The function `fdecQ` has the effect of consuming one instance of the event $0 \gg \mathtt{@dec}_1 \Rightarrow \mathtt{@ackdec}_1$ (resp. $0 \gg \mathtt{@dec}_2 \Rightarrow \mathtt{@ackdec}_2$), by entering the state $\mathtt{@dec}_1$ (resp. $\mathtt{@dec}_2$) which triggers such event. Also the function `zeroQ` enters in one of the states $\mathtt{@dec}_i$, but in this case the computation will have the ability to continue only if the state $\mathtt{@zero}_i$ will be reached (this because the produced event $0 \gg \mathtt{@zero}_i \Rightarrow \mathtt{@aQ}$ must be triggered to allow the computation to continue). But $\mathtt{@zero}_i$ can be reached only if no event $0 \gg \mathtt{@dec}_i$ is present, because only in this case the function $\mathtt{fdec}_i$ can be invoked (remember that events have priority w.r.t. function invocations).

We finally observe that the transitions $\mathtt{I}_M(\mathtt{Q}, \_, \Psi) \longrightarrow^* \mathtt{I}_M(\mathtt{Q}', \_, \Psi')$ which mimick the Minsky machine step $(\mathtt{Q}, v_1, v_2) \longrightarrow_{\mathtt{M}} (\mathtt{Q}', v_1', v_2')$ are not the unique possibile transitions. Nevertheless, in case different alternative transitions are executed, the contract $\mathtt{I}_M$ will have no longer the possibility to reach a state corresponding to a Minsky machine state, thus the simulation of the machine gets stuck. One of the alternative transitions is the invocation of the function `fdecQ` when

Let M be a Minsky machine with initial state $\mathtt{Q}_0$. Let $\mathtt{I}_M$ be the μ*Stipula*$^{\mathtt{I}}$ contract

$$\texttt{stipula I}_M \ \{\ \texttt{init Start} \quad F_M\ \}$$

where F_M contains the functions

- $\mathtt{@Start\ fstart\ \{\ now} \gg \mathtt{\ @aQ}_0 \Rightarrow \mathtt{\ @bQ}_0\ \} \Rightarrow \mathtt{@Q}_0$
- for every instruction $\mathtt{Q} : \mathit{Inc}(\mathtt{R}_i, \mathtt{Q}')$, with $i \in \{1, 2\}$:

$$\begin{array}{l}\mathtt{@Q\ fincQ\ \{\ now} \gg \mathtt{@dec}_i \Rightarrow \mathtt{@ackdec}_i \\ \qquad\qquad \mathtt{now} \gg \mathtt{@bQ} \Rightarrow \mathtt{@Q}' \\ \qquad\qquad \mathtt{now} \gg \mathtt{@aQ}' \Rightarrow \mathtt{@bQ}' \\ \}\Rightarrow \mathtt{@aQ}\end{array}$$

- for every instruction $\mathtt{Q} : \mathit{DecJump}(\mathtt{R}_i, \mathtt{Q}', \mathtt{Q}'')$, with $i \in \{1, 2\}$:

$$\begin{array}{l}\mathtt{@Q\ fdecQ\ \{\ now} \gg \mathtt{@ackdec}_i \Rightarrow \mathtt{@aQ} \\ \qquad\qquad \mathtt{now} \gg \mathtt{@bQ} \Rightarrow \mathtt{@Q}'' \\ \qquad\qquad \mathtt{now} \gg \mathtt{@aQ}'' \Rightarrow \mathtt{@bQ}'' \\ \}\Rightarrow \mathtt{@dec}_i\end{array} \qquad \begin{array}{l}\mathtt{@Q\ fzeroQ\ \{\ now} \gg \mathtt{@zero}_i \Rightarrow \mathtt{@aQ} \\ \qquad\qquad \mathtt{now} \gg \mathtt{@bQ} \Rightarrow \mathtt{@Q}' \\ \qquad\qquad \mathtt{now} \gg \mathtt{@aQ}' \Rightarrow \mathtt{@bQ}' \\ \}\Rightarrow \mathtt{@dec}_i\end{array}$$

- $\mathtt{@dec}_1\ \mathtt{fdec1}\ \{\ \}\Rightarrow \mathtt{@zero}_1$ and $\mathtt{@dec}_2\ \mathtt{fdec2}\ \{\ \}\Rightarrow \mathtt{@zero}_2$

Table 2 The μ*Stipula*$^{\mathtt{I}}$ contract modelling a Minsky machine

the corresponding register is empty. In this case, the simulation gets stuck because the state $\mathtt{@ackdec}_i$, necessary to trigger the event $0 \gg \mathtt{@ackdec}_i \Rightarrow \mathtt{@aQ}$, cannot be reached. Similarly, if `fzeroQ` is invoked when the corresponding register is nonempty the state $\mathtt{@zero}_i$, necessary to trigger the event $0 \gg \mathtt{@zero}_i \Rightarrow \mathtt{@aQ}$, cannot be reached. Also the elapsing of time is problematic because both the events $0 \gg \mathtt{@dec}_1 \Rightarrow \mathtt{@ackdec}_1$ and $0 \gg \mathtt{@dec}_2 \Rightarrow \mathtt{@ackdec}_2$, which model the content of the registers, are erased by a [Tick] transition. This could corrupt the modeling of the registers. In this case the simulation gets stuck because also the event $0 \gg \mathtt{aQ} \Rightarrow \mathtt{bQ}$ is erased, which is necessary to model the transition from a state `Q` of the Minsky machine to the next one.

Theorem 2 *State reachability is undecidable for the fragment* μ*Stipula*$^{\mathtt{I}}$.

3.2 Undecidability for μ*Stipula*$^{\mathtt{TA}}$

Also in this case we reduce from the halting problem of Minsky machines to state reachability in μ*Stipula*$^{\mathtt{TA}}$. The encoding of a machine M is defined in Table 3. In this case, states of the μ*Stipula*$^{\mathtt{TA}}$ contract alternates between "*machine states*" occurring, say, at even time clocks, and "*management states*" occurring at odd time clocks. For this reason we add erroneous transitions at even time clocks from management states to `end` (a state without outgoing transitions) and at odd time clocks from machine states to `end`. Similarly to Table 2, a unit in the register i is encoded by an event $\mathtt{now} + 1 \gg \mathtt{@dec}_i \Rightarrow \mathtt{@ackdec}_i$ (assuming to be in a machine state). Therefore the instruction $\mathtt{Q} : \mathit{Inc}(\mathtt{R}_i, \mathtt{Q}')$, which occurs in a machine state `Q`, amounts to adding to the next time-clock such event and the erroneous event $\mathtt{now} + 1 \gg \mathtt{@Q}' \Rightarrow \mathtt{@end}$ that makes the contract transit to `end` if it is still in $\mathtt{Q}'$ at the beginning of the next time-clock.

The encoding of $\mathtt{Q} : \mathit{DecJump}(\mathtt{R}_i, \mathtt{Q}', \mathtt{Q}'')$ is more convoluted because we have to move all the events $\mathtt{now} + 1 \gg \mathtt{@dec}_i \Rightarrow \mathtt{@ackdec}_i$ ahead two clock units (except one, if the corresponding register value is positive). Assume an invocation of `fdecQ` occurs and $i = 1$. Then a bunch of events are created (see the body of `fdecQ` in Table 3) and the contract transits into `wait`. In this state, a [Tick] transition can occur; hence the time values of the events are decreased by one. Then $0 \gg \mathtt{wait} \Rightarrow \mathtt{dec}_1$ is enabled and the protocol responsible for moving ahead all the events $0 \gg \mathtt{dec}_1 \Rightarrow \mathtt{ackdec}_1$ (except one) and all the events $0 \gg \mathtt{dec}_2 \Rightarrow \mathtt{ackdec}_2$ starts. The protocol works as follows:

1. since the state is $\mathtt{dec}_1$, $0 \gg \mathtt{dec}_1 \Rightarrow \mathtt{ackdec}_1$ is fired (assume the value of R_1 is positive) and the contract transits to $\mathtt{ackdec}_1$;
2. in $\mathtt{ackdec}_1$, $0 \gg \mathtt{ackdec}_1 \Rightarrow \mathtt{nextQ''}_1$ is fired and the contract transits to state $\mathtt{nextQ''}_1$ (one of the events $0 \gg \mathtt{dec}_1 \Rightarrow \mathtt{ackdec}_1$ has been erased without being moved ahead);
3. in $\mathtt{nextQ''}_1$, the function

$$\mathtt{@nextQ''}_1\ \ \mathtt{fnextQ''\_1}\ \{\ \mathtt{now}{+}1 \gg \mathtt{@next} \Rightarrow \mathtt{@Q''}\} \Rightarrow \mathtt{@dec}_1$$

can be invoked. A transition to $\mathtt{dec}_1$ happens and the event $1 \gg \mathtt{next} \Rightarrow \mathtt{Q''}$ is created. When the transfer protocol terminates, this event will make the contract transit to $\mathtt{Q''}$;

Let M be a Minsky machine with initial state $\mathtt{Q}_0$. Let $\mathtt{TA}_M$ be the $\mu\mathit{Stipula}^{\mathtt{TA}}$ contract

$$\mathtt{stipula}\ \mathtt{TA}_M\ \{\ \mathtt{init}\ \mathtt{Q}_0 \quad F_M\ \}$$

with F_M containing the functions

– for every instruction $\mathtt{Q} : \mathit{Inc}(\mathtt{R}_i, \mathtt{Q}')$, with $i \in \{1, 2\}$:

$$\begin{array}{l}\mathtt{@Q\ fincQ}\ \{\ \mathtt{now}+1 \gg \mathtt{@dec}_i \Rightarrow \mathtt{@ackdec}_i\\ \qquad\qquad \mathtt{now}+1 \gg \mathtt{@Q}' \Rightarrow \mathtt{@end}\\ \}\Rightarrow \mathtt{@Q}'\end{array}$$

– for every instruction $\mathtt{Q} : \mathit{DecJump}(\mathtt{R}_i, \mathtt{Q}', \mathtt{Q}'')$, with $i \in \{1, 2\}$:

$$\begin{array}{l}\mathtt{@Q\ fdecQ}\ \{\ \mathtt{now}+1 \gg \mathtt{@ackdec}_i \Rightarrow \mathtt{@nextQ''}_i\\ \qquad\qquad \mathtt{now}+1 \gg \mathtt{@wait} \Rightarrow \mathtt{@dec}_1\\ \qquad\qquad \mathtt{now}+2 \gg \mathtt{@dec}_1 \Rightarrow \mathtt{@end}\\ \qquad\qquad \mathtt{now}+2 \gg \mathtt{@dec}_2 \Rightarrow \mathtt{@end}\\ \qquad\qquad \mathtt{now}+2 \gg \mathtt{@nextQ''}_i \Rightarrow \mathtt{@end}\\ \qquad\qquad \mathtt{now}+2 \gg \mathtt{@ackdec}_1 \Rightarrow \mathtt{@end}\\ \qquad\qquad \mathtt{now}+2 \gg \mathtt{@ackdec}_2 \Rightarrow \mathtt{@end}\\ \qquad\qquad \mathtt{now}+3 \gg \mathtt{@Q''} \Rightarrow \mathtt{@end}\\ \}\Rightarrow \mathtt{@wait}\end{array}$$

$$\begin{array}{l}\mathtt{@Q\ fzeroQ}\ \{\ \mathtt{now}+1 \gg \mathtt{@ackdec}_i \Rightarrow \mathtt{@end}\\ \qquad\qquad \mathtt{now}+2 \gg \mathtt{@next} \Rightarrow \mathtt{@Q}'\\ \qquad\qquad \mathtt{now}+1 \gg \mathtt{@wait} \Rightarrow \mathtt{@dec}_1\\ \qquad\qquad \mathtt{now}+3 \gg \mathtt{@Q}' \Rightarrow \mathtt{@end}\\ \}\Rightarrow \mathtt{@wait}\end{array}$$

– for $i \in \{1, 2\}$ and for every state $\mathtt{Q}$ of M, we have the following functions:

$$\mathtt{@wait\ fwait}\ \{\ \}\ \Rightarrow\ \mathtt{@end} \qquad \mathtt{@dec}_1\ \mathtt{fdec1}\ \{\ \}\ \Rightarrow\ \mathtt{@dec}_2 \qquad \mathtt{@dec}_2\ \mathtt{fdec2}\ \{\ \}\ \Rightarrow\ \mathtt{@next}$$

$$\mathtt{@ackdec}_i\ \mathtt{fackdec\_i}\ \{\ \mathtt{now}+2 \gg \mathtt{@dec}_i \Rightarrow \mathtt{@ackdec}_i\}\ \Rightarrow\ \mathtt{@dec}_i \qquad \mathtt{@nextQ}_i\ \mathtt{fnextQ\_i}\ \{\ \mathtt{now}+1 \gg \mathtt{@next} \Rightarrow \mathtt{@Q}\}\ \Rightarrow\ \mathtt{@dec}_i$$

Table 3 The $\mu\mathit{Stipula}^{\mathtt{TA}}$ contract modelling a Minsky machine

4. at this stage the transfer of events $0 \gg \mathtt{dec}_i \Rightarrow \mathtt{ackdec}_i$ occurs. At first the protocol moves all the events $0 \gg \mathtt{dec}_1 \Rightarrow \mathtt{ackdec}_1$ (every such event is fired and then the function $\mathtt{fackdec}_1$ that recreates the same event at $\mathtt{now}+2$ is executed) then the function $\mathtt{fdec}_1$ is invoked and the same protocol is applied to the events $0 \gg \mathtt{dec}_2 \Rightarrow \mathtt{ackdec}_2$;
5. at the end of the transfers, the function $\mathtt{fdec}_2$ is invoked and the contract transits to `next`;
6. in `next`, no event nor function can be executed, therefore a [Tick] occurs and the time values of the events are decreased by one (in particular those $2 \gg \mathtt{dec}_i \Rightarrow \mathtt{ackdec}_i$ that were transferred at step 4). Then $0 \gg \mathtt{next} \Rightarrow \mathtt{Q}''$ that was created at step 3 is executed and the contract transits to $\mathtt{Q}''$.

When the register R_1 is 0 and `fdecQ` is invoked then the event at step 2 cannot be produced and the computation is fated to reach an `end` state (by `fdec1`, `fdec2` or by [tick] and then performing $0 \gg \mathtt{dec}_1 \Rightarrow \mathtt{end}$). If, on the contrary, the invoked function is `fzeroQ`, the same protocol as above is used to transfer the events $0 \gg \mathtt{dec}_i \Rightarrow \mathtt{ackdec}_i$ (in this case with $i = 2$ only) and the contract reaches the step 5 where, after a [Tick], the event $\mathtt{now} + 2 \gg \mathtt{next} \Rightarrow \mathtt{Q}'$ can be executed. The undecidability result for $\mu\mathit{Stipula}^{\mathtt{TA}}$ follows.

Theorem 3 *State reachability is undecidable for the fragment* $\mu\mathit{Stipula}^{\mathtt{TA}}$.

3.3 Undecidability for $\mu\mathit{Stipula}^{\mathtt{D}}$

Also in this case we reduce from the halting problem of Minsky machines to state reachability in $\mu\mathit{Stipula}^{\mathtt{D}}$. In $\mu\mathit{Stipula}^{\mathtt{D}}$, functions and events start in different states. Therefore the encoding of Table 3 is inadequate since we used the expedient that events preempt functions when enabled in the same state to make the contract transit to the `end` state (which indicates an error). For $\mu\mathit{Stipula}^{\mathtt{D}}$ we need to refine the sequence *machine-management states* in order to have extra management over erroneous operations (decrease of zero register or zero-test of a positive register). The idea is to manage at different times the events $\mathtt{dec}_1 \Rightarrow \mathtt{ackdec}_1$ and $\mathtt{dec}_2 \Rightarrow \mathtt{ackdec}_2$ that model registers' units. In particular, if the contract is in a (machine) state $\mathtt{Q}$ at time 0 then $\mathtt{dec}_1 \Rightarrow \mathtt{ackdec}_1$ are at time 1 and $\mathtt{dec}_2 \Rightarrow \mathtt{ackdec}_2$ are at time 3. The states at times 2 and 4 perform management operations. Therefore the sequence of states becomes

machine-state $\rightarrow$ *transfer1-state* $\rightarrow$ *management1-state* $\rightarrow$ *transfer2-state* $\rightarrow$ *management2-state*

where every state is one tick ahead the previous one. The states *transfer1-state* and *transfer2-state* manage the transfer of all the events $\mathtt{dec}_1 \Rightarrow \mathtt{ackdec}_1$ and of the events $\mathtt{dec}_2 \Rightarrow \mathtt{ackdec}_2$ ahead five clock units.

Let M be a Minsky machine with initial state $\mathtt{Q}_0$. Let $\mathtt{D}_M$ be the μ*Stipula*$^{\mathsf{D}}$ contract

$$\texttt{stipula D}_M \ \{\ \texttt{init Start} \quad F_M\ \}$$

with F_M contains the functions

– `@Start fstart { now ≫ @notickA ⇒ @cont } ⇒ @Q`$_0$

– for every $\mathtt{Q} : Inc(R_1, \mathtt{Q}')$:

```
@Q fAincQ {                          @Q fBincQ {
    now + 1 ≫ @dec1 ⇒ @ackdec1           now + 1 ≫ @dec1 ⇒ @ackdec1
    now ≫ @cont ⇒ @Q'                    now ≫ @cont ⇒ @Q'
    now ≫ @notickB ⇒ @cont               now ≫ @notickA ⇒ @cont
} ⇒ @notickA                         } ⇒ @notickB
```

– for every $\mathtt{Q} : Inc(R_2, \mathtt{Q}')$:

```
@Q fAincQ {                          @Q fBincQ {
    now + 3 ≫ @dec2 ⇒ @ackdec2           now + 3 ≫ @dec2 ⇒ @ackdec2
    now ≫ @cont ⇒ @Q'                    now ≫ @cont ⇒ @Q'
    now ≫ @notickB ⇒ @cont               now ≫ @notickA ⇒ @cont
} ⇒ @notickA                         } ⇒ @notickB
```

– for every $\mathtt{Q} : DecJump(R_1, \mathtt{Q}', \mathtt{Q}'')$:

```
@Q fAdecQ {                              @Q fBdecQ {
    now + 1 ≫ @ackdec1 ⇒ @Q''_start1         now + 1 ≫ @ackdec1 ⇒ @Q''_start1
    now + 1 ≫ @s1notick ⇒ @cont              now + 1 ≫ @s1notick ⇒ @cont
    now ≫ @cont ⇒ @dec1                      now ≫ @cont ⇒ @dec1
} ⇒ @notickA                             } ⇒ @notickB

@Q fAzeroQ {                             @Q fBzeroQ {
    now ≫ @cont ⇒ @dec1                      now ≫ @cont ⇒ @dec1
    now + 2 ≫ @dec1 ⇒ @dec2                  now + 2 ≫ @dec1 ⇒ @dec2
    now + 3 ≫ @ackdec2 ⇒ @copy2              now + 3 ≫ @ackdec2 ⇒ @copy2
    now + 3 ≫ @c2notickA ⇒ @cont             now + 3 ≫ @c2notickA ⇒ @cont
    now + 5 ≫ @dec2 ⇒ @Q'                    now + 5 ≫ @dec2 ⇒ @Q'
    now + 5 ≫ @notickB ⇒ @cont               now + 5 ≫ @notickA ⇒ @cont
} ⇒ @notickA                             } ⇒ @notickB
```

– for every $\mathtt{Q} : DecJump(R_2, \mathtt{Q}', \mathtt{Q}'')$:

```
@Q fAdecQ {                              @Q fBdecQ {
    now ≫ @cont ⇒ @dec1                      now ≫ @cont ⇒ @dec1
    now + 1 ≫ @ackdec1 ⇒ @copy1              now + 1 ≫ @ackdec1 ⇒ @copy1
    now + 1 ≫ @c1notickA ⇒ @cont             now + 1 ≫ @c1notickA ⇒ @cont
    now + 2 ≫ @dec1 ⇒ @dec2                  now + 2 ≫ @dec1 ⇒ @dec2
    now + 3 ≫ @ackdec2 ⇒ @Q''_start2         now + 3 ≫ @ackdec2 ⇒ @Q''_start2
    now + 3 ≫ @s2notick ⇒ @cont              now + 3 ≫ @s2notick ⇒ @cont
} ⇒ @notickA                             } ⇒ @notickB

@Q fAzeroQ {                             @Q fBzeroQ {
    now ≫ @cont ⇒ @dec1                      now ≫ @cont ⇒ @dec1
    now + 1 ≫ @ackdec1 ⇒ @copy1              now + 1 ≫ @ackdec1 ⇒ @copy1
    now + 1 ≫ @c1notickA ⇒ @cont             now + 1 ≫ @c1notickA ⇒ @cont
    now + 2 ≫ @dec1 ⇒ @dec2                  now + 2 ≫ @dec1 ⇒ @dec2
    now + 5 ≫ @dec2 ⇒ @Q'                    now + 5 ≫ @dec2 ⇒ @Q'
    now + 5 ≫ @notickB ⇒ @cont               now + 5 ≫ @notickA ⇒ @cont
} ⇒ @notickA                             } ⇒ @notickB
```

– the management functions in Table 5.

Table 4 The μ*Stipula*$^{\mathsf{D}}$ contract modelling a Minsky machine

```
@Q_start1 fQstart1 {
    now ≫ @cont ⇒ @dec1
    now ≫ @ackdec1 ⇒ @copy1
    now ≫ @c1notickA ⇒ @cont
    now + 1 ≫ @dec1 ⇒ @dec2
    now + 2 ≫ @ackdec2 ⇒ @copy2
    now + 2 ≫ @c2notickA ⇒ @cont
    now + 4 ≫ @dec2 ⇒ @Q
    now + 4 ≫ @notickA ⇒ @cont
} ⇒ @s1notick
```

```
@Q_start2 fQstart2 {
    now ≫ @cont ⇒ @dec2
    now ≫ @ackdec2 ⇒ @copy2
    now ≫ @c2notickA ⇒ @cont
    now + 2 ≫ @dec2 ⇒ @Q
    now + 2 ≫ @notickA ⇒ @cont
} ⇒ @s2notick
```

```
@copy1 fAcopy1 {
    now ≫ @cont ⇒ @dec1
    now ≫ @ackdec1 ⇒ @copy1
    now ≫ @c1notickB ⇒ @cont
    now + 5 ≫ @dec1 ⇒ @ackdec1
} ⇒ @c1notickA
```

```
@copy2 fAcopy2 {
    now ≫ @cont ⇒ @dec2
    now ≫ @ackdec2 ⇒ @copy2
    now ≫ @c2notickB ⇒ @cont
    now + 5 ≫ @dec2 ⇒ @ackdec2
} ⇒ @c2notickA
```

```
@copy1 fBcopy1 {
    now ≫ @cont ⇒ @dec1
    now ≫ @ackdec1 ⇒ @copy1
    now ≫ @c1notickA ⇒ @cont
    now + 5 ≫ @dec1 ⇒ @ackdec1
} ⇒ @c1notickB
```

```
@copy2 fBcopy2 {
    now ≫ @cont ⇒ @dec2
    now ≫ @ackdec2 ⇒ @copy2
    now ≫ @c2notickA ⇒ @cont
    now + 5 ≫ @dec2 ⇒ @ackdec2
} ⇒ @c2notickB
```

Table 5 The management functions of Table 4 (Q is a state of the Minsky machine)

Clearly, misplaced [Tick] transitions may break the rigidity of the protocol. This means that it is necessary to stop the simulation if a wrong [Tick] transition is performed. We already used a similar mechanism in Table 2. In that case, a management event at time 0 (that is created in the past transition) is necessary to simulate the Minsky machine transition; in turn, the simulation creates a similar management event for the next one. Henceforth, if a tick occurs before the invocation of a function (thus erasing registers' values that were events at time 0, as well) then the simulation stops because the management event is also erased.

A similar expedient cannot be used for the μ*Stipula*$^{\mathtt{D}}$ encoding because the registers' values are at different times (+1 and +3 with respect to the machine state) and erasing the management event with a tick may be useless if the μ*Stipula*$^{\mathtt{D}}$ function produces an equal management event, which is the case when the corresponding transition is circular (initial and final states are the same). Therefore we refine the technique in Table 2 by adding sibling functions and the simulation uses standard functions or sibling ones according to the presence of the management event $0 \gg \mathtt{notickA} \Rightarrow \mathtt{cont}$ or of $0 \gg \mathtt{notickB} \Rightarrow \mathtt{cont}$. For example, the encoding of $\mathtt{Q} : \mathit{Inc}(R_1, \mathtt{Q}')$ is (the events of register R_1 are at $\mathtt{now} + 1$):

```
@Q : fAincQ {
    now + 1 ≫ @dec1 ⇒ @ackdec1
    now ≫ @cont ⇒ @Q′
    now ≫ @notickB ⇒ @cont
} ⇒ @notickA

@Q : fBincQ {
    now + 1 ≫ @dec1 ⇒ @ackdec1
    now ≫ @cont ⇒ @Q′
    now ≫ @notickA ⇒ @cont
} ⇒ @notickB
```

Assuming to be in a configuration with state Q and event $0 \gg \mathtt{notickA} \Rightarrow \mathtt{cont}$, the unique function that can be invoked is fAincQ; thereafter, the combined effect of $0 \gg \mathtt{notickA} \Rightarrow \mathtt{cont}$ and $0 \gg \mathtt{cont} \Rightarrow \mathtt{Q}'$ allows the contract to transit to $\mathtt{Q}'$ with an additional event $1 \gg \mathtt{dec}_1 \Rightarrow \mathtt{ackdec}_1$ (corresponding to a register increment) and the presence of $0 \gg \mathtt{notickB} \Rightarrow \mathtt{cont}$ that compels the next instruction, if any, to be a sibling one (*e.g.* $\mathtt{fBincQ}'$).

Tables 4 and 5 define the encoding of a Minsky machine M into a μ*Stipula*$^{\mathtt{D}}$ contract $\mathtt{D}_M$. The reader may notice that the management functions fQstart1 and fQstart2 in Table 5 do not have sibling functions because they always produce a management event $\mathtt{k} \gg \mathtt{notickA} \Rightarrow \mathtt{cont}$ (with k be either 2 or 4). Actually this is an optimization: they are invoked because a decrement occurred and, in such cases it is not possible that the foregoing management events are used in the simulation of the current instruction. The undecidability result for μ*Stipula*$^{\mathtt{D}}$ follows.

Theorem 4 *State reachability is undecidable for the fragment* μ*Stipula*$^{\mathtt{D}}$.

4 Decidability for μ*Stipula*$^{\mathtt{DI}}$

We demonstrate that state reachability is decidable for μ*Stipula*$^{\mathtt{DI}}$. As for the previous section, here we present our proof techique and informally discuss its correctness, while the details of the proofs can be found in Appendix B.

We first recall that, in μ*Stipula*$^{\mathtt{DI}}$, for every pair of clauses $\mathtt{Q{\cdot}f{\cdot}Q}'$ and $\mathtt{Q}''\mathtt{{\cdot}ev{\cdot}Q}'''$ in C, we have $\mathtt{Q} \neq \mathtt{Q}''$. In our decidability proof, we resort to a variant of the semantics of μ*Stipula*$^{\mathtt{DI}}$ with an alternative [Tick] rule.

Let $\texttt{InitEv}(\texttt{C})$ be the set of initial states of events in $\texttt{C}$, where $\texttt{C}$ is a $\mu\mathit{Stipula}$ contract. Let

$$\frac{\texttt{Q} \notin \texttt{InitEv}(\texttt{C})}{\texttt{C}(\texttt{Q}, \_, \Psi) \longrightarrow \texttt{C}(\texttt{Q}, \_, \Psi\downarrow)} \text{ [Tick-Plus]}$$

That is, unlike [Tick], [Tick-Plus] may only be used in states that are not initial states of events. Let $\mu\mathit{Stipula}^{\texttt{DI}}_{+}$ be the language whose operational semantics uses [Tick-Plus] instead of [Tick]; we denote with $\longrightarrow_{\texttt{tp}}$ the transition relation of $\mu\mathit{Stipula}^{\texttt{DI}}_{+}$. We observe that, syntactically, nothing is changed: every $\mu\mathit{Stipula}^{\texttt{DI}}$ contract is a $\mu\mathit{Stipula}^{\texttt{DI}}_{+}$ contract and conversely. We denote by $\mathbb{TS}_{\texttt{tp}}(\texttt{C}) = (\mathcal{C}_{\texttt{C}}, \longrightarrow_{\texttt{tp}})$ the transitions system associated to contract $\texttt{C}$ using $\longrightarrow_{\texttt{tp}}$ as transition rule.

Definition 3 Let $\mathbb{C}$ be a possible configuration of a $\mu\mathit{Stipula}$ contract. We say that $\mathbb{C}$ is *stuck* if, for every computation $\mathbb{C} \longrightarrow^* \mathbb{C}'$ the transitions therein are always instances of [Tick].

Proposition 1 *Let* $\texttt{C}(\texttt{Q}, \Sigma, \Psi)$ *be a configuration of a* $\mu\mathit{Stipula}^{\texttt{DI}}$ *contract* $\texttt{C}$ *(or a* $\mu\mathit{Stipula}^{\texttt{DI}}_{+}$ *contract). Then*

(i) whenever $\texttt{Q} \notin \texttt{InitEv}(\texttt{C})$ *or* $\Sigma \neq \_$*:*

$$\texttt{C}(\texttt{Q}, \Sigma, \Psi) \longrightarrow \mathbb{C} \text{ if and only if } \texttt{C}(\texttt{Q}, \Sigma, \Psi) \longrightarrow_{\texttt{tp}} \mathbb{C}\ ;$$

(ii) whenever $\texttt{Q} \in \texttt{InitEv}(\texttt{C})$ *and* $\Psi = 0 \gg_{\texttt{n}} \texttt{Q} \Rightarrow \texttt{Q}' \mid \Psi'$*:*

$$\texttt{C}(\texttt{Q}, \_, \Psi) \longrightarrow \mathbb{C} \text{ if and only if } \texttt{C}(\texttt{Q}, \_, \Psi) \longrightarrow_{\texttt{tp}} \mathbb{C}\ ;$$

(iii) whenever $\texttt{Q} \in \texttt{InitEv}(\texttt{C})$ *and* $\Psi, \texttt{Q} \not\rightarrow$ *:*

$$\texttt{C}(\texttt{Q}, \_, \Psi) \text{ is stuck if and only if } \texttt{C}(\texttt{Q}, \_, \Psi) \not\rightarrow_{\texttt{tp}}.$$

A consequence of Proposition 1 is that a state $\texttt{Q}$ is reachable in $\mu\mathit{Stipula}^{\texttt{DI}}$ if and only if it is reachable in $\mu\mathit{Stipula}^{\texttt{DI}}_{+}$. This allows us to safely reduce state reachability arguments to $\mu\mathit{Stipula}^{\texttt{DI}}_{+}$. In particular we demonstrate that $\mathbb{TS}_{\texttt{tp}}(\texttt{C})$, where $\texttt{C}$ is a $\mu\mathit{Stipula}^{\texttt{DI}}_{+}$ contract, is a well-structured transition system.

We begin with some background on well-structured transition systems [19]. A relation $\leq \subseteq X \times X$ is called *quasi-ordering* if it is reflexive and transitive. A *well-quasi-ordering* is a quasi-ordering $\leq \subseteq X \times X$ such that, for every infinite sequence $x_1, x_2, x_3, \cdots$, there exist $i < j$ with $x_i \leq x_j$.

Definition 4 A *well-structured transition system* is a tuple $(\mathcal{C}, \longrightarrow, \preceq)$ where $(\mathcal{C}, \longrightarrow)$ is a transition system and $\preceq \subseteq \mathcal{C} \times \mathcal{C}$ is a quasi-ordering such that:

(1) $\preceq$ is a well-quasi-ordering
(2) $\preceq$ is upward compatible with $\longrightarrow$, i.e., for every $\mathbb{C}_1, \mathbb{C}_1', \mathbb{C}_2 \in \mathcal{C}$ such that $\mathbb{C}_1 \preceq \mathbb{C}_1'$ and $\mathbb{C}_1 \longrightarrow \mathbb{C}_2$ there exists $\mathbb{C}_2'$ in $\mathcal{C}$ verifying $\mathbb{C}_1' \longrightarrow^* \mathbb{C}_2'$ and $\mathbb{C}_2 \preceq \mathbb{C}_2'$

Given a configuration $\mathbb{C}$ of a well-structured transition system, $\mathit{Pred}(\mathbb{C})$ denotes the set of immediate predecessors of $\mathbb{C}$ (*i.e.*, $\mathit{Pred}(\mathbb{C}) = \{\mathbb{C}' \mid \mathbb{C}' \longrightarrow \mathbb{C}\}$) while $\uparrow \mathbb{C}$ denotes the set of configurations greater than $\mathbb{C}$ (*i.e.*, $\uparrow \mathbb{C} = \{\mathbb{C}' \mid \mathbb{C} \preceq \mathbb{C}'\}$). With abuse of notation, we will denote with $\mathit{Pred}(\cdot)$ and $\uparrow\cdot$ also their natural extension to sets of configurations. A *basis* of an upward-closed set of configurations $\mathcal{D} \subseteq \mathcal{C}$ is a set $\mathcal{D}^\flat$ such that $\mathcal{D} = \cup_{\mathbb{C} \in \mathcal{D}^\flat} \uparrow \mathbb{C}$. We know that every upward-closed set of a well-quasi-ordering admits a finite basis [19].

Several properties are decidable for well-structured transition systems (under some conditions discussed below) [1,19], we will consider the following one.

Definition 5 Let $(\mathcal{C}, \longrightarrow, \preceq)$ be a well-structured transition system. The *coverability problem* is to decide, given the initial configuration $\mathbb{C}_{\texttt{init}} \in \mathcal{C}$ and a target configuration $\mathbb{C} \in \mathcal{C}$, whether there exists a configuration $\mathbb{C}' \in \mathcal{C}$ such that $\mathbb{C} \preceq \mathbb{C}'$ and $\mathbb{C}_{\texttt{init}} \longrightarrow^* \mathbb{C}'$.

In well-structured transition systems the coverability problem is decidable when the transition relation $\longrightarrow$, the ordering $\preceq$ and, for every $\mathbb{C} \in \mathcal{C}$, a finite-basis for $\uparrow \mathit{Pred}(\uparrow \mathbb{C})$ are effectively computable.

Let us now define the relation $\preceq$ as the least quasi-ordering relation such that $\Psi \preceq \Psi \mid \Psi'$ for every Ψ'. The relation $\preceq$ is lifted to configurations as follows

$$\texttt{C}(\texttt{Q}, \Sigma, \Psi) \preceq \texttt{C}(\texttt{Q}, \Sigma, \Psi') \quad \text{if} \quad \Psi \preceq \Psi'\ .$$

It is worth to observe that, according to the relation $\preceq$, the state reachability problem for $\texttt{Q}$ is equivalent to the coverability problem for the initial configuration $\mathbb{C}_{\texttt{init}}$ and the target configuration $\texttt{C}(\texttt{Q}, \_, \_)$.

Lemma 1 *For a* $\mu\mathit{Stipula}^{\texttt{DI}}_{+}$ *contract* $\texttt{C}$*,* $(\mathcal{C}_{\texttt{C}}, \longrightarrow_{\texttt{tp}}, \preceq)$ *is a well-structured transition system.*

It is worth to notice that Lemma 1 does not hold for $\mu\mathit{Stipula}^{\texttt{DI}}$ because $\longrightarrow$ is not upward compatible with $\preceq$. In fact, while $\texttt{C}(\texttt{Q}, \_, \_) \longrightarrow \texttt{C}(\texttt{Q}, \_, \_)$ with a rule [Tick] and $\texttt{C}(\texttt{Q}, \_, \_) \preceq \texttt{C}(\texttt{Q}, \_, 0 \gg_{\texttt{n}} \texttt{Q} \Rightarrow \texttt{Q}')$, the unique computation of $\texttt{C}(\texttt{Q}, \_, 0 \gg_{\texttt{n}} \texttt{Q} \Rightarrow \texttt{Q}')$ when $\texttt{Q}' \in \texttt{InitEv}(\texttt{C})$ is (we recall that, in $\mu\mathit{Stipula}$, events preempt function invocations and ticks):

$$\texttt{C}(\texttt{Q}, \_, 0 \gg_{\texttt{n}} \texttt{Q} \Rightarrow \texttt{Q}') \longrightarrow \texttt{C}(\texttt{Q}, \_ \Rightarrow \texttt{Q}', \_) \longrightarrow \texttt{C}(\texttt{Q}', \_, \_)$$

and then it gets stuck. Therefore no configuration $\mathbb{C}$ is reachable such that $\texttt{C}(\texttt{Q}, \_, \_) \preceq \mathbb{C}$.

Lemma 2 *Let* $(\mathcal{C}, \longrightarrow_{\texttt{tp}}, \preceq)$ *be the well-structured transition system of a* $\mu\mathit{Stipula}^{\texttt{DI}}_{+}$ *contract. Then,* $\longrightarrow_{\texttt{tp}}$ *and* $\preceq$ *are decidable and there exists an algorithm for computing a finite basis of* $\uparrow \mathit{Pred}(\uparrow \mathbb{C})$ *for any* $\mathbb{C} \in \mathcal{C}$*.*

From Lemma 2 and the above mentioned results, on well-structured transition systems we get the following result.

Theorem 5 *The state reachability problem is decidable for the fragment* $\mu\mathit{Stipula}_{+}^{\mathtt{DI}}$.

As a direct consequence of Proposition 1 and Theorem 5 we have:

Corollary 1 *The state reachability problem is decidable for the fragment* $\mu\mathit{Stipula}^{\mathtt{DI}}$.

5 Related work

The decidability of problems about infinite-state systems has been largely addressed in the literature. We refer to [1] for an overview of the research area.

It turns out that critical features of $\mu\mathit{Stipula}$, such as garbage-collecting elapsed events or preempting events with respect to functions and progression of time may be modelled by variants of Petri nets with inhibitor and reset arcs. While standard Petri nets, which are infinite-state systems, have decidable problems of reachability, coverability, boundedness, etc. (see, e.g., [17]), the above variants of Petri nets are Turing complete, therefore all non-trivial properties become undecidable [16].

It is not obvious whether Petri nets with inhibitor and reset arcs may be modelled by $\mu\mathit{Stipula}$ contracts. It seems that these features have different expressive powers than time progression and events. Hence, other formalisms, such as pi-calculus and actor languages, might have a stricter correspondence with $\mu\mathit{Stipula}$. As regards the decidability of problems in pi-calculus, we recall the decidability of reachability and termination in the depth-bounded fragment of the pi-calculus [27] and the decidability of the reachability problem for various fragments of the asynchronous pi-calculus that feature name generation, name mobility, and unbounded control [8]. Regarding actor languages, in [13], we demonstrated the decidability of termination for stateless actors (actors without fields) and for actors with states when the number of actors is bounded and the state is read-only. It is worth to observe that all these results have been achieved by using techniques that are similar to those used in this paper: either demonstrating that the model of the calculus is a well-structured transition system [19] (for which, under certain computability conditions, the reachability and termination problems are decidable, see Section 4) or simulating a Turing complete model, such as the Minsky machines, into the calculus under analysis (hence the undecidability results of problems such as termination).

The $\mu\mathit{Stipula}$ calculus has some similarities with formal models of timed systems such as timed automata [7]. The control state reachability problem, namely, given a timed automaton A and a control state q, does there exist a run of A that visits q, is known to be decidable [7]. A similar result holds for Timed Networks (TN) [5,4], a formal model consisting of a family of timed automata with a distinct *controller* defined as a finite-state automaton without clocks. Each process in a TN can communicate with all other processes via rendezvous messages. Control state reachability is also decidable for Timed Networks with transfer actions [2]. A transfer action forces all processes in a given state to move to a successor state as transfer arcs in Petri nets, which allows to move all tokens contained in a certain place to another [20]. $\mu\mathit{Stipula}$ can also be seen as a language for modelling asynchronous programs in which callbacks are scheduled using timers. Verification problems for formal models of (untimed) asynchronous programs have been considered in [30,23]. In this context, Boolean program execution is modeled using a pushdown automaton, while asynchronous calls are modeled by adding a multiset of pending callbacks to the global state of the program. Callbacks are only executed when the program stack is empty. Verification of safety properties is decidable for this model via a non-trivial reduction to coverability of Petri nets [21].

6 Conclusions

We have systematically studied the computational power of $\mu\mathit{Stipula}$, a basic calculus defining legal contracts. The calculus is stateful and features clauses that may be either functions to be invoked by the external environment or events that can be executed at certain time slots. We have demonstrated that in several legally relevant fragments of $\mu\mathit{Stipula}$ a problem such as reachability of state is undecidable. The decidable fragment, $\mu\mathit{Stipula}^{\mathtt{DI}}$, is the one whose event and functions start in disjoint states and where events are instantaneous (the time expressions are 0).

We conclude by indicating some relevant line for future research. First of all, the decidability result for $\mu\mathit{Stipula}^{\mathtt{DI}}$ leaves open the question about the complexity of the state reachability problem in that fragment. Our current conjecture is that the problem is EXPSPACE-complete and we plan to prove this conjecture by reducing the coverability problem for Petri nets into the state reachability problem for $\mu\mathit{Stipula}^{\mathtt{DI}}$. Another interesting line of research regards the investigation of sound, but incomplete, algorithms for checking state reachability in $\mu\mathit{Stipula}$. A preliminary algorithm was investigated in [25]. The presented algorithm spots clauses that are

unreachable in μ*Stipula* contracts and is not tailored to any particular fragment of the language. The results in this paper show that this algorithm may be improved to achieve completeness when the input contract complies with the μ*Stipula*$^{\mathtt{DI}}$ constraints.

References

1. Parosh Aziz Abdulla, Karlis Cerans, Bengt Jonsson, and Yih-Kuen Tsay. General decidability theorems for infinite-state systems. In *Proc. 11th Annual IEEE Symposium on Logic in Computer Science*, pages 313–321. IEEE Computer Society, 1996.
2. Parosh Aziz Abdulla, Giorgio Delzanno, Othmane Rezine, Arnaud Sangnier, and Riccardo Traverso. Parameterized verification of time-sensitive models of ad hoc network protocols. *Theor. Comput. Sci.*, 612:1–22, 2016.
3. Parosh Aziz Abdulla and Bengt Jonsson. Verifying programs with unreliable channels. In *Proceedings of the Eighth Annual Symposium on Logic in Computer Science (LICS '93), Montreal, Canada, June 19-23, 1993*, pages 160–170. IEEE Computer Society, 1993.
4. Parosh Aziz Abdulla and Bengt Jonsson. Model checking of systems with many identical timed processes. *TCS*, 290(1):241–264, 2003.
5. Parosh Aziz Abdulla and Aletta Nylén. Timed petri nets and bqos. In *ICATPN'01*, volume 2075 of *LNCS*, pages 53–70. Springer, 2001.
6. Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools (2nd Edition)*. Addison Wesley, Boston, MA, USA, 2006.
7. Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.
8. Roberto M. Amadio and Charles Meyssonnier. On decidability of the control reachability problem in the asynchronous pi-calculus. *Nordic J. of Computing*, 9(2):70–101, 2002.
9. Andrew W. Appel and Maia Ginsburg. *Modern Compiler Implementation in C*. Cambridge University Press, Cambridge, UK, 1997.
10. Gérard Cécé, Alain Finkel, and S. Purushothaman Iyer. Unreliable channels are easier to verify than perfect channels. *Inf. Comput.*, 124(1):20–31, 1996.
11. Silvia Crafa and Cosimo Laneve. Programming legal contracts - A beginners guide to *Stipula*. In *The Logic of Software. A Tasting Menu of Formal Methods - Essays Dedicated to Reiner Hähnle on the Occasion of His 60th Birthday*, volume 13360 of *Lecture Notes in Computer Science*, pages 129–146, Cham, Switzerland, 2022. Springer.
12. Silvia Crafa, Cosimo Laneve, Giovanni Sartor, and Adele Veschetti. Pacta sunt servanda: Legal contracts in *Stipula*. *Sci. Comput. Program.*, 225:102911, 2023.
13. Frank S. de Boer, Mohammad Mahdi Jaghoori, Cosimo Laneve, and Gianluigi Zavattaro. Decidability problems for actor systems. *Log. Methods Comput. Sci.*, 10(4), 2014.
14. Giorgio Delzanno, Cosimo Laneve, Arnaud Sangnier, and Gianluigi Zavattaro. Decidability problems for micro-stipula. In Cinzia Di Giusto and António Ravara, editors, *Coordination Models and Languages - 27th IFIP WG 6.1 International Conference, COORDINATION 2025, Held as Part of the 20th International Federated Conference on Distributed Computing Techniques, DisCoTec 2025, Lille, France, June 17-19, 2025, Proceedings*, volume 15731 of *Lecture Notes in Computer Science*, pages 133–152. Springer, 2025.
15. Leonard Eugene Dickson. Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors. *Bull. Amer. Math. Soc.*, 19(6):285, 1913.
16. Catherine Dufourd, Alain Finkel, and Philippe Schnoebelen. Reset nets between decidability and undecidability. In Kim Guldstrand Larsen, Sven Skyum, and Glynn Winskel, editors, *Automata, Languages and Programming, 25th International Colloquium, ICALP'98, Aalborg, Denmark, July 13-17, 1998, Proceedings*, volume 1443 of *Lecture Notes in Computer Science*, pages 103–115. Springer, 1998.
17. Javier Esparza and Mogens Nielsen. Decidability issues for petri nets - a survey. *J. Inf. Process. Cybern.*, 30(3):143–160, 1994.
18. Samuele Evangelisti. Stipula Reachability Analyzer, February 2024. Available on github: `https://github.com/stipula-language`.
19. A. Finkel and Ph. Schnoebelen. Well-structured transition systems everywhere! *Theor. Comput. Sci.*, 256:63–92, 2001.
20. Alain Finkel, Jean-Franccois Raskin, Mathias Samuelides, and Laurent Van Begin. Monotonic extensions of petri nets: Forward and backward search revisited. *Electr. Notes Theor. Comput. Sci.*, 68(6):85–106, 2002.
21. Pierre Ganty and Rupak Majumdar. Algorithmic verification of asynchronous programs. *ACM Trans. Program. Lang. Syst.*, 34(1):6:1–6:48, 2012.
22. Hans Hansson and Bengt Jonsson. A calculus for communicating systems with time and probabitilies. In *Proceedings of the Real-Time Systems Symposium - 1990*, pages 278–287, New York, USA, 1990. IEEE Computer Society.
23. Ranjit Jhala and Rupak Majumdar. Interprocedural analysis of asynchronous programs. In Martin Hofmann and Matthias Felleisen, editors, *Proceedings of the 34th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2007, Nice, France, January 17-19, 2007*, pages 339–350. ACM, 2007.
24. Richard M. Karp and Raymond E. Miller. Parallel program schemata. *J. Comput. Syst. Sci.*, 3(2):147–195, 1969.
25. Cosimo Laneve. Reachability analysis in Micro-Stipula. In *Proc. 26th International Symposium on Principles and Practice of Declarative Programming, PPDP 2024*, pages 17:1–17:12. ACM, 2024.
26. Richard Mayr. Undecidability of weak bisimulation equivalence for 1-counter processes. In Jos C. M. Baeten, Jan Karel Lenstra, Joachim Parrow, and Gerhard J. Woeginger, editors, *Automata, Languages and Programming, 30th International Colloquium, ICALP 2003, Eindhoven, The Netherlands, June 30 - July 4, 2003. Proceedings*, volume 2719 of *Lecture Notes in Computer Science*, pages 570–583. Springer, 2003.
27. Roland Meyer. On boundedness in depth in the pi-calculus. In *IFIP TCS*, volume 273 of *IFIP*, pages 477–489. Springer, 2008.
28. Marvin Minsky. *Computation: finite and infinite machines*. Prentice Hall, 1967.
29. Faron Moller and Chris M. N. Tofts. A temporal calculus of communicating systems. In *Proceedings of CONCUR '90*, volume 458 of *Lecture Notes in Computer Science*, pages 401–415, Berlin Heidelberg, Germany, 1990. Springer.
30. Koushik Sen and Mahesh Viswanathan. Model checking multithreaded programs with asynchronous atomic

methods. In Thomas Ball and Robert B. Jones, editors, *Computer Aided Verification, 18th International Conference, CAV 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, volume 4144 of *Lecture Notes in Computer Science*, pages 300–314. Springer, 2006.

A Legal relevance of *μStipula* and its fragments

To illustrate the relevance in a legal context of the fragments of *μStipula* identified in the Introduction, we refer to simple legal contracts and employ a subset of operations from the full *Stipula* language. We begin with a legal contract for bike rentals; it consists of three articles:

1. This Agreement shall commence when the Borrower takes possession of the Bike and remain in full force and effect until the Bike is returned to Lender. The Borrower shall return the Bike `k` hours after the rental and will pay Euro `cost` in advance where half of the amount is of surcharge for late return.
2. Payment. Borrower shall pay the amount specified in Article 1 when this agreement commences.
3. End. If the Bike is returned within the agreed time specified in Article 1, then half of the `cost` will be returned to Borrower; the other half is given to the Lender. Otherwise the full `cost` is given to the Lender.

These articles are transposed into *μStipula* using a few extensions to the calculus – operations that are almost standard and will be briefly discussed below.

```
1  stipula Bike_Rental {
2    assets wallet, bike
3    init @Inactive
4    @Inactive Lender : offer[b] {
5      b —∘ bike // the access code of the bike is stored
6                // in the contract
7    } ⇛ @Payment
8    @Payment Borrower : pay[x]
9      (x == cost) {
10       x —∘ wallet // the contract keeps the money
11       bike —∘ Borrower // the Borrower keeps
12                        // the bike access code
13       now + k ≫ @Using {
14           wallet —∘ Lender // deadline expired: the
15                            // whole wallet to Lender
16       } ⇛ @End
17   } ⇛ @Using
18   @Using Borrower : end { // bike returned before
19                           // the deadline
20     0.5 × wallet —∘ wallet, Lender // half wallet
21                                    // to Lender
22     wallet —∘ Borrower // half wallet to Borrower
23   } ⇛ @End
24 }
```

Listing 1 The bike-rental contract

The contract `Bike_Rental` has two *assets*: `wallet` and `bike` that will contain money and the access code of a bike, respectively. The contract starts when the Lender offers an access code `b` of a bike (line 4) that is stored in the asset `bike` – the operation `b —∘ bike` at line 5 (this is Article 1 in the legal contract). Then the Borrower can pay – the function at line 8 – which amounts to store in the `wallet` the payment `x` that has been made (this is Article 2). At the same time the bike code is sent to the Borrower (the operation `bike —∘ Borrower` at line 11), so the Borrower can use the bike, and event is scheduled at time `now + k` (lines 13-16): if the bike is not returned at that time, the Borrower has to pay the whole amount in the `wallet` as specified in Article 3. The function at lines 18-23 defines the timely return of the bike by the Borrower (Article 3; notice that the operation `0.5 × wallet —∘ wallet, Lender` halves the content of the `wallet`; therefore the following operation `wallet —∘ Borrower` sends to the Borrower the remaining half).

A substantial problem in the legal contract domain is spotting normative clauses, either functions or events, that

will be never applied. In fact, it is possible, that an ureachable clause is considered too oppressive by one of the parties (take, for instance, the event of the foregoing Article 3), thus making the legal relationship fail. To address this problem, we have defined μ*Stipula*, a sub-calculus of *Stipula* that allows us to focus on the control flow of a contract. For example, you get a μ*Stipula* contract by dropping all the operations $\multimap$, the parameters of the functions and the asset declarations from Listing 1. In particular, the resulting contract belongs to the fragment $\mu\mathit{Stipula}^{\mathtt{TA}}$ because `k` (the renting time) is positive.

There are other fragments of μ*Stipula* that are also relevant. Consider the following alternative bike rental contract, in which the Borrower may keep the bike indefinitely, but becomes eligible for a discount on the price if the bike is returned.

1. This Agreement shall commence when the Borrower takes possession of the Bike and remain in full force and effect until the Bike is returned to Lender. The Borrower shall return the Bike at his discretion and will pay Euro `cost` in advance.
2. Payment. Borrower shall pay the amount `cost` as specified in Article 1 when this agreement commences.
3. End. If the Bike is returned then 90% of `cost` will be given to Lender and 10% of `cost` will be returned to Borrower.
4. Problems and resolutions. If the Lender detects a problem, he may trigger a resolution process that is managed by the Authority. The Authority will immediately enforce resolution to either the Lender or the Borrower.

(To keep things simple, we have omitted the details of Article 4.) The alternative contract is similar to that of Listing 1 upto line 7.

```
1  stipula alt_Bike_Rental {
2    assets wallet, bike
3    init @Inactive
4    @Inactive Lender : offer[b] {
5      b —o bike // the bike access code is stored
6                // in the contract
7    } ⇒ @Payment
8    @Payment Borrower : pay[x]
9      (x >= cost) {
10       x —o wallet // the contract keeps the money
11       bike —o Borrower // the Borrower keeps the bike
12                        // access code
13   } ⇒ @Using
14   @Using Borrower : end { // bike returned
15     0.9 × wallet —o wallet, Lender // 90% wallet to
16                                    // Lender
17     wallet —o Borrower // 10% wallet to Borrower
18   } ⇒ @End
19   @Using Lender : problems(u) {
20         // communicate the problems u to Authority
21         // that will manage the issue
22   } ⇒ @Manage
23   @Manage Authority : resolution (v) {
24     if (v == 0) {
25        now ≫ @Problem {
26         // immediately enforce resolution to the Lender
27        } ⇒ @PbLender
28     } else { // there is no management to do
29        now ≫ @Problem { // immediately enforce
30                         // resolution to the Borrower
31        } ⇒ @PbBorrower
32     }
33   } ⇒ @Problem
```

Listing 2 The alternative bike-rental contract

The key lines of Listing 2 are 23-32 where the Authority enforces an immediate resolution in favour of the Lender or of the Borrower. In these cases, the events are respectively `now ≫ @Problem { } ⇒ @PbLender` and `now ≫ @Problem { } ⇒ @PbBorrower`. Since the time expressions are always `now`, the contract belongs to the fragment $\mu\mathit{Stipula}^{\mathtt{I}}$. Actually, it also belongs to $\mu\mathit{Stipula}^{\mathtt{D}}$ because the initial state `Problem` of the two events is different from `Inactive`, `Payment`, `Using` and `Manage`, which are the initial state of the functions. As such, the contract of Listing 2 fits with the constraints of $\mu\mathit{Stipula}^{\mathtt{DI}}$.

B Proofs

B.1 Proofs of Section 3

Theorem 2 *State reachability is undecidable for the fragment* $\mu\mathit{Stipula}^{\mathtt{I}}$.

Proof Let M be a Minsky machine with states $\{\mathtt{Q}_0, \cdots, \mathtt{Q}_n\}$ where $\mathtt{Q}_0$ is the initial state. Let $\mathtt{I}_M$ be the encoding in $\mu\mathit{Stipula}^{\mathtt{I}}$ as defined in Table 2 and let $\mathtt{S}, \mathtt{S}', \cdots$ range over states of $\mathtt{I}_M$. The initial state of $\mathtt{I}_M$ is $\mathtt{I}_M(\mathtt{Start}, \_, \_)$ and, after a sequence of transitions [Tick], the contract $\mathtt{I}_M$ performs the transitions

$$\mathtt{I}_M(\mathtt{Start}, \_, \_) \xrightarrow{\mathtt{fstart}} \mathtt{I}_M(\mathtt{Start}, Ev \Rightarrow \mathtt{Q}_0, \_) \longrightarrow \mathtt{I}_M(\mathtt{Q}_0, \_, Ev)$$

where $Ev = 0 \gg \mathtt{aQ}_0 \Rightarrow \mathtt{bQ}_0$. Next, let

$$[\![v_1, v_2]\!]^{\mathtt{I}}_{\mathtt{Q}} = \underbrace{0 \gg \mathtt{dec}_1 \Rightarrow \mathtt{ackdec}_1}_{v_1 \text{ times}} \mid \underbrace{0 \gg \mathtt{dec}_2 \Rightarrow \mathtt{ackdec}_2}_{v_2 \text{ times}} \mid 0 \gg \mathtt{aQ} \Rightarrow \mathtt{bQ}\,.$$

We demonstrate that

(1) $(\mathtt{Q}_i, v_1, v_2) \longrightarrow_{\mathtt{M}} (\mathtt{Q}_j, v_1', v_2')$ implies $\mathtt{I}_M(\mathtt{Q}_i, \_, [\![v_1, v_2]\!]^{\mathtt{I}}_{\mathtt{Q}_i}) \longrightarrow^* \mathtt{I}_M(\mathtt{Q}_j, \_, [\![v_1', v_2']\!]^{\mathtt{I}}_{\mathtt{Q}_j})$

(2) $\mathtt{I}_M$ does not transit to unreachable Minsky's states. That is, if $\mathtt{I}_M(\mathtt{Q}_i, \_, [\![v_1, v_2]\!]^{\mathtt{I}}_{\mathtt{Q}_i}) \longrightarrow^* \mathtt{I}_M(\mathtt{S}, \Sigma, \Psi) \longrightarrow \mathtt{I}_M(\mathtt{Q}_j, \Sigma', \Psi')$ with $\mathtt{S} \notin \{\mathtt{Q}_0, \cdots, \mathtt{Q}_n\}$ then $\Sigma' = \_$ and $\Psi' = [\![v_1', v_2']\!]^{\mathtt{I}}_{\mathtt{Q}_j}$ and $(\mathtt{Q}_i, v_1, v_2) \longrightarrow^*_{\mathtt{M}} (\mathtt{Q}_j, v_1', v_2')$.

The proof of (1) is a case analysis on the type of Minsky instructions. We omit the simulation of the *Inc* instruction, which is simple, and discuss in detail the simulation of *DecJump*. There are two cases (we assume the register is R_1):

$R_1 > 0$: therefore M performs the transition $(\mathtt{Q}_i, v_1, v_2) \longrightarrow_{\mathtt{M}} (\mathtt{Q}_j, v_1 - 1, v_2)$. Correspondingly, in $\mathtt{I}_M$, an invocation of the function $\mathtt{fdecQ}_i$ occurs. Then the contract performs the transitions

$$\begin{aligned}\mathtt{I}_M(\mathtt{Q}_i, \_, [\![v_1, v_2]\!]^{\mathtt{I}}_{\mathtt{Q}_i}) &\xrightarrow{\mathtt{fdecQ_i}} \mathtt{I}_M(\mathtt{Q}_i, E_1 | E_2 | E_3 \Rightarrow \mathtt{dec}_1, [\![v_1, v_2]\!]^{\mathtt{I}}_{\mathtt{Q}_i}) \\ &\longrightarrow \mathtt{I}_M(\mathtt{dec}_1, \_, [\![v_1, v_2]\!]^{\mathtt{I}}_{\mathtt{Q}_i} | E_1 | E_2 | E_3)\end{aligned}$$

where $E_1 = 0 \gg \mathtt{ackdec}_1 \Rightarrow \mathtt{aQ_i}$, $E_2 = 0 \gg \mathtt{bQ_i} \Rightarrow \mathtt{Q_j}$, and $E_3 = 0 \gg \mathtt{aQ_j} \Rightarrow \mathtt{bQ_j}$. Since $v_1 > 0$, there is at least one event $0 \gg \mathtt{@dec}_1 \Rightarrow \mathtt{ackdec}_1$ in $[\![v_1, v_2]\!]^{\mathtt{I}}_{\mathtt{Q}_i}$. Then

$$\begin{aligned}&\mathtt{I}_M(\mathtt{dec}_1, \_, [\![v_1, v_2]\!]^{\mathtt{I}}_{\mathtt{Q}_i} | E_1 | E_2 | E_3) \\ &\quad\longrightarrow \mathtt{I}_M(\mathtt{dec}_1, \_ \Rightarrow \mathtt{ackdec}_1, [\![v_1 - 1, v_2]\!]^{\mathtt{I}}_{\mathtt{Q}_i} | E_1 | E_2 | E_3) \\ &\quad\longrightarrow \mathtt{I}_M(\mathtt{ackdec}_1, \_, [\![v_1 - 1, v_2]\!]^{\mathtt{I}}_{\mathtt{Q}_i} | E_1 | E_2 | E_3) \\ &\quad\longrightarrow \mathtt{I}_M(\mathtt{ackdec}_1, \_ \Rightarrow \mathtt{aQ_i}, [\![v_1 - 1, v_2]\!]^{\mathtt{I}}_{\mathtt{Q}_i} | E_2 | E_3) \\ &\quad\longrightarrow \mathtt{I}_M(\mathtt{aQ_i}, \_, [\![v_1 - 1, v_2]\!]^{\mathtt{I}}_{\mathtt{Q}_i} | E_2 | E_3) \\ &\quad\longrightarrow \mathtt{I}_M(\mathtt{aQ_i}, \_ \Rightarrow \mathtt{bQ_i}, [\![v_1 - 1, v_2]\!]^{\mathtt{I}}_{\mathtt{Q}_j} | E_2) \\ &\quad\longrightarrow \mathtt{I}_M(\mathtt{bQ_i}, \_, [\![v_1 - 1, v_2]\!]^{\mathtt{I}}_{\mathtt{Q}_j} | E_2) \\ &\quad\longrightarrow \mathtt{I}_M(\mathtt{bQ_i}, \_ \Rightarrow \mathtt{Q}_j, [\![v_1 - 1, v_2]\!]^{\mathtt{I}}_{\mathtt{Q}_j}) \\ &\quad\longrightarrow \mathtt{I}_M(\mathtt{Q}_j, \_, [\![v_1 - 1, v_2]\!]^{\mathtt{I}}_{\mathtt{Q}_j})\end{aligned}$$

$R_1 = 0$: therefore M performs the transition $(\mathtt{Q}_i, 0, v_2) \longrightarrow_{\mathtt{M}} (\mathtt{Q}_j, 0, v_2)$. Correspondingly, in $\mathtt{I}_M$, an invocation of the

function $\texttt{fzeroQ}_i$ occurs. Then the contract performs the transitions

$$\texttt{I}_M(\texttt{Q}_i, \_, [\![0, v_2]\!]^{\texttt{I}}_{\texttt{Q}_i}) \xrightarrow{\texttt{fzeroQ}_i} \texttt{I}_M(\texttt{Q}_i, E_1' | E_2 | E_3 \Rightarrow \texttt{dec}_1, [\![v_1, v_2]\!]^{\texttt{I}}_{\texttt{Q}_i}) \longrightarrow \texttt{I}_M(\texttt{dec}_1, \_, [\![0, v_2]\!]^{\texttt{I}}_{\texttt{Q}_i} | E_1' | E_2 | E_3)$$

where $E_1' = 0 \gg \texttt{zero}_1 \Rightarrow \texttt{aQ}_\texttt{i}$ (E_2 and E_3 are as above). Since $v_1 = 0$, there is no event $0 \gg \texttt{@dec}_1 \Rightarrow \texttt{ackdec}_1$ in $[\![v_1, v_2]\!]^{\texttt{I}}_{\texttt{Q}_i}$. Then it is possible to invoke $\texttt{fdec1}$:

$$\begin{aligned}
&\texttt{I}_M(\texttt{dec}_1, _, [\![0, v_2]\!]^{\texttt{I}}_{\texttt{Q}_i} | E_1' | E_2 | E_3)\\
&\quad \xrightarrow{\texttt{fdec1}} \texttt{I}_M(\texttt{dec}_1, _ \Rightarrow \texttt{zero}_1, [\![0, v_2]\!]^{\texttt{I}}_{\texttt{Q}_i} | E_1' | E_2 | E_3)\\
&\quad \longrightarrow \texttt{I}_M(\texttt{zero}_1, _, [\![0, v_2]\!]^{\texttt{I}}_{\texttt{Q}} | E_1' | E_2 | E_3)\\
&\quad \longrightarrow \texttt{I}_M(\texttt{zero}_1, _ \Rightarrow \texttt{aQ}_\texttt{i}, [\![0, v_2]\!]^{\texttt{I}}_{\texttt{Q}_i} | E_2 | E_3)\\
&\quad \longrightarrow \texttt{I}_M(\texttt{aQ}_\texttt{i}, _, [\![0, v_2]\!]^{\texttt{I}}_{\texttt{Q}_i} | E_2 | E_3)\\
&\quad \longrightarrow^* \texttt{// similarly to the above case}\\
&\quad \longrightarrow \texttt{I}_M(\texttt{Q}_j, _, [\![0, v_2]\!]^{\texttt{I}}_{\texttt{Q}_j})
\end{aligned}$$

As regards (2), we assume that $\texttt{I}_M(\texttt{Q}_i, \_, [\![v_1, v_2]\!]^{\texttt{I}}_{\texttt{Q}}) \longrightarrow^* \texttt{I}_M(\texttt{S}, \Sigma, \Psi) \longrightarrow \texttt{I}_M(\texttt{Q}_j, \Sigma', \Psi')$ is such that $\texttt{I}_M(\texttt{Q}_i, \_, [\![v_1, v_2]\!]^{\texttt{I}}_{\texttt{Q}}) \longrightarrow^* \texttt{I}_M(\texttt{S}, \Sigma, \Psi)$ does not contain pairs of transitions $\texttt{I}_M(\texttt{S}'', \Sigma'', \Psi'') \longrightarrow \texttt{I}_M(\texttt{Q}'', \Sigma''', \Psi''')$ with $\texttt{S}'' \notin \{\texttt{Q}_0, \cdots, \texttt{Q}_n\}$ and $\texttt{Q}'' \in \{\texttt{Q}_0, \cdots, \texttt{Q}_n\}$. The general case follows by recurrence.

The proof is a case analysis on the first transition of $\texttt{I}_M(\texttt{Q}_i, \_, [\![v_1, v_2]\!]^{\texttt{I}}_{\texttt{Q}_i})$:

- if $\texttt{I}_M(\texttt{Q}_i, \_, [\![v_1, v_2]\!]^{\texttt{I}}_{\texttt{Q}}) \longrightarrow \texttt{I}_M(\texttt{Q}_i, \_, \_)$ is an instance of [Tick]. This case is vacuous: the continuations of $\texttt{I}_M(\texttt{Q}_i, \_, \_)$ may be either instances of [Tick] or the invocation of exactly one function $\texttt{f} \in \{\texttt{fincQ}_\texttt{i}, \texttt{fdecQ}_\texttt{i}, \texttt{fzeroQ}_\texttt{i}\}$ possibly followed by an invocation of either $\texttt{fdec}_1$ or $\texttt{fdec}_2$ and by the execution of the event $0 \gg \texttt{zero}_1 \Rightarrow \texttt{aQ}_\texttt{i}$ or $0 \gg \texttt{zero}_2 \Rightarrow \texttt{aQ}_\texttt{i}$. Therefore no transition $\texttt{I}_M(\texttt{S}, \Sigma, \Psi) \longrightarrow \texttt{I}_M(\texttt{Q}_j, \Sigma', \Psi')$ will be ever possible.
- if $\texttt{I}_M(\texttt{Q}_i, \_, [\![v_1, v_2]\!]^{\texttt{I}}_{\texttt{Q}_i}) \xrightarrow{\texttt{fincQ}_\texttt{i}} \texttt{I}_M(\texttt{Q}_i, Ev \Rightarrow \texttt{aQ}_\texttt{i}, [\![v_1, v_2]\!]^{\texttt{I}}_{\texttt{Q}_i})$ then, according to Table 2, M contains the instruction $\texttt{Q}_i : \mathit{Inc}(R_1, \texttt{Q}_j)$ or $\texttt{Q}_i : \mathit{Inc}(R_2, \texttt{Q}_j)$. We consider the first case, i.e., R_1 is incremented (the argument for R_2 is similar). It is easy to verify that $\texttt{I}_M(\texttt{Q}_i, Ev \Rightarrow \texttt{aQ}_\texttt{i}, [\![v_1, v_2]\!]^{\texttt{I}}_{\texttt{Q}_i}) \longrightarrow^5 \texttt{I}_M(\texttt{Q}_j, \_, [\![v_1 + 1, v_2]\!]^{\texttt{I}}_{\texttt{Q}_j})$ with a deterministic computation that does not traverse configurations $\texttt{I}_M(S'', \Sigma'', \Psi'')$ with $S'' \in \{\texttt{Q}_0, \cdots, \texttt{Q}_n\}$.
- if $\texttt{I}_M(\texttt{Q}_i, \_, [\![v_1, v_2]\!]^{\texttt{I}}_{\texttt{Q}_i}) \xrightarrow{\texttt{fdecQ}_\texttt{i}} \texttt{I}_M(\texttt{Q}_i, Ev \Rightarrow \texttt{aQ}_\texttt{i}, [\![v_1, v_2]\!]^{\texttt{I}}_{\texttt{Q}_i})$ then, according to Table 2, M contains $\texttt{Q}_i : \mathit{DecJump}(R_1, \texttt{Q}', \texttt{Q}'')$ or $\texttt{Q}_i : \mathit{DecJump}(R_2, \texttt{Q}', \texttt{Q}'')$. We consider the first case, i.e., R_1 is decreased/tested (the argument for R_2 is similar). There are two subcases:
  - ($v_1 > 0$) In this case it is easy to verify that $\texttt{I}_M(\texttt{Q}_i, Ev \Rightarrow \texttt{aQ}_\texttt{i}, [\![v_1, v_2]\!]^{\texttt{I}}_{\texttt{Q}_i}) \longrightarrow^8 \texttt{I}_M(\texttt{Q}'', \_, [\![v_1 - 1, v_2]\!]^{\texttt{I}}_{\texttt{Q}''})$ with a computation that does not traverse configurations $\texttt{I}_M(S'', \Sigma'', \Psi'')$ with $S'' \in \{\texttt{Q}_0, \cdots, \texttt{Q}_n\}$. In this case, the computation is not deterministic because the third transition is the invocation of $\texttt{fdec}_1$. Actually, it is also possible to perform a transition [Tick]. Analogous to the first case above, after the transition [Tick], no transition $\texttt{I}_M(\texttt{S}, \Sigma, \Psi) \longrightarrow \texttt{I}_M(\texttt{Q}_j, \Sigma', \Psi')$ will be ever possible.
  - ($v_1 = 0$) This case is vacuous because we have $\texttt{I}_M(\texttt{Q}_i, Ev \Rightarrow \texttt{dec}_1, [\![0, v_2]\!]^{\texttt{I}}_{\texttt{Q}_i}) \longrightarrow^2 \texttt{I}_M(\texttt{dec}_1, \_, [\![0, v_2]\!]^{\texttt{I}}_{\texttt{Q}_i} \mid Ev)$ and, for every $\texttt{I}_M(\texttt{dec}_1, \_, [\![0, v_2]\!]^{\texttt{I}}_{\texttt{Q}_i} \mid Ev) \longrightarrow^* \texttt{I}_M(\texttt{S}, \Sigma, \Psi)$, it is easy to verify that $\texttt{S} \notin \{\texttt{Q}_0, \cdots, \texttt{Q}_n\}$.
- if $\texttt{I}_M(\texttt{Q}_i, \_, [\![v_1, v_2]\!]^{\texttt{I}}_{\texttt{Q}_i}) \xrightarrow{\texttt{fzeroQ}_\texttt{i}} \texttt{I}_M(\texttt{Q}_i, Ev \Rightarrow \texttt{dec}_1, [\![v_1, v_2]\!]^{\texttt{I}}_{\texttt{Q}_i})$ then, according to Table 2, M contains $\texttt{Q}_i : \mathit{DecJump}(R_1, \texttt{Q}', \texttt{Q}'')$ or $\texttt{Q}_i : \mathit{DecJump}(R_2, \texttt{Q}', \texttt{Q}'')$. We consider the first case, i.e., R_1 is considered (the argument for R_2 is similar). There are two subcases:
  - ($v_1 = 0$) We have that $\texttt{I}_M(\texttt{Q}_i, Ev \Rightarrow \texttt{dec}_1, [\![v_1, v_2]\!]^{\texttt{I}}_{\texttt{Q}_i}) \longrightarrow^9 \texttt{I}_M(\texttt{Q}', \_, [\![0, v_2]\!]^{\texttt{I}}_{\texttt{Q}'})$. As in the case of $\texttt{fdecQ}_i$, the computation is not deterministic due to the presence of alternative [Tick] transitions, but after a [Tick] no transition $\texttt{I}_M(\texttt{S}, \Sigma, \Psi) \longrightarrow \texttt{I}_M(\texttt{Q}_j, \Sigma', \Psi')$ will be ever possible.
  - ($v_1 > 0$) Vacuous, analogous to the case of $\texttt{fdecQ}_i$ with $v_1 = 0$.

This concludes the proof. □

Theorem 3 *State reachability is undecidable for the fragment μStipula$^{\texttt{TA}}$.*

Proof Let M be the Minsky machine with states $\{\texttt{Q}_1, \cdots, \texttt{Q}_n\}$ and $\texttt{TA}_M$ be the encoding in μStipula$^{\texttt{TA}}$ as defined in Table 3. Let

$$[\![v_1, v_2]\!]^{\texttt{TA}}_{\texttt{Q}} = \underbrace{1 \gg \texttt{dec}_1 \Rightarrow \texttt{ackdec}_1}_{v_1 \text{ times}} \mid \underbrace{1 \gg \texttt{dec}_2 \Rightarrow \texttt{ackdec}_2}_{v_2 \text{ times}} \mid 1 \gg \texttt{Q} \Rightarrow \texttt{end}$$

and let $\Psi_{\texttt{TA}}$, $\Psi'_{\texttt{TA}}$ be either $\_$ or (i is either 1 or 2)

$$0 \gg \texttt{dec}_1 \Rightarrow \texttt{end} \mid 0 \gg \texttt{dec}_2 \Rightarrow \texttt{end} \mid 0 \gg \texttt{nextQb}_i \Rightarrow \texttt{end} \mid 0 \gg \texttt{ackdec}_1 \Rightarrow \texttt{@end} \mid 0 \gg \texttt{ackdec}_2 \Rightarrow \texttt{end}$$

(the cases where Ψ and Ψ' are not $\_$ correspond to those where the machine transition is a *DecJump*). We need to prove

(1) $(\texttt{Q}_i, v_1, v_2) \longrightarrow_{\texttt{M}} (\texttt{Q}_j, v_1', v_2')$ implies $\texttt{TA}_M(\texttt{Q}_i, \_, [\![v_1, v_2]\!]^{\texttt{TA}}_{\texttt{Q}_i} | \Psi) \longrightarrow^* \texttt{TA}_M(\texttt{Q}_j, \_, [\![v_1', v_2']\!]^{\texttt{TA}}_{\texttt{Q}_j} | \Psi')$

(2) $\texttt{TA}_M$ does not transit to unreachable Minsky's states. That is, if $\texttt{TA}_M(\texttt{Q}_i, \_, [\![v_1, v_2]\!]_{\texttt{Q}_i} \mid \Psi_{\texttt{TA}}) \longrightarrow^* \texttt{TA}_M(\texttt{S}, \Sigma, \Psi) \longrightarrow \texttt{TA}_M(\texttt{Q}_j, \Sigma', \Psi')$ with $\texttt{S} \notin \{\texttt{Q}_0, \cdots, \texttt{Q}_n\}$ then $\Sigma' = \_$ and $\Psi' = [\![v_1', v_2']\!]^{\texttt{TA}}_{\texttt{Q}_j} \mid \Psi'_{\texttt{TA}}$ and also $(\texttt{Q}_i, v_1, v_2) \longrightarrow^*_{\texttt{M}} (\texttt{Q}_j, v_1', v_2')$.

The proofs of (1) and (2) consist of a detailed case analysis on the transitions of $\texttt{TA}_M$. The proof of (1) is similar to the corresponding one of Theorem 2, therefore omitted. As regards (2), the difference with that of Theorem 2 is the fact that $\texttt{TA}_M(\texttt{Q}_i, \_, [\![v_1, v_2]\!]^{\texttt{TA}}_{\texttt{Q}_i} \mid \Psi_{\texttt{TA}}) \longrightarrow^* \texttt{TA}_M(\texttt{S}, \Sigma, \Psi) \longrightarrow \texttt{TA}_M(\texttt{Q}_j, \Sigma', \Psi')$ contains exactly 3 transitions [Tick] (in the basic case where no state $\{\texttt{Q}_0, \cdots, \texttt{Q}_n\}$ is traversed). Otherwise one has to demonstrate that the case becomes vacuous. The details are omitted. In the same way we may conclude the proof. □

Theorem 4 *State reachability is undecidable for the fragment μStipula$^{\texttt{D}}$.*

Proof Let M be the Minsky machine and $\texttt{D}_M$ be the encoding in μStipula$^{\texttt{D}}$ as defined in Tables 4 and 5. The initial state of $\texttt{D}_M$ is $\texttt{D}_M(\texttt{Start}, \_, \_)$ and, after a sequence of transitions [Tick], the contract $\texttt{D}_M$ performs the transitions

$$\texttt{D}_M(\texttt{Start}, \_, \_) \xrightarrow{\texttt{fstart}} \texttt{D}_M(\texttt{Start}, Ev \Rightarrow \texttt{Q}_0, \_) \longrightarrow \texttt{D}_M(\texttt{Q}_0, \_, Ev)$$

where $\texttt{Q}_0$ is the initial state of M and $Ev = 0 \gg \texttt{notickA} \Rightarrow \texttt{cont}$. Next, let

$$[\![v_1, v_2]\!]^{\texttt{D}}_{\texttt{X}} = \underbrace{1 \gg \texttt{dec}_1 \Rightarrow \texttt{ackdec}_1}_{v_1 \text{ times}} \mid \underbrace{3 \gg \texttt{dec}_2 \Rightarrow \texttt{ackdec}_2}_{v_2 \text{ times}} \mid 0 \gg \texttt{notickX} \Rightarrow \texttt{cont}$$

where $\texttt{X}$ may be either $\texttt{A}$ or $\texttt{B}$. We need to prove

(1) $(\texttt{Q}_i, v_1, v_2) \longrightarrow_{\texttt{M}} (\texttt{Q}_j, v_1', v_2')$ implies $\texttt{D}_M(\texttt{Q}_i, \_, [\![v_1, v_2]\!]^{\texttt{D}}_{\texttt{X}}) \longrightarrow^* \texttt{D}_M(\texttt{Q}_j, \_, [\![v_1', v_2']\!]^{\texttt{D}}_{\texttt{Y}})$ where $\texttt{X}$ and $\texttt{Y}$ are either $\texttt{A}$ or $\texttt{B}$;

(2) $\mathtt{D}_M$ does not transit to unreachable Minsky's states.
That is, if $\mathtt{D}_M(\mathtt{Q}_i, \_, [\![v_1, v_2]\!]^{\mathtt{D}}_{\mathtt{X}}) \longrightarrow^* \mathtt{D}_M(\mathtt{S}, \Sigma, \Psi) \longrightarrow \mathtt{D}_M(\mathtt{Q}_j, \Sigma', \Psi')$, with $\mathtt{S} \notin \{\mathtt{Q}_0, \cdots, \mathtt{Q}_n\}$, then $\Sigma' = \_$ and $\Psi' = [\![v'_1, v'_2]\!]^{\mathtt{D}}_{\mathtt{Y}}$ and $(\mathtt{Q}_i, v_1, v_2) \longrightarrow^*_{\mathtt{M}} (\mathtt{Q}_j, v'_1, v'_2)$.

The proof of (1) is a case analysis on the type of Minsky instructions. We omit the simulation of the *Inc* instruction, which is simple, and discuss in detail the simulation of *DecJump*. While the proof is similar to that of Theorem 2, in this case it is much longer because of the complexity of the protocol. In the proof we use the following notation:

$$(\!|v_1, v_2|\!)^{\mathtt{D}} = \underbrace{1 \gg \mathtt{dec}_1 \Rightarrow \mathtt{ackdec}_1}_{v_1 \text{ times}} \mid \underbrace{3 \gg \mathtt{dec}_2 \Rightarrow \mathtt{ackdec}_2}_{v_2 \text{ times}}$$

that is, $(\!|v_1, v_2|\!)^{\mathtt{D}}$ is $[\![v_1, v_2]\!]^{\mathtt{D}}_{\mathtt{X}}$ without the event $0 \gg \mathtt{notickX} \Rightarrow \mathtt{cont}$. To highlight [Tick] transitions we colour them in red.

We assume the involved register is R_1 and that the multiset of events contains $0 \gg \mathtt{notickA} \Rightarrow \mathtt{cont}$ (the case when the multiset contains $0 \gg \mathtt{notickB} \Rightarrow \mathtt{cont}$ is similar). There are two cases:

$R_1 > 0$: therefore M performs the transition $(\mathtt{Q}_i, v_1, v_2) \longrightarrow_{\mathtt{M}} (\mathtt{Q}_j, v_1 - 1, v_2)$. Correspondingly, in $\mathtt{D}_M$, an invocation of the function $\mathtt{fAdecQ}_i$ occurs. Then the contract performs the transitions

$$\begin{aligned}\mathtt{D}_M(\mathtt{Q}_i, \_, [\![v_1, v_2]\!]^{\mathtt{D}}_{\mathtt{A}}) &\xrightarrow{\mathtt{fAdecQ}_i} \mathtt{D}_M(\mathtt{Q}_i, E_1|E_2|E_3 \Rightarrow \mathtt{notickA}, [\![v_1, v_2]\!]^{\mathtt{D}}_{\mathtt{A}}) \\ &\longrightarrow \mathtt{D}_M(\mathtt{Q}_i, \_ \Rightarrow \mathtt{notickA}, [\![v_1, v_2]\!]^{\mathtt{D}}_{\mathtt{A}}|E_1|E_2|E_3) \\ &\longrightarrow \mathtt{D}_M(\mathtt{notickA}, \_, [\![v_1, v_2]\!]^{\mathtt{D}}_{\mathtt{A}}|E_1|E_2|E_3)\end{aligned}$$

where $E_1 = 1 \gg \mathtt{ackdec}_1 \Rightarrow \mathtt{Q}_j\mathtt{\_start1}$, $E_2 = 1 \gg \mathtt{s1notick} \Rightarrow \mathtt{cont}$, and $E_3 = 0 \gg \mathtt{cont} \Rightarrow \mathtt{dec}_1$. Then we have (in the following we omit the labels of the events):

$$\begin{aligned}&\mathtt{D}_M(\mathtt{notickA}, \_, [\![v_1, v_2]\!]^{\mathtt{D}}_{\mathtt{A}}|E_1|E_2|E_3) \\ &\quad\longrightarrow \mathtt{D}_M(\mathtt{notickA}, \_ \Rightarrow \mathtt{cont}, (\!|v_1, v_2|\!)^{\mathtt{D}}|E_1|E_2|E_3) \\ &\quad\longrightarrow \mathtt{D}_M(\mathtt{cont}, \_, (\!|v_1, v_2|\!)^{\mathtt{D}}|E_1|E_2|E_3) \\ &\quad\longrightarrow \mathtt{D}_M(\mathtt{cont}, \_ \Rightarrow \mathtt{dec}_1, (\!|v_1, v_2|\!)^{\mathtt{D}}|E_2) \\ &\quad\longrightarrow \mathtt{D}_M(\mathtt{dec}_1, \_, (\!|v_1, v_2|\!)^{\mathtt{D}}|E_1|E_2) \\ &\quad\longrightarrow \mathtt{D}_M(\mathtt{dec}_1, \_, (\!|v_1, v_2|\!)^{\mathtt{D}}|E_1|E_2)\downarrow)\end{aligned}$$

where the last transition is an instance of [Tick]; notice that no other transition is possible. Since $v_1 > 0$, there is at least one event $0 \gg \mathtt{@dec}_1 \Rightarrow \mathtt{ackdec}_1$ in $(\!|v_1, v_2|\!)^{\mathtt{D}}\downarrow$. Then

$$\begin{aligned}&\mathtt{D}_M(\mathtt{dec}_1, \_, (\!|v_1, v_2|\!)^{\mathtt{D}}|E_1|E_2)\downarrow) \\ &\quad\longrightarrow \mathtt{D}_M(\mathtt{dec}_1, \_ \Rightarrow \mathtt{ackdec}_1, (\!|v_1{-}1, v_2|\!)^{\mathtt{D}}|E_1|E_2)\downarrow) \\ &\quad\longrightarrow \mathtt{D}_M(\mathtt{ackdec}_1, \_, (\!|v_1{-}1, v_2|\!)^{\mathtt{D}}|E_1|E_2)\downarrow) \\ &\quad\longrightarrow \mathtt{D}_M(\mathtt{ackdec}_1, \_ \Rightarrow \mathtt{Q}_j\mathtt{\_start1}, (\!|v_1{-}1, v_2|\!)^{\mathtt{D}}|E_2)\downarrow) \\ &\quad\longrightarrow \mathtt{D}_M(\mathtt{Q}_j\mathtt{\_start1}, \_, (\!|v_1{-}1, v_2|\!)^{\mathtt{D}}|E_2)\downarrow)\end{aligned}$$

At this point, we have to move ahead of 5 time units the events in $(\!|v_1{-}1, v_2|\!)^{\mathtt{D}}\downarrow$. We use the management functions of Table 5, in particular $\mathtt{fQ}_j\mathtt{\_start1}$, then the protocol continues with $\mathtt{fAcopy1}$ alternating with $\mathtt{fBcopy1}$ for $v_1{-}1$ times in total, and $\mathtt{fAcopy2}$ alternating with $\mathtt{fBcopy2}$ for v_2 times in total. Let $\prod_{h\in 1..7} E'_h = E'_1|\cdots|E'_7$ and let E'_A, E'_B and V_r^{+k}, where r is either 1 or 2, be defined as follows:

$$\begin{aligned}E'_1 &= \mathtt{@cont} \Rightarrow \mathtt{@dec}_1 \\ E'_2 &= \mathtt{now} \gg \mathtt{@ackdec}_1 \Rightarrow \mathtt{@copy}_1 \\ E'_3 &= \mathtt{now} + 1 \gg \mathtt{@dec}_1 \Rightarrow \mathtt{@dec}_2 \\ E'_4 &= \mathtt{now} + 2 \gg \mathtt{@ackdec}_2 \Rightarrow \mathtt{@copy}_2 \\ E'_5 &= \mathtt{now} + 2 \gg \mathtt{@c2notickA} \Rightarrow \mathtt{@cont} \\ E'_6 &= \mathtt{now} + 4 \gg \mathtt{@dec}_2 \Rightarrow \mathtt{@Q}_j \\ E'_7 &= \mathtt{now} + 4 \gg \mathtt{@notickA} \Rightarrow \mathtt{@cont}\end{aligned}$$

$$\begin{aligned}E'_A &= \mathtt{now} \gg \mathtt{@c1notickA} \Rightarrow \mathtt{@cont} \\ E'_B &= \mathtt{now} \gg \mathtt{@c1notickB} \Rightarrow \mathtt{@cont} \\ V_r^{+k} &= k \gg \mathtt{@dec}_r \Rightarrow \mathtt{@ackdec}_r\end{aligned}$$

We also use the following notation: X may be either A or B. If $X = A$ then X' is B; if $X = B$ then $X' = A$. Then:

$$\begin{aligned}&\mathtt{D}_M(\mathtt{Q}_j\mathtt{\_start1}, \_, (\!|v_1{-}1, v_2|\!)^{\mathtt{D}}|E_2)\downarrow) \\ &\xrightarrow{\mathtt{fQ}_j\mathtt{start1}} \mathtt{D}_M(\mathtt{Q}_j\mathtt{\_start1}, \textstyle\prod_{h\in 1..7} E'_h|E'_A \Rightarrow \mathtt{s1notick}, \\ &\qquad\qquad (\!|v_1{-}1, v_2|\!)^{\mathtt{D}}|E_2)\downarrow) \\ &\quad\longrightarrow \mathtt{D}_M(\mathtt{Q}_j\mathtt{\_start1}, \_ \Rightarrow \mathtt{s1notick}, \\ &\qquad\qquad \textstyle\prod_{h\in 1..7} E'_h|E'_A|(\!|v_1{-}1, v_2|\!)^{\mathtt{D}}|E_2)\downarrow) \\ &\quad\longrightarrow \mathtt{D}_M(\mathtt{s1notick}, \_, \textstyle\prod_{h\in 1..7} E'_h|E'_A|(\!|v_1{-}1, v_2|\!)^{\mathtt{D}}|E_2)\downarrow) \\ &\quad\longrightarrow \mathtt{D}_M(\mathtt{s1notick}, \_ \Rightarrow \mathtt{cont}, \textstyle\prod_{h\in 1..7} E'_h|E'_A|(\!|v_1{-}1, v_2|\!)^{\mathtt{D}}\downarrow) \\ &\quad\longrightarrow \mathtt{D}_M(\mathtt{cont}, \_, \textstyle\prod_{h\in 1..7} E'_h|E'_A|(\!|v_1{-}1, v_2|\!)^{\mathtt{D}}\downarrow) \\ &\qquad\qquad \mathtt{//\ initially\ } X = A \mathtt{\ and\ } \ell = 0 \\ &= \Big(\mathtt{D}_M(\mathtt{cont}, \_, \textstyle\prod_{h\in 1..7} E'_h|E'_X|(\!|v_1{-}1, v_2|\!)^{\mathtt{D}}\downarrow|(V_1^{+5})^{\ell \text{ times}}) \\ &\quad\longrightarrow \mathtt{D}_M(\mathtt{cont}, \_ \Rightarrow \mathtt{dec}_1, \\ &\qquad\qquad \textstyle\prod_{h\in 2..7} E'_h|E'_X|(\!|v_1{-}1, v_2|\!)^{\mathtt{D}}\downarrow|(V_1^{+5})^{\ell \text{ times}}) \\ &\quad\longrightarrow \mathtt{D}_M(\mathtt{dec}_1, \_, \textstyle\prod_{h\in 2..7} E'_h|E'_A|(\!|v_1{-}1, v_2|\!)^{\mathtt{D}}\downarrow|(V_1^{+5})^{\ell \text{ times}}) \\ &\quad\longrightarrow \mathtt{D}_M(\mathtt{dec}_1, \_ \Rightarrow \mathtt{ackdec}_1, \\ &\qquad\qquad \textstyle\prod_{h\in 2..7} E'_h|E'_X|(\!|v_1{-}2, v_2|\!)^{\mathtt{D}}\downarrow|(V_1^{+5})^{\ell \text{ times}}) \\ &\quad\longrightarrow \mathtt{D}_M(\mathtt{ackdec}_1, \_, \\ &\qquad\qquad \textstyle\prod_{h\in 2..7} E'_h|E'_X|(\!|v_1{-}2, v_2|\!)^{\mathtt{D}}\downarrow|(V_1^{+5})^{\ell \text{ times}}) \\ &\quad\longrightarrow \mathtt{D}_M(\mathtt{ackdec}_1, \_ \Rightarrow \mathtt{copy}_1, \\ &\qquad\qquad \textstyle\prod_{h\in 3..7} E'_h|E'_X|(\!|v_1{-}2, v_2|\!)^{\mathtt{D}}\downarrow|(V_1^{+5})^{\ell \text{ times}}) \\ &\quad\longrightarrow \mathtt{D}_M(\mathtt{copy}_1, \_, \textstyle\prod_{h\in 3..7} E'_h|E'_X|(\!|v_1{-}2, v_2|\!)^{\mathtt{D}}\downarrow|(V_1^{+5})^{\ell \text{ times}}) \\ &\xrightarrow{\mathtt{fXcopy1}} \mathtt{D}_M(\mathtt{copy}_1, \textstyle\prod_{h\in 1..2} E'_h|E'_{X'}|V_1^{+5} \Rightarrow \mathtt{c1notickX}, \\ &\qquad\qquad \textstyle\prod_{h\in 3..7} E'_h|E'_X|(\!|v_1{-}2, v_2|\!)^{\mathtt{D}}\downarrow|(V_1^{+5})^{\ell \text{ times}}) \\ &\quad\longrightarrow \mathtt{D}_M(\mathtt{copy}_1, \_ \Rightarrow \mathtt{c1notickX}, \\ &\qquad\qquad \textstyle\prod_{h\in 1..7} E'_h|E'_X|E'_{X'}|(\!|v_1{-}2, v_2|\!)^{\mathtt{D}}\downarrow|(V_1^{+5})^{\ell+1 \text{ times}}) \\ &\quad\longrightarrow \mathtt{D}_M(\mathtt{c1notickX}, \_, \\ &\qquad\qquad \textstyle\prod_{h\in 1..7} E'_h|E'_X|E'_{X'}|(\!|v_1{-}2, v_2|\!)^{\mathtt{D}}\downarrow|(V_1^{+5})^{\ell+1 \text{ times}}) \\ &\quad\longrightarrow \mathtt{D}_M(\mathtt{c1notickX}, \_ \Rightarrow \mathtt{cont}, \\ &\qquad\qquad \textstyle\prod_{h\in 1..7} E'_h|E'_{X'}|(\!|v_1{-}2, v_2|\!)^{\mathtt{D}}\downarrow|(V_1^{+5})^{\ell+1 \text{ times}}) \\ &\quad\longrightarrow \mathtt{D}_M(\mathtt{cont}, \_, \\ &\qquad\qquad \textstyle\prod_{h\in 1..7} E'_h|E'_{X'}|(\!|v_1{-}2, v_2|\!)^{\mathtt{D}}\downarrow|(V_1^{+5})^{\ell+1 \text{ times}}) \\ &\Big) \mathtt{\ //\ repeated\ } v_1 - 1 \mathtt{\ times;\ } X \mathtt{\ takes\ the\ value\ of\ } X'\end{aligned}$$

At this point, the events $\mathtt{dec}_1 \Rightarrow \mathtt{ackdec}_1$ have been moved 5 ticks ahead. We need to do the same for the events $\mathtt{dec}_2 \Rightarrow \mathtt{ackdec}_2$. This will require to advance of two ticks:

$$\begin{aligned}&\mathtt{D}_M(\mathtt{cont}, \_, \textstyle\prod_{h\in 1..7} E'_h|E'_{X'}|(\!|0, v_2|\!)^{\mathtt{D}}\downarrow|(V_1^{+5})^{v_1-1 \text{ times}}) \\ &\longrightarrow \mathtt{D}_M(\mathtt{cont}, \_ \Rightarrow \mathtt{dec}_1, \\ &\qquad\qquad \textstyle\prod_{h\in 2..7} E'_h|E'_{X'}|(\!|0, v_2|\!)^{\mathtt{D}}\downarrow|(V_1^{+5})^{v_1-1 \text{ times}}) \\ &\longrightarrow \mathtt{D}_M(\mathtt{dec}_1, \_, \textstyle\prod_{h\in 2..7} E'_h|E'_{X'}|(\!|0, v_2|\!)^{\mathtt{D}}\downarrow|(V_1^{+5})^{v_1-1 \text{ times}}) \\ &\longrightarrow \mathtt{D}_M(\mathtt{dec}_1, \_, (\textstyle\prod_{h\in 3..7} E'_h)\downarrow|(\!|0, v_2|\!)^{\mathtt{D}}\downarrow\downarrow|(V_1^{+4})^{v_1-1 \text{ times}}) \\ &\longrightarrow \mathtt{D}_M(\mathtt{dec}_1, \_ \Rightarrow \mathtt{dec}_2, \\ &\qquad\qquad \textstyle\prod_{h\in 4..7} E'_h\downarrow|(\!|0, v_2|\!)^{\mathtt{D}}\downarrow\downarrow|(V_1^{+4})^{v_1-1 \text{ times}}) \\ &\longrightarrow \mathtt{D}_M(\mathtt{dec}_2, \_, \textstyle\prod_{h\in 4..7} E'_h\downarrow|(\!|0, v_2|\!)^{\mathtt{D}}\downarrow\downarrow|(V_1^{+4})^{v_1-1 \text{ times}}) \\ &\longrightarrow \mathtt{D}_M(\mathtt{dec}_2, \_, \textstyle\prod_{h\in 4..7} E'_h\downarrow\downarrow|(\!|0, v_2|\!)^{\mathtt{D}}\downarrow\downarrow\downarrow|(V_1^{+3})^{v_1-1 \text{ times}}) \\ &\longrightarrow \mathtt{D}_M(\mathtt{dec}_2, \_ \Rightarrow \mathtt{ackdeck}_2, \\ &\qquad\qquad \textstyle\prod_{h\in 5..7} E'_h\downarrow\downarrow|(\!|0, v_2|\!)^{\mathtt{D}}\downarrow\downarrow\downarrow|(V_1^{+3})^{v_1-1 \text{ times}}) \\ &\longrightarrow \mathtt{D}_M(\mathtt{ackdec}_2, \_, \\ &\qquad\qquad \textstyle\prod_{h\in 5..7} E'_h\downarrow\downarrow|(\!|0, v_2|\!)^{\mathtt{D}}\downarrow\downarrow\downarrow|(V_1^{+3})^{v_1-1 \text{ times}}) \\ &\longrightarrow \mathtt{D}_M(\mathtt{ackdec}_2, \_ \Rightarrow \mathtt{copy}_2, \\ &\qquad\qquad \textstyle\prod_{h\in 6..7} E'_h\downarrow\downarrow|(\!|0, v_2|\!)^{\mathtt{D}}\downarrow\downarrow\downarrow|(V_1^{+3})^{v_1-1 \text{ times}}) \\ &\longrightarrow \mathtt{D}_M(\mathtt{copy}_2, \_, \textstyle\prod_{h\in 6..7} E'_h\downarrow\downarrow|(\!|0, v_2|\!)^{\mathtt{D}}\downarrow\downarrow\downarrow|(V_1^{+3})^{v_1-1 \text{ times}}) \\ &\longrightarrow \mathtt{//\ similar\ to\ the\ above\ iteration;\ use\ fXcopy}_2 \\ &\longrightarrow \mathtt{D}_M(\mathtt{dec}_2, \_, \textstyle\prod_{h\in 6..7} E'_h\downarrow\downarrow|(V_1^{+3})^{v_1-1 \text{ times}}|(V_2^{+5})^{v_2 \text{ times}}) \\ &\longrightarrow^2 \mathtt{D}_M(\mathtt{dec}_2, \_, \\ &\qquad\qquad \textstyle\prod_{h\in 6..7} E'_h\downarrow\downarrow\downarrow\downarrow|(V_1^{+1})^{v_1-1 \text{ times}}|(V_2^{+3})^{v_2 \text{ times}}) \\ &\longrightarrow \mathtt{D}_M(\mathtt{dec}_2, \_ \Rightarrow \mathtt{Q}_j, [\![v_1 - 1, v_2]\!]^{\mathtt{D}}_{\mathtt{A}}) \\ &\longrightarrow \mathtt{D}_M(\mathtt{Q}_j, \_, [\![v_1 - 1, v_2]\!]^{\mathtt{D}}_{\mathtt{A}})\end{aligned}$$

where the [Tick] transitions are the unique possible in the corresponding configurations.

$R_1 = 0$: therefore M performs the transition $(\mathtt{Q}_i, 0, v_2) \longrightarrow_{\mathtt{M}} (\mathtt{Q}_j, 0, v_2)$. Correspondingly, in $\mathtt{D}_M$, an invocation of the function $\mathtt{fzeroQ}_i$ occurs. Let

$$E_1 = 0 \gg \mathtt{cont} \Rightarrow \mathtt{dec1} \qquad E_2 = 2 \gg \mathtt{dec}_1 \Rightarrow \mathtt{dec}_2$$
$$E_3 = 3 \gg \mathtt{ackdec}_2 \Rightarrow \mathtt{copy}_2 \qquad E_4 = 3 \gg \mathtt{c2nothickA} \Rightarrow \mathtt{cont}$$
$$E_5 = 5 \gg \mathtt{dec}_2 \Rightarrow \mathtt{Q}_j \qquad E_6 = 5 \gg \mathtt{nothickB} \Rightarrow \mathtt{cont}\ .$$

Then the contract performs the transitions:

$$\begin{aligned}
&\mathtt{D}_M(\mathtt{Q}_i, _, [\![0, v_2]\!]^{\mathtt{D}}_{\mathtt{A}}) \\
&\overset{\mathtt{fAzeroQ}_i}{\longrightarrow} \mathtt{D}_M(\mathtt{Q}_i, \textstyle\prod_{h\in 1..6} E_h \Rightarrow \mathtt{notickA}, [\![0, v_2]\!]^{\mathtt{D}}_{\mathtt{A}}) \\
&\longrightarrow \mathtt{D}_M(\mathtt{Q}_i, _ \Rightarrow \mathtt{notickA}, [\![0, v_2]\!]^{\mathtt{D}}_{\mathtt{A}} | \textstyle\prod_{h\in 1..6} E_h) \\
&\longrightarrow \mathtt{D}_M(\mathtt{notickA}, _, [\![0, v_2]\!]^{\mathtt{D}}_{\mathtt{A}} | \textstyle\prod_{h\in 1..6} E_h) \\
&\longrightarrow \mathtt{D}_M(\mathtt{notickA}, _, [\![0, v_2]\!]^{\mathtt{D}}_{\mathtt{A}} | \textstyle\prod_{h\in 1..6} E_h) \\
&\longrightarrow \mathtt{D}_M(\mathtt{notickA}, _ \Rightarrow \mathtt{cont}, \wr 0, v_2 \wr^{\mathtt{D}}_{\mathtt{A}} | \textstyle\prod_{h\in 1..6} E_h) \\
&\longrightarrow \mathtt{D}_M(\mathtt{cont}, _, \wr 0, v_2 \wr^{\mathtt{D}}_{\mathtt{A}} | \textstyle\prod_{h\in 1..6} E_h) \\
&\longrightarrow \mathtt{D}_M(\mathtt{cont}, _ \Rightarrow \mathtt{dec}_1, \wr 0, v_2 \wr^{\mathtt{D}}_{\mathtt{A}} | \textstyle\prod_{h\in 2..6} E_h) \\
&\longrightarrow \mathtt{D}_M(\mathtt{dec}_1, _, \wr 0, v_2 \wr^{\mathtt{D}}_{\mathtt{A}} | \textstyle\prod_{h\in 2..6} E_h)
\end{aligned}$$

In this configuration, two tick transitions occur and we have

$$\begin{aligned}
&\mathtt{D}_M(\mathtt{dec}_1, _, \wr 0, v_2 \wr^{\mathtt{D}}_{\mathtt{A}} | \textstyle\prod_{h\in 2..6} E_h) \\
&\longrightarrow^2 \mathtt{D}_M(\mathtt{dec}_1, _, \wr 0, v_2 \wr^{\mathtt{D}}_{\mathtt{A}} \downarrow\downarrow | \textstyle\prod_{h\in 2..6} E_h \downarrow\downarrow) \\
&\longrightarrow \mathtt{D}_M(\mathtt{dec}_1, _ \Rightarrow \mathtt{dec}_2, \wr 0, v_2 \wr^{\mathtt{D}}_{\mathtt{A}} \downarrow\downarrow | \textstyle\prod_{h\in 3..6} E_h \downarrow\downarrow) \\
&\longrightarrow \mathtt{D}_M(\mathtt{dec}_2, _, \wr 0, v_2 \wr^{\mathtt{D}}_{\mathtt{A}} \downarrow\downarrow | \textstyle\prod_{h\in 3..6} E_h \downarrow\downarrow) \\
&\longrightarrow \mathtt{D}_M(\mathtt{dec}_2, _, \wr 0, v_2 \wr^{\mathtt{D}}_{\mathtt{A}} \downarrow\downarrow\downarrow | \textstyle\prod_{h\in 3..6} E_h \downarrow\downarrow\downarrow) \\
&\longrightarrow \mathtt{D}_M(\mathtt{dec}_2, _ \Rightarrow \mathtt{ackdec}_2, \wr 0, v_2 - 1 \wr^{\mathtt{D}}_{\mathtt{A}} \downarrow\downarrow\downarrow | \textstyle\prod_{h\in 3..6} E_h \downarrow\downarrow\downarrow) \\
&\longrightarrow \mathtt{D}_M(\mathtt{ackdec}_2, _, \wr 0, v_2 - 1 \wr^{\mathtt{D}}_{\mathtt{A}} \downarrow\downarrow\downarrow | \textstyle\prod_{h\in 3..6} E_h \downarrow\downarrow\downarrow) \\
&\longrightarrow \mathtt{D}_M(\mathtt{ackdec}_2, _ \Rightarrow \mathtt{copy}_2, \wr 0, v_2 - 1 \wr^{\mathtt{D}}_{\mathtt{A}} \downarrow\downarrow\downarrow | \textstyle\prod_{h\in 4..6} E_h \downarrow\downarrow\downarrow) \\
&\longrightarrow \mathtt{D}_M(\mathtt{copy}_2, _, \wr 0, v_2 - 1 \wr^{\mathtt{D}}_{\mathtt{A}} \downarrow\downarrow\downarrow | \textstyle\prod_{h\in 4..6} E_h \downarrow\downarrow\downarrow)
\end{aligned}$$

At this point two transitions may occur: either a tick transition or the management function $\mathtt{fAcopy2}$ of Table 5 can be invoked. In the first case the contract transits to $\mathtt{D}_M(\mathtt{copy}_2, \_, \prod_{h\in 5..6} E_h \downarrow\downarrow\downarrow\downarrow)$ and it is easy to verify that, from this configuration, it is not possible to transit to a configuration $\mathtt{D}_M(\mathtt{Q}, \Sigma, \Psi)$ anymore. In the second case, by calling $\mathtt{fAcopy2}$ and $\mathtt{fBcopy2}$ one after the other for v_2 times (in total), it is possible to move ahead of 5 time units the events $\mathtt{dec}_2 \Rightarrow \mathtt{ackdec}_2$. Let $\prod_{h\in 1..3} E'_h$ where

$$E'_1 = 0 \gg \mathtt{cont} \Rightarrow \mathtt{dec}_2 \qquad E'_2 = 0 \gg \mathtt{ackdec}_2 \Rightarrow \mathtt{copy}_2$$
$$E'_3 = 0 \gg \mathtt{c2notickB} \Rightarrow \mathtt{cont}$$

and let $V_2^{+k} = k \gg \mathtt{dec}_2 \Rightarrow @\mathtt{ackdec}_2$ (as before). Then

$$\begin{aligned}
&\mathtt{D}_M(\mathtt{copy}_2, _, \wr 0, v_2 - 1 \wr^{\mathtt{D}}_{\mathtt{A}} \downarrow\downarrow\downarrow | \textstyle\prod_{h\in 4..6} E_h \downarrow\downarrow\downarrow) \\
&\overset{\mathtt{fAcopy2}}{\longrightarrow} \mathtt{D}_M(\mathtt{copy}_2, (\textstyle\prod_{h\in 1..3} E'_h | V_2^{+5}) \Rightarrow \mathtt{c2notickA}, \\
&\qquad \wr 0, v_2 - 1 \wr^{\mathtt{D}}_{\mathtt{A}} \downarrow\downarrow\downarrow | \textstyle\prod_{h\in 4..6} E_h \downarrow\downarrow\downarrow) \\
&\longrightarrow \mathtt{D}_M(\mathtt{copy}_2, _ \Rightarrow \mathtt{c2notickA}, \\
&\qquad \wr 0, v_2 - 1 \wr^{\mathtt{D}}_{\mathtt{A}} \downarrow\downarrow\downarrow | \textstyle\prod_{h\in 4..6} E_h \downarrow\downarrow\downarrow | \textstyle\prod_{h\in 1..3} E'_h | V_2^{+5}) \\
&\longrightarrow \mathtt{D}_M(\mathtt{c2notickA}, _, \\
&\qquad \wr 0, v_2 - 1 \wr^{\mathtt{D}}_{\mathtt{A}} \downarrow\downarrow\downarrow | \textstyle\prod_{h\in 4..6} E_h \downarrow\downarrow\downarrow | \textstyle\prod_{h\in 1..3} E'_h | V_2^{+5}) \\
&\longrightarrow \mathtt{D}_M(\mathtt{c2notickA}, _ \Rightarrow \mathtt{cont}, \\
&\qquad \wr 0, v_2 - 1 \wr^{\mathtt{D}}_{\mathtt{A}} \downarrow\downarrow\downarrow | \textstyle\prod_{h\in 5..6} E_h \downarrow\downarrow\downarrow | \textstyle\prod_{h\in 1..3} E'_h | V_2^{+5}) \\
&\longrightarrow \mathtt{D}_M(\mathtt{cont}, _, \\
&\qquad \wr 0, v_2 - 1 \wr^{\mathtt{D}}_{\mathtt{A}} \downarrow\downarrow\downarrow | \textstyle\prod_{h\in 5..6} E_h \downarrow\downarrow\downarrow | \textstyle\prod_{h\in 1..3} E'_h | V_2^{+5}) \\
&\longrightarrow \mathtt{D}_M(\mathtt{cont}, _ \Rightarrow \mathtt{dec}_2, \\
&\qquad \wr 0, v_2 - 1 \wr^{\mathtt{D}}_{\mathtt{A}} \downarrow\downarrow\downarrow | \textstyle\prod_{h\in 5..6} E_h \downarrow\downarrow\downarrow | \textstyle\prod_{h\in 2..3} E'_h | V_2^{+5}) \\
&\longrightarrow \mathtt{D}_M(\mathtt{dec}_2, _, \\
&\qquad \wr 0, v_2 - 1 \wr^{\mathtt{D}}_{\mathtt{A}} \downarrow\downarrow\downarrow | \textstyle\prod_{h\in 5..6} E_h \downarrow\downarrow\downarrow | \textstyle\prod_{h\in 2..3} E'_h | V_2^{+5}) \\
&\longrightarrow \mathtt{D}_M(\mathtt{dec}_2, _ \Rightarrow \mathtt{ackdec}_2, \\
&\qquad \wr 0, v_2 - 2 \wr^{\mathtt{D}}_{\mathtt{A}} \downarrow\downarrow\downarrow | \textstyle\prod_{h\in 5..6} E_h \downarrow\downarrow\downarrow | \textstyle\prod_{h\in 2..3} E'_h | V_2^{+5}) \\
&\longrightarrow \mathtt{D}_M(\mathtt{ackdec}_2, _, \\
&\qquad \wr 0, v_2 - 2 \wr^{\mathtt{D}}_{\mathtt{A}} \downarrow\downarrow\downarrow | \textstyle\prod_{h\in 5..6} E_h \downarrow\downarrow\downarrow | \textstyle\prod_{h\in 2..3} E'_h | V_2^{+5}) \\
&\longrightarrow \mathtt{D}_M(\mathtt{ackdec}_2, _ \Rightarrow \mathtt{copy}_2, \\
&\qquad \wr 0, v_2 - 2 \wr^{\mathtt{D}}_{\mathtt{A}} \downarrow\downarrow\downarrow | \textstyle\prod_{h\in 5..6} E_h \downarrow\downarrow\downarrow | E'_3 | V_2^{+5}) \\
&\longrightarrow \mathtt{D}_M(\mathtt{copy}_2, _, \wr 0, v_2 - 2 \wr^{\mathtt{D}}_{\mathtt{A}} \downarrow\downarrow\downarrow | \textstyle\prod_{h\in 5..6} E_h \downarrow\downarrow\downarrow | E'_3 | V_2^{+5}) \\
&\overset{\mathtt{fBcopy2}}{\longrightarrow} \cdots \quad \texttt{// as before } v_2 - 2 \texttt{ times} \\
&\cdots \\
&\longrightarrow \mathtt{D}_M(\mathtt{dec}_2, _, \textstyle\prod_{h\in 5..6} E_h \downarrow\downarrow\downarrow | E'_3 | (V_2^{+5})^{v_2 \text{ times}}) \\
&\longrightarrow^2 \mathtt{D}_M(\mathtt{dec}_2, _, \textstyle\prod_{h\in 5..6} E_h \downarrow\downarrow\downarrow\downarrow\downarrow | (V_2^{+3})^{v_2 \text{ times}}) \\
&\longrightarrow \mathtt{D}_M(\mathtt{dec}_2, _ \Rightarrow \mathtt{Q}_j, [\![0, v_2]\!]^{\mathtt{D}}_{\mathtt{X}}) \\
&\longrightarrow \mathtt{D}_M(\mathtt{Q}_j, _, [\![0, v_2]\!]^{\mathtt{D}}_{\mathtt{X}})
\end{aligned}$$

where, as before, the last two [Tick] transitions are the unique possible in the corresponding configuration.

As regards (2), we assume that $\mathtt{D}_M(\mathtt{Q}_i, \_, [\![v_1, v_2]\!]^{\mathtt{D}}_{\mathtt{X}}) \longrightarrow^* \mathtt{D}_M(\mathtt{S}, \Sigma, \Psi) \longrightarrow \mathtt{D}_M(\mathtt{Q}_j, \Sigma', \Psi')$ is such that $\mathtt{D}_M(\mathtt{Q}_i, \_, [\![v_1, v_2]\!]^{\mathtt{D}}_{\mathtt{X}}) \longrightarrow^* \mathtt{D}_M(\mathtt{S}, \Sigma, \Psi)$ does not contain pairs of transitions $\mathtt{D}_M(\mathtt{S}'', \Sigma'', \Psi'') \longrightarrow \mathtt{D}_M(\mathtt{Q}'', \Sigma''', \Psi''')$ such that $\mathtt{S}'' \notin \{\mathtt{Q}_0, \cdots, \mathtt{Q}_n\}$ and $\mathtt{Q}'' \in \{\mathtt{Q}_0, \cdots, \mathtt{Q}_n\}$. The general case follows by recurrence.

The proof is a case analysis on the first transition of $\mathtt{D}_M(\mathtt{Q}_i, \_, [\![v_1, v_2]\!]^{\mathtt{D}}_{\mathtt{X}})$:

- if $\mathtt{D}_M(\mathtt{Q}_i, \_, [\![v_1, v_2]\!]^{\mathtt{D}}_{\mathtt{X}}) \longrightarrow \mathtt{D}_M(\mathtt{Q}_i, \_, \wr v_1, v_2 \wr^{\mathtt{D}})$ is an instance of [Tick]. This case is vacuous: the continuations of $\mathtt{D}_M(\mathtt{Q}_i, \_, \wr v_1, v_2 \wr^{\mathtt{D}})$ may be either instances of [Tick] or the invocation of exactly one function $\mathtt{f} \in \{\mathtt{fXincQ}_i, \mathtt{fXdecQ}_i, \mathtt{fXzeroQ}_i\}$, where $\mathtt{X}$ is either $\mathtt{A}$ or $\mathtt{B}$. In this last case, the continuations may only be [Tick] transitions because they transit to states $\mathtt{notickX}$ and produce events $\mathtt{notickY}$ with $\mathtt{X} \neq \mathtt{Y}$. Therefore no transition $\mathtt{D}_M(\mathtt{S}, \Sigma, \Psi) \longrightarrow \mathtt{D}_M(\mathtt{Q}_j, \Sigma', \Psi')$ will be ever possible.
- if $\mathtt{D}_M(\mathtt{Q}_i, \_, [\![v_1, v_2]\!]^{\mathtt{D}}_{\mathtt{X}}) \overset{\mathtt{fincYQ}_i}{\longrightarrow} \mathtt{D}_M(\mathtt{Q}_i, Ev \Rightarrow \mathtt{notickY}, [\![v_1, v_2]\!]^{\mathtt{D}}_{\mathtt{X}})$ then, according to Table 4, M contains the instruction $\mathtt{Q}_i$: $Inc(R_1, \mathtt{Q}_j)$ or $\mathtt{Q}_i$: $Inc(R_2, \mathtt{Q}_j)$. We consider the first case, i.e., R_1 is incremented (the argument for R_2 is similar). There are two subcases: (i) $\mathtt{X} = \mathtt{Y}$ and (ii) $\mathtt{X} \neq \mathtt{Y}$. In case (i), it is easy to verify that $\mathtt{D}_M(\mathtt{Q}_i, Ev \Rightarrow \mathtt{notickX}, [\![v_1, v_2]\!]^{\mathtt{D}}_{\mathtt{X}}) \longrightarrow^6 \mathtt{D}_M(\mathtt{Q}_j, \_, [\![v_1 + 1, v_2]\!]^{\mathtt{D}}_{\mathtt{Y}})$ with a deterministic computation that does not traverse configurations $\mathtt{D}_M(S'', \Sigma'', \Psi'')$ with $S'' \in \{\mathtt{Q}_0, \cdots, \mathtt{Q}_n\}$. In case (ii), $\mathtt{D}_M(\mathtt{Q}_i, Ev \Rightarrow \mathtt{notickY}, [\![v_1, v_2]\!]^{\mathtt{D}}_{\mathtt{X}}) \longrightarrow^2 \mathtt{D}_M(\mathtt{notickY}, \_, [\![v_1, v_2]\!]^{\mathtt{D}}_{\mathtt{X}} | Ev)$. In $[\![v_1, v_2]\!]^{\mathtt{D}}_{\mathtt{X}} | Ev$ there is no event that starts at $\mathtt{notickY}$ (and there is no function that can be invoked), therefore the unique possible transitions are [Tick] and no state in $\{\mathtt{Q}_0, \cdots, \mathtt{Q}_n\}$ will be ever reached again – the subcase is vacuous.
- if $\mathtt{D}_M(\mathtt{Q}_i, \_, [\![v_1, v_2]\!]^{\mathtt{D}}_{\mathtt{X}}) \overset{\mathtt{fYdecQ}_i}{\longrightarrow} \mathtt{D}_M(\mathtt{Q}_i, Ev \Rightarrow \mathtt{notickY}, [\![v_1, v_2]\!]^{\mathtt{D}}_{\mathtt{X}})$ then, according to Table 4, M contains either $\mathtt{Q}_i$: $DecJump(R_1, \mathtt{Q}', \mathtt{Q}'')$ or $\mathtt{Q}_i$: $DecJump(R_2, \mathtt{Q}', \mathtt{Q}'')$. We consider the

first case, i.e., R_1 is decreased/tested (the argument for R_2 is similar). There are three subcases:

Subcase $\mathtt{X} = \mathtt{Y}$ *and* $v_1 > 0$: Then, letting $E_1 = 1 \gg \mathtt{ackdec}_1 \Rightarrow \mathtt{Q}_j\_\mathtt{start1}$ and $E_2 = 1 \gg \mathtt{s1notick} \Rightarrow \mathtt{cont}$, we have

$$\begin{aligned}&\mathtt{D}_M(\mathtt{Q}_i, Ev \Rightarrow \mathtt{notickX}, [\![v_1, v_2]\!]^{\mathtt{D}}_{\mathtt{X}})\\ &\quad\longrightarrow^6 \mathtt{D}_M(\mathtt{dec}_1, \_, \wr v_1, v_2 \wr^{\mathtt{D}} | E_1 | E_2)\\ &\quad\longrightarrow \mathtt{D}_M(\mathtt{dec}_1, \_, \wr v_1, v_2 \wr^{\mathtt{D}} \downarrow | E_1 \downarrow | E_2 \downarrow)\\ &\quad\longrightarrow^4 \mathtt{D}_M(\mathtt{Q}_j\_\mathtt{start1}, \_, \wr v_1 - 1, v_2 \wr^{\mathtt{D}} \downarrow | E_2 \downarrow)\end{aligned}$$

where the [Tick] transition in the second line is the unique that is possible. In the state $\mathtt{Q}_j\_\mathtt{start1}$ there are two further subcases: (*i*) a [Tick] transition occurs or (*ii*) the function $\mathtt{fQ}_j\_\mathtt{start1}$ is invoked. The subcase (*i*) is vacuous because we obtain the configuration $\mathtt{D}_M(\mathtt{Q}_j\_\mathtt{start1}, \_, \wr 0, v_2 \wr^{\mathtt{D}} \downarrow\downarrow)$ where one may either invoke $\mathtt{fQ}_j\_\mathtt{start1}$ or perform [Tick] transitions. In any case, missing an event that starts at $\mathtt{s1notick}$ like E_1, only [Tick] transitions are possible and a state in $\{\mathtt{Q}_0, \cdots, \mathtt{Q}_n\}$ will be never reached. In case (*ii*), one starts the protocol of Table 5. This protocol transfers the events $1 \gg \mathtt{dec}_1 \Rightarrow \mathtt{ackdec}_1$ and $3 \gg \mathtt{dec}_2 \Rightarrow \mathtt{ackdec}_2$ to 5 ticks ahead by invoking the functions $\mathtt{fXcopy1}$ and $\mathtt{fXcopy2}$, respectively. In correspondence of these invocations, a [Tick] may occur and the case becomes vacuous (as before); otherwise the protocol continues. No state in $\{\mathtt{Q}_0, \cdots, \mathtt{Q}_n\}$ is ever traversed – no such state is present in the functions $\mathtt{fXcopy1}$ and $\mathtt{fXcopy2}$. When the transfer is completed (and only three [Tick] transitions have been performed), the event $0 \gg \mathtt{dec}_2 \Rightarrow \mathtt{Q}_j$ can be executed. The correctness of the final state follows by the discussion in the property (1).

Subcase $\mathtt{X} = \mathtt{Y}$ *and* $v_1 = 0$: This case is vacuous because we have

$$\mathtt{D}_M(\mathtt{Q}_i, Ev \Rightarrow \mathtt{notickX}, [\![0, v_2]\!]^{\mathtt{D}}_{\mathtt{X}}) \longrightarrow^6 \mathtt{D}_M(\mathtt{dec}_1, \_, \wr 0, v_2 \wr^{\mathtt{D}} | E_1 | E_2)$$

where $E_1 = 1 \gg \mathtt{ackdec}_1 \Rightarrow \mathtt{Q}_j\_\mathtt{start1}$ and $E_2 = 1 \gg \mathtt{s1notick} \Rightarrow \mathtt{cont}$. However, missing any event starting at $\mathtt{dec}_1$, the contract may only perform [Tick] transitions.

Subcase $\mathtt{X} \neq \mathtt{Y}$: This case is also vacuous because

$$\mathtt{D}_M(\mathtt{Q}_i, Ev \Rightarrow \mathtt{notickX}, [\![0, v_2]\!]^{\mathtt{D}}_{\mathtt{X}}) \longrightarrow^2 \mathtt{D}_M(\mathtt{notickY}, \_, [\![0, v_2]\!]^{\mathtt{D}}_{\mathtt{X}} | Ev)$$

no event that starts at $\mathtt{notickY}$ is available. Therefore only [Tick] transitions can occur and no state in $\{\mathtt{Q}_0, \cdots, \mathtt{Q}_n\}$ can be ever reached.

– if $\mathtt{D}_M(\mathtt{Q}_i, \_, [\![v_1, v_2]\!]^{\mathtt{D}}_{\mathtt{X}}) \xrightarrow{\mathtt{fYzeroQ}_i} \mathtt{D}_M(\mathtt{Q}_i, Ev \Rightarrow \mathtt{notickY}, [\![v_1, v_2]\!]^{\mathtt{D}}_{\mathtt{X}})$ then, according to Table 4, M contains $\mathtt{Q}_i : \mathit{DecJump}(R_1, \mathtt{Q}', \mathtt{Q}'')$ or $\mathtt{Q}_i : \mathit{DecJump}(R_2, \mathtt{Q}', \mathtt{Q}'')$. We omit the details of the proof because it is similar to the above case; here the unique non-vacuous subcase is when $\mathtt{X} = \mathtt{Y}$ and $v_i = 0$.

This concludes the proof. □

B.2 Proofs of Section 4

Proposition 1 *Let* $\mathtt{C}(\mathtt{Q}, \Sigma, \Psi)$ *be a configuration of a* μ*Stipula*$^{\mathtt{DI}}$ *contract* $\mathtt{C}$ *(or a* μ*Stipula*$^{\mathtt{DI}}_{+}$ *contract). Then*

(i) whenever $\mathtt{Q} \notin \mathtt{InitEv}(\mathtt{C})$ *or* $\Sigma \neq \_$*:*

$$\mathtt{C}(\mathtt{Q}, \Sigma, \Psi) \longrightarrow \mathbb{C} \text{ if and only if } \mathtt{C}(\mathtt{Q}, \Sigma, \Psi) \longrightarrow_{\mathtt{tp}} \mathbb{C}\,;$$

(ii) whenever $\mathtt{Q} \in \mathtt{InitEv}(\mathtt{C})$ *and* $\Psi = 0 \gg_{\mathtt{n}} \mathtt{Q} \Rightarrow \mathtt{Q}' \mid \Psi'$*:*

$$\mathtt{C}(\mathtt{Q}, \_, \Psi) \longrightarrow \mathbb{C} \text{ if and only if } \mathtt{C}(\mathtt{Q}, \_, \Psi) \longrightarrow_{\mathtt{tp}} \mathbb{C}\,;$$

(iii) whenever $\mathtt{Q} \in \mathtt{InitEv}(\mathtt{C})$ *and* $\Psi, \mathtt{Q} \not\rightarrow$ *:*

$$\mathtt{C}(\mathtt{Q}, \_, \Psi) \text{ is stuck if and only if } \mathtt{C}(\mathtt{Q}, \_, \Psi) \not\rightarrow_{\mathtt{tp}}.$$

Proof The proofs of statements (*i*) and (*ii*) are case analyses on the possible transition rules that can be applied. They are straightforward and omitted. As regards (*iii*), we notice that no rule [Function] can be applied because $\mathtt{Q} \in \mathtt{InitEv}(\mathtt{C})$, hence $\mathtt{Q}$ cannot be an initial state of function in μ*Stipula*$^{\mathtt{DI}}$ and μ*Stipula*$^{\mathtt{DI}}_{+}$. [State-Change] cannot be applied because $\Sigma = \_$ and [Event-Match] cannot be applied because $\Psi, \mathtt{Q} \not\rightarrow$. Therefore, the unique rule that can be applied is [Tick] and we have

$$\mathtt{C}(\mathtt{Q}, \_, \Psi) \longrightarrow \mathtt{C}(\mathtt{Q}, \_, \_) \longrightarrow^* \mathtt{C}(\mathtt{Q}, \_, \_)$$

where $\mathtt{C}(\mathtt{Q}, \_, \_) \longrightarrow^* \mathtt{C}(\mathtt{Q}, \_, \_)$ are instances of [Tick], therefore $\mathtt{C}(\mathtt{Q}, \_, \Psi)$ is stuck. Correspondingly, since [Tick] is replaced by [Tick-Plus] in μ*Stipula*$^{\mathtt{DI}}_{+}$, no transition $\longrightarrow_{\mathtt{tp}}$ is possible. Hence $\mathtt{C}(\mathtt{Q}, \Sigma, \Psi) \not\rightarrow_{\mathtt{tp}}$. □

Lemma 1 *For a* μ*Stipula*$^{\mathtt{DI}}_{+}$ *contract* $\mathtt{C}$*,* $(\mathcal{C}_{\mathtt{C}}, \longrightarrow_{\mathtt{tp}}, \preceq)$ *is a well-structured transition system.*

Proof To verify that $(\mathcal{C}, \longrightarrow_{\mathtt{tp}}, \preceq)$ is well-structured we need to demonstrate the properties (1) and (2) of Definition 4:

(1) $\preceq$ *is a well-quasi ordering.* Ψ are multisets of events and the relation $\preceq$ is multiset inclusion. Furthermore, by definitions the set of all possible events is always finite. Therefore, by Dickson's Lemma [15], $\preceq$ is a well-quasi ordering.

(2) $\preceq$ *is upward compatible with* $\longrightarrow_{\mathtt{tp}}$. We demonstrate that, if $\mathtt{C}(\mathtt{Q}, \Sigma, \Psi) \longrightarrow_{\mathtt{tp}} \mathtt{C}(\mathtt{Q}', \Sigma', \Psi')$ and $\mathtt{C}(\mathtt{Q}, \Sigma, \Psi) \preceq \mathtt{C}(\mathtt{Q}, \Sigma, \Psi'')$ then there is Ψ''' s.t. $\mathtt{C}(\mathtt{Q}, \Sigma, \Psi'') \longrightarrow_{\mathtt{tp}} \mathtt{C}(\mathtt{Q}', \Sigma', \Psi''')$ with $\mathtt{C}(\mathtt{Q}, \Sigma, \Psi') \preceq \mathtt{C}(\mathtt{Q}, \Sigma, \Psi''')$. The proof consists of a case analysis on the rule used in the transition $\mathtt{C}(\mathtt{Q}, \Sigma, \Psi) \longrightarrow_{\mathtt{tp}} \mathtt{C}(\mathtt{Q}', \Sigma', \Psi')$. When the rule is an instance of [Function] or [State-Change] the upward compatibility is immediate. If the rule is an instance of [Tick-Plus] then $\mathtt{Q} \notin \mathtt{InitEv}(\mathtt{C})$. Hence the same rule may be applied to $\mathtt{C}(\mathtt{Q}, \Sigma, \Psi'')$; in this case $\Psi' = \_ = \Psi'''$ (because the events in Ψ and Ψ'' have time expressions 0). When the rule is an instance of [Event-Match] then let $0 \gg_{\mathtt{n}} \mathtt{Q} \Rightarrow \mathtt{Q}''$ be the event in Ψ that has been scheduled. Since $\Psi \preceq \Psi''$, the same event can be scheduled in $\mathtt{C}(\mathtt{Q}, \Sigma, \Psi'')$. By definition, we have $\Psi' \preceq \Psi'''$. □

Lemma 2 *Let* $(\mathcal{C}, \longrightarrow_{\mathtt{tp}}, \preceq)$ *be the well-structured transition system of a* μ*Stipula*$^{\mathtt{DI}}_{+}$ *contract. Then,* $\longrightarrow_{\mathtt{tp}}$ *and* $\preceq$ *are decidable and there exists an algorithm for computing a finite basis of* $\uparrow \mathit{Pred}(\uparrow \mathbb{C})$ *for any* $\mathbb{C} \in \mathcal{C}$*.*

Proof Regarding 1, the algorithm for $\preceq$ is trivial, as well as its decidability. The algorithm for $\longrightarrow_{\mathtt{tp}}$ is defined by the operational semantics, henceforth its decidability.

Regarding 2, we know that since $\preceq$ is a well-quasi-ordering the set $\uparrow \mathbb{C}$ has a finite basis. Let $\mathcal{B}$ be such basis, we prove that $\uparrow \mathit{Pred}(\mathcal{B})$ has a finite basis that is also computable. Let $\mathbb{C}' \in \mathcal{B}$; there are three cases:

$\mathbb{C}' = \mathtt{C}(\mathtt{Q}', \Sigma, \Psi)$ and $\Sigma \neq \_$: In this case, the transition $\mathbb{C}'' \longrightarrow_{\mathtt{tp}} \mathbb{C}'$ con be obtained in two possibile ways: by applying [Function] or [Event-Match]. In the first case, $\mathit{Pred}(\mathcal{B})$ contains all the tuples $\mathtt{C}(\mathtt{Q}', \_, \Psi)$ for which there is a function $@\mathtt{Q}'\ \mathtt{f}\,\{\,W\,\} \Rightarrow @\mathtt{Q}'' \in \mathtt{C}$ that can produce Σ; in the second case, $\mathit{Pred}(\mathcal{B})$ contains all the tuples $\mathtt{C}(\mathtt{Q}', \_, \Psi')$ where Ψ' extends Ψ with an event $0 \gg @\mathtt{Q}' \Rightarrow @\mathtt{Q}'' \in \mathtt{C}$ that can produce Σ. In both cases, $\mathit{Pred}(\mathcal{B})$ is finite and computable.

$\mathbb{C}' = \mathtt{C}(\mathtt{Q}', \_, \Psi)$ and $\Psi \neq \_$: The unique rule that gives a transition $\mathbb{C}'' \longrightarrow_{\mathtt{tp}} \mathbb{C}'$ is [State-Change]. In this case we have $\mathbb{C}'' = \mathtt{C}(\mathtt{Q}, W \Rightarrow \mathtt{Q}', \Psi')$ where $W = \left(\mathtt{O} \gg_{\mathtt{n}_i} \mathtt{Q}_i \Rightarrow \mathtt{Q}'_i\right)^{i \in 1..h}$ and $\Psi = \Psi' | \mathtt{O} \gg_{\mathtt{n}_1} \mathtt{Q}_1 \Rightarrow \mathtt{Q}'_1 | \cdots | \mathtt{O} \gg_{\mathtt{n}_h} \mathtt{Q}_h \Rightarrow \mathtt{Q}'_h$. There are two possible cases for obtaining the terms $W \Rightarrow \mathtt{Q}'$: (i) the functions $@\mathtt{Q}\ \mathtt{f}\,\{\, W \,\} \Rightarrow @\mathtt{Q}' \in \mathtt{C}$, for every possible subterm W of Ψ; and (ii) the events $\mathtt{O} \gg @\mathtt{Q} \Rightarrow @\mathtt{Q}' \in \mathtt{C}$ (in this sub-case, $W = \_$). In both cases, $\mathit{Pred}(\mathcal{B})$ is finite and computable.

$\mathbb{C}' = \mathtt{C}(\mathtt{Q}', \_, \_)$: Then the transition $\mathbb{C}'' \longrightarrow_{\mathtt{tp}} \mathbb{C}'$ is an instance of [Tick-Plus]. $\mathbb{C}''$ could be any tuple of type $\mathtt{C}(\mathtt{Q}', \_, \Psi')$. $\{\mathtt{C}(\mathtt{Q}', \_, \_)\}$ is a finite basis for all such tuples. □