Decidability Problems for Micro-Stipula

G. Delzanno[1], C. Laneve[2(✉)], A. Sangnier[1], and G. Zavattaro[2]

[1] DIBRIS, University of Genova, Genova, Italy
[2] DISI, University of Bologna, Bologna, Italy
cosimo.laneve@unibo.it

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, and the instantaneous fragment, where all time expressions evaluate to zero. On the other hand, we identify a decidable subfragment: within the instantaneous fragment, reachability becomes decidable when the initial states of functions and events are disjoint.

1 Introduction

Micro-Stipula, noted μ*Stipula*, is a basic calculus defining *contracts*, namely sets of clauses that are either (a) *parameterless functions*, to be invoked by the external environment, or (b) *events* that are triggered at given times. 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 [24] we have defined an analyzer that uses symbolic expressions to approximate time expressions at static-time and that computes the set of reachable clauses by means of a closure operation based on a fixpoint technique. The analyzer, whose prototype is at [17], is sound (every clause it spots is unreachable) but not complete (there may be unreachable clauses that are not recognized).

© IFIP International Federation for Information Processing 2025
Published by springer Nature Switzerland AG 2025
C. Di Giusto and A. Ravara (Eds.): COORDINATION 2025, LNCS 15731, pp. 133–152, 2025.
https://doi.org/10.1007/978-3-031-95589-1_7

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 *μStipula* that are used to model distinctive elements of legal contracts:

μStipula$^{\mathtt{TA}}$: 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

– *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 *μStipula* with an event like

```
now + k ≫ @Using {
    cost ⊸ Lender
} ⇒ @End
```

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

μStipula$^{\mathtt{I}}$: 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 *μStipula* 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.

μStipula$^{\mathtt{D}}$: 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. The technical report [14] contains a legal contract written in *μStipula*$^{\mathtt{D}}$.

We demonstrate that, even for the aforesaid fragments of *μStipula*, there is no complete algorithm for determining unreachable clauses. The proof technique is based on reducing the unreachability problem to the halting problem for Minsky machines (finite state machines with two registers) [27]. The *μStipula* 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 *μStipula*$^{\mathtt{TA}}$ 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 *μStipula*$^{\mathtt{DI}}$, a fragment of *μStipula* that is the intersection of *μStipula*$^{\mathtt{I}}$ and *μStipula*$^{\mathtt{D}}$, and demonstrate that the corresponding contracts are an instance of well-structured transition systems [18], whereby the reachability problem is decidable. To achieve this result, we had to modify the semantics of *μStipula* 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 *μStipula*$^{\mathtt{DI}}$.

The rest of this paper is structured as follows. Section 2 presents the calculus *μStipula* with examples and the semantics. Section 3 contains the undecidability results for *μStipula*$^{\mathtt{TA}}$, *μStipula*$^{\mathtt{I}}$ and *μStipula*$^{\mathtt{D}}$. Section 4 contains the decidability results about *μStipula*$^{\mathtt{DI}}$. Section 5 reports and discusses related work and Sect. 6 presents general conclusions and future work. The proofs of our propositions, lemmas and theorems are reported in the technical report [14].

2 The Calculus *μStipula*

μStipula is a calculus of contracts. A contract is declared by the term

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

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:

$$\begin{array}{llcl}
\textit{Functions} & F & ::= & \_ \quad | \quad \mathtt{@Q}\ \mathtt{f}\,\{\,W\,\}\Rightarrow \mathtt{@Q'}\ \ F \\
\textit{Events} & W & ::= & \_ \quad | \quad \mathtt{t} \gg \mathtt{@Q} \Rightarrow \mathtt{@Q'}\ \ W \\
\textit{Time expressions} & \mathtt{t} & ::= & \mathtt{now} + \mathtt{k} \qquad\qquad\qquad (\mathtt{k} \in \mathsf{Nat})
\end{array}$$

Contracts transit from one state to another either by invoking a *function* or by running an *event*. Functions $\mathtt{@Q}\ \mathtt{f}\,\{\,W\,\}\Rightarrow \mathtt{@Q'}$ 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 $\mathtt{t} \gg \mathtt{@Q} \Rightarrow \mathtt{@Q}'$ schedules a transition from $\mathtt{Q}$ to $\mathtt{Q}'$ at $\mathtt{t}$ time slot ahead the current time if the contract will be in the state $\mathtt{Q}$. The time expressions are additions $\mathtt{now} + \mathtt{k}$, where $\mathtt{k}$ is a constant (a natural number representing, for example, *minutes*); $\mathtt{now}$ is a place-holder that will be replaced by 0 during the execution, see rule [Function] in Table 1. We always shorten $\mathtt{now} + \mathtt{0}$ into $\mathtt{now}$.

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

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 "$\Rightarrow \mathtt{@Q}$" (see lines 5 and 8) and (*ii*) make the events in their body to be scheduled. In particular, an event $\mathtt{now + k} \gg \mathtt{@Q} \Rightarrow \mathtt{@Q}'$ (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 $\mathtt{ev}_4$. In Q1 there is still a unique possibility: executing $\mathtt{ev}_4$. However,

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

to execute it, it is necessary to wait `1` minute (one clock tick must elapse) – 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 $\mathtt{C}(\mathtt{Q}\,,\,\Sigma\,,\,\Psi)$ where

- $\mathtt{C}$ is the contract name;
- $\mathtt{Q}$ is the current state of the contract;
- Σ is either $\_$ or a term $\Psi \Rightarrow \mathtt{Q}$. 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;
- 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 "|" 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 *μStipula* 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

Table 1. The operational semantics of μ*Stipula*

[Function]
$$\frac{\texttt{@Q f}\{\,W\,\}\Rightarrow\texttt{@Q}'\in\texttt{C}\quad \Psi'=\texttt{LC}_{\texttt{Q}\cdot\texttt{f}\cdot\texttt{Q}'}(W)\qquad \Psi,\texttt{Q}\not\rightarrow}{\texttt{C}(\texttt{Q}\,,\,\_\,,\,\Psi)\xrightarrow{\texttt{f}}\texttt{C}(\texttt{Q}\,,\,\Psi'\Rightarrow\texttt{Q}'\,,\,\Psi)}$$

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

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

[Tick]
$$\frac{\Psi,\texttt{Q}\not\rightarrow}{\texttt{C}(\texttt{Q}\,,\,\_\,,\,\Psi)\longrightarrow\texttt{C}(\texttt{Q}\,,\,\_\,,\,\Psi\downarrow)}$$

– the predicate $\Psi,\texttt{Q}\not\rightarrow$, whose definition is

$$\Psi,\texttt{Q}\not\rightarrow\;\stackrel{\text{def}}{=}\begin{cases}\textit{true} & \text{if }\ \Psi=\_\\ \textit{false} & \text{if }\ \Psi=0\gg_{\texttt{n}}\texttt{Q}\Rightarrow\texttt{Q}'\mid\Psi'\\ \Psi',\texttt{Q}\not\rightarrow & \text{if }\ \Psi=\texttt{k}\gg_{\texttt{n}}\texttt{Q}'\Rightarrow\texttt{Q}''\mid\Psi'\ \text{ and }\ (\texttt{k}\neq 0\ \text{ or }\ \texttt{Q}'\neq\texttt{Q})\end{cases}$$

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

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

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 $\Psi,\texttt{Q}\not\rightarrow$ (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 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

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

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

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 [24], they were introduced to ease the formal reasonings, labels 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 remarked.

- *μStipula* has three *causes for nondeterminism*: (*i*) two 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 and 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 *μStipula*$^{\mathtt{I}}$). This paradoxical behaviour, which is also present in process calculi with time [21,28], 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 *μStipula* in this paper is different from, yet equivalent to, [12,24]. 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 *μStipula* semantics, we discuss the computations of the `PingPong` contract. Let $\mathtt{Ev}_4 = 1 \gg_4 \mathtt{Q1} \Rightarrow \mathtt{Q2}$ 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):

$$\begin{array}{rll}
\mathtt{PingPong}(\mathtt{Q0}\,,\,\_\,,\,\_) \longrightarrow^{k} & \mathtt{PingPong}(\mathtt{Q0}\,,\,\_\,,\,\_) & [\textsc{Tick}] \\
\xrightarrow{\mathtt{ping}} & \mathtt{PingPong}(\mathtt{Q0}\,,\,\mathtt{Ev}_4 \Rightarrow \mathtt{Q1}\,,\,\_) & [\textsc{Function}] \\
\longrightarrow & \mathtt{PingPong}(\mathtt{Q1}\,,\,\_\,,\,\mathtt{Ev}_4) & [\textsc{State-Change}] \\
\longrightarrow & \mathtt{PingPong}(\mathtt{Q1}\,,\,\_\,,\,\mathtt{Ev}_4{}^{(-1)}) & [\textsc{Tick}] \\
\xrightarrow{\mathtt{ev}_4} & \mathtt{PingPong}(\mathtt{Q1}\,,\,\_ \Rightarrow \mathtt{Q2}\,,\,\_) & [\textsc{Event-Match}] \\
\longrightarrow & \mathtt{PingPong}(\mathtt{Q2}\,,\,\_\,,\,\_) & [\textsc{State-Change}]
\end{array}$$

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

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

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* $\mathtt{C(Q\,,\,\_\,,\,}\Psi\mathtt{)}$ *such that* $\mathbb{C}_{\mathtt{init}} \longrightarrow^* \mathtt{C(Q\,,\,\_\,,\,}\Psi\mathtt{)}$*.*

It is worth noting that our notion of state reachability is similar to the notion of control state reachability introduced by Alur and Dill in the context of timed automata in [7]. Control state reachability is defined as the problem of checking, given an automaton A and a control state q, if there exists a run of A that visits q. Control state reachability was studied also for lossy Minsky Machines in [25] and lossy FIFO channel systems in [4,10]. In the context of Petri Nets it can be reformulated in terms of coverability of a given target marking [23].

2.3 Relevant Sublanguages

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

μ*Stipula*$^{\mathtt{I}}$, called instantaneous μ*Stipula*, is the fragment where every time expression of the events is `now + 0`;

μ*Stipula*$^{\mathtt{TA}}$, called time-ahead μ*Stipula*, is the fragment where every time expression of the events is `now + k`, with $\mathtt{k} > 0$;

μ*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 $\mathtt{@Q\ f}\,\{\,W\,\} \Rightarrow \mathtt{@Q'}$ and event $\mathtt{t} \gg \mathtt{@Q''} \Rightarrow \mathtt{@Q'''}$ in a contract, we impose $\mathtt{Q} \neq \mathtt{Q''}$;

μ*Stipula*$^{\mathtt{DI}}$, called determinate-instantaneous μ*Stipula*, is the intersection between μ*Stipula*$^{\mathtt{D}}$ and μ*Stipula*$^{\mathtt{I}}$; *i.e.*, for each function $\mathtt{@Q\ f}\,\{\,W\,\} \Rightarrow \mathtt{@Q'}$ and event $\mathtt{t} \gg \mathtt{@Q''} \Rightarrow \mathtt{@Q'''}$ in a contract, we impose $\mathtt{Q} \neq \mathtt{Q''}$ and $\mathtt{t} = \mathtt{now} + \mathtt{0}$.

3 Undecidability Results

To show the undecidability of state reachability, we rely on a reduction technique from a Turing-complete model to μ*Stipula*$^{\mathtt{I}}$, μ*Stipula*$^{\mathtt{TA}}$, and μ*Stipula*$^{\mathtt{D}}$. The Turing-complete models we consider are the Minsky machines [27]. 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:

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

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

$\mathtt{stipulaI_M\{initStart\ \ F_M\}}$

where F_M contains the functions

– $\mathtt{@Start\ fstart\ \{\ now \gg\ @aQ_0\ \Rrightarrow\ @bQ_0\ \}\ \Rrightarrow\ @Q_0}$

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

```
@Q fincQ { now ≫ @dec_i ⇛ @ackdec_i
           now ≫ @bQ ⇛ @Q′
           now ≫ @aQ′ ⇛ @bQ′
} ⇛ @aQ
```

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

```
@Q fdecQ { now ≫ @ackdec_i ⇛ @aQ          @Q fzeroQ { now ≫ @zero_i ⇛ @aQ
           now ≫ @bQ ⇛ @Q″                            now ≫ @bQ ⇛ @Q′
           now ≫ @aQ″ ⇛ @bQ″                          now ≫ @aQ′ ⇛ @bQ′
} ⇛ @dec_i                                } ⇛ @dec_i
```

– $\mathtt{@dec_1\ fdec1\ \{\ \}\ \Rrightarrow\ @zero_1}$ and $\mathtt{@dec_2\ fdec2\ \{\ \}\ \Rrightarrow\ @zero_2}$

– $\mathtt{Q} : \mathit{Inc}(\mathtt{R}_i, \mathtt{Q}')$: 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 [27]. In the rest of the section, we will demonstrate the undecidability of state reachability for μ*Stipula*$^{\mathtt{I}}$, μ*Stipula*$^{\mathtt{TA}}$, and μ*Stipula*$^{\mathtt{D}}$ by providing encodings of Minsky machines. The encodings we use are increasingly complex. Therefore, we will present the undecidability results starting from the simplest one.

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

Table 2 defines the encoding of a Minsky machine M into a μ*Stipula*$^{\mathtt{I}}$ 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 $0 \gg \mathtt{@dec_1} \Rrightarrow \mathtt{@ackdec_1}$ and $0 \gg \mathtt{@dec_2} \Rrightarrow \mathtt{@ackdec_2}$ 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 $\mathtt{fincQ}$ 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 $0 \gg$ `@dec`$_1 \Rightarrow$ `@ackdec`$_1$ (or $0 \gg$ `@dec`$_2 \Rightarrow$ `@ackdec`$_2$). The function `fdecQ` has the effect of consuming one instance of the event $0 \gg$ `@dec`$_1 \Rightarrow$ `@ackdec`$_1$ (resp. $0 \gg$ `@dec`$_2 \Rightarrow$ `@ackdec`$_2$), by entering the state `@dec`$_1$ (resp. `@dec`$_2$) which triggers such event. Also the function `zeroQ` enters in one of the states `@dec`$_i$, but in this case the computation will have the ability to continue only if the state `@zero`$_i$ will be reached (this because the produced event $0 \gg$ `@zero`$_i \Rightarrow$ `@aQ` must be triggered to allow the computation to continue). But `@zero`$_i$ can be reached only if no event $0 \gg$ `@dec`$_i$ is present, because only in this case the function `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 the corresponding register is empty. In this case, the simulation gets stuck because the state `@ackdec`$_i$, necessary to trigger the event $0 \gg$ `@ackdec`$_i \Rightarrow$ `@aQ`, cannot be reached. Similarly, if `fzeroQ` is invoked when the corresponding register is nonempty the state `@zero`$_i$, necessary to trigger the event $0 \gg$ `@zero`$_i \Rightarrow$ `@aQ`, cannot be reached. Also the elapsing of time is problematic because the events $0 \gg$ `@dec`$_1 \Rightarrow$ `@ackdec`$_1$ and $0 \gg$ `@dec`$_2 \Rightarrow$ `@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 "management event" $0 \gg$ `aQ` $\Rightarrow$ `bQ` is erased, which is necessary to model the transition from a state `Q` of the Minsky machine to the next one.

Theorem 1. *State reachability is undecidable in* μ*Stipula*$^{\mathtt{I}}$.

3.2 Undecidability Results for μStipula$^{\mathtt{TA}}$

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 `now` $+ 1 \gg$ `@dec`$_i \Rightarrow$ `@ackdec`$_i$ (assuming to be in a machine state). Therefore the instruction `Q` : *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 `now` $+ 1 \gg$ `@Q'` $\Rightarrow$ `@end` that makes the contract transit to `end` if it is still in `Q'` at the beginning of the next time-clock.

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

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

$$\mathtt{stipulaTA_M}\{\mathtt{initQ_0}\ \ \mathtt{F_M}\}$$

with F_M containing the functions

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

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

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

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

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

$$\begin{array}{l} \mathtt{@wait\ \ fwait}\ \{\ \}\ \Rightarrow\ \mathtt{@end} \\ \mathtt{@dec_1\ \ fdec1}\ \{\ \}\ \Rightarrow\ \mathtt{@dec_2} \quad \text{and} \quad \mathtt{@dec_2\ \ fdec2}\ \{\ \}\ \Rightarrow\ \mathtt{@next} \\ \mathtt{@ackdec}_i\ \ \mathtt{fackdec\_i}\ \{\ \mathtt{now} + 2 \gg \mathtt{@dec}_i \Rightarrow \mathtt{@ackdec}_i\}\ \Rightarrow\ \mathtt{@dec}_i \\ \mathtt{@nextQ}_i\ \ \mathtt{fnextQ\_i}\ \{\ \mathtt{now} + 1 \gg \mathtt{@next} \Rightarrow \mathtt{@Q}\}\ \Rightarrow\ \mathtt{@dec}_i \end{array}$$

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 $\mathtt{fdecQ}$ occurs and $i = 1$. Then a bunch of events are created (see the body of $\mathtt{fdecQ}$ in Table 3) and the contract transits into $\mathtt{wait}$. 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 moving ahead all the events $0 \gg \mathtt{dec}_1 \Rightarrow \mathtt{ackdec}_1$ (except one) and $0 \gg \mathtt{dec}_2 \Rightarrow \mathtt{ackdec}_2$ 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 event $0 \gg \mathtt{dec}_1 \Rightarrow \mathtt{ackdec}_1$ 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 $\texttt{dec}_1$ happens and the event $1 \gg \texttt{next} \Rightarrow \texttt{Q}''$ is created. When the transfer protocol terminates, this event will make the contract transit to $\texttt{Q}''$;
4. at this stage the transfer of events $0 \gg \texttt{dec}_i \Rightarrow \texttt{ackdec}_i$ occurs. At first the protocol moves the events $0 \gg \texttt{dec}_1 \Rightarrow \texttt{ackdec}_1$ (every such event is fired and then the function $\texttt{fackdec}_1$ that recreates the same event at $\texttt{now} + 2$ is executed) then the function $\texttt{fdec}_1$ is invoked and the same protocol is applied to the events $0 \gg \texttt{dec}_2 \Rightarrow \texttt{ackdec}_2$;
5. at the end of the transfers, the function $\texttt{fdec}_2$ 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 $2 \gg \texttt{dec}_i \Rightarrow \texttt{ackdec}_i$ that were transferred at step 4). Then $0 \gg \texttt{next} \Rightarrow \texttt{Q}''$ that was created at step 3 is executed and the contract transits to $\texttt{Q}''$.

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 $0 \gg \texttt{dec}_1 \Rightarrow \texttt{end}$). If, on the contrary, the invoked function is `fzeroQ`, the same protocol as above is used to transfer the events $0 \gg \texttt{dec}_i \Rightarrow \texttt{ackdec}_i$ (in this case with $i = 2$ only) and the contract reaches the step 5 where, after a [Tick], the event $\texttt{now} + 2 \gg \texttt{next} \Rightarrow \texttt{Q}'$ can be executed. The undecidability result for $\mu\mathit{Stipula}^{\texttt{TA}}$ follows.

Theorem 2. *State reachability is undecidable in* $\mu\mathit{Stipula}^{\texttt{TA}}$*.*

3.3 Undecidability Results for μStipula$^{\texttt{D}}$

Also in this case we reduce from the halting problem of Minsky machines to state reachability in $\mu\mathit{Stipula}^{\texttt{D}}$. In $\mu\mathit{Stipula}^{\texttt{D}}$, 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 $\mu\mathit{Stipula}^{\texttt{D}}$ 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 $\texttt{dec}_1 \Rightarrow \texttt{ackdec}_1$ and $\texttt{dec}_2 \Rightarrow \texttt{ackdec}_2$ that model registers' units. In particular, if the contract is in a (machine) state `Q` at time 0 then $\texttt{dec}_1 \Rightarrow \texttt{ackdec}_1$ are at time 1 and $\texttt{dec}_2 \Rightarrow \texttt{ackdec}_2$ are at time 3. At times 2 and 4 management states 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. Therefore, *transfer1-state* and *transfer2-state* manage the transfer of $\texttt{dec}_1 \Rightarrow \texttt{ackdec}_1$ and $\texttt{dec}_2 \Rightarrow \texttt{ackdec}_2$ ahead five clock units.

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

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

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

```
stipula D_M { init Start   F_M }
```

with F_M contains the functions

– `@Start fstart { now ≫ @notickA ⇒ @cont } ⇒ @Q_0`

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

```
@Q : fAincQ {
    now + 1 ≫ @dec_1 ⇒ @ackdec_1
    now ≫ @cont ⇒ @Q'
    now ≫ @notickB ⇒ @cont
} ⇒ @notickA

@Q : fBincQ {
    now + 1 ≫ @dec_1 ⇒ @ackdec_1
    now ≫ @cont ⇒ @Q'
    now ≫ @notickA ⇒ @cont
} ⇒ @notickB
```

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

```
@Q : fAincQ {
    now + 3 ≫ @dec_2 ⇒ @ackdec_2
    now ≫ @cont ⇒ @Q'
    now ≫ @notickB ⇒ @cont
} ⇒ @notickA

@Q : fBincQ {
    now + 3 ≫ @dec_2 ⇒ @ackdec_2
    now ≫ @cont ⇒ @Q'
    now ≫ @notickA ⇒ @cont
} ⇒ @notickB
```

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

```
@Q : fAdecQ {
    now + 1 ≫ @ackdec_1 ⇒ @Q''_start1
    now + 1 ≫ @s1notick ⇒ cont
    now ≫ @cont ⇒ @dec_1
} ⇒ @notickA

@Q : fBdecQ {
    now + 1 ≫ @ackdec_1 ⇒ @Q''_start1
    now + 1 ≫ @s1notick ⇒ @cont
    now ≫ @cont ⇒ @dec_1
} ⇒ @notickB

@Q : fAzeroQ {
    now ≫ @cont ⇒ @dec_1
    now + 2 ≫ @dec_1 ⇒ @dec_2
    now + 3 ≫ @ackdec_2 ⇒ @copy_2
    now + 3 ≫ @c2notickA ⇒ @cont
    now + 4 ≫ @dec_2 ⇒ @Q'
    now + 5 ≫ @notickB ⇒ @cont
} ⇒ @notickA

@Q : fBzeroQ {
    now ≫ @cont ⇒ @dec_1
    now + 2 ≫ @dec_1 ⇒ @dec_2
    now + 3 ≫ @ackdec_2 ⇒ @copy_2
    now + 3 ≫ @c2notickA ⇒ @cont
    now + 4 ≫ @dec_2 ⇒ @Q'
    now + 5 ≫ @notickA ⇒ @cont
} ⇒ @notickB
```

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

```
@Q : fAdecQ {
    now ≫ @cont ⇒ @dec_1
    now + 1 ≫ @ackdec_1 ⇒ @copy_1
    now + 1 ≫ @c1notickA ⇒ cont
    now + 2 ≫ @dec_1 ⇒ dec_2
    now + 3 ≫ @ackdec_2 ⇒ @Q''_start2
    now + 3 ≫ @s2notick ⇒ @cont
} ⇒ @notickA

@Q : fBdecQ {
    now ≫ @cont ⇒ @dec_1
    now + 1 ≫ @ackdec_1 ⇒ @copy_1
    now + 1 ≫ @c1notickA ⇒ cont
    now + 2 ≫ @dec_1 ⇒ dec_2
    now + 3 ≫ @ackdec_2 ⇒ @Q''_start2
    now + 3 ≫ @s2notick ⇒ @cont
} ⇒ @notickB

@Q : fAzeroQ {
    now ≫ @cont ⇒ @dec_1
    now + 1 ≫ @ackdec_1 ⇒ @copy_1
    now + 1 ≫ @c1notickA ⇒ @cont
    now + 2 ≫ @dec_1 ⇒ @dec_2
    now + 4 ≫ @dec_2 ⇒ @Q'
    now + 5 ≫ @notickB ⇒ @cont
} ⇒ @notickA

@Q : fBzeroQ {
    now ≫ @cont ⇒ @dec_1
    now + 1 ≫ @ackdec_1 ⇒ @copy_1
    now + 1 ≫ @c1notickA ⇒ @cont
    now + 2 ≫ @dec_1 ⇒ @dec_2
    now + 4 ≫ @dec_2 ⇒ @Q'
    now + 5 ≫ @notickA ⇒ @cont
} ⇒ @notickB
```

– the management functions in Table 5.

Table 5. The management functions of Table 4 ($\mathtt{Q}$ is a state of the Minsky machine)

$\mathtt{@Q\_start1} : \mathtt{fQstart1}\ \{$ $\quad \mathtt{now} \gg \mathtt{@ackdec}_1 \Rightarrow \mathtt{@copy}_1$ $\quad \mathtt{now} \gg \mathtt{@cont} \Rightarrow \mathtt{@dec}_1$ $\quad \mathtt{now} \gg \mathtt{@c1notickA} \Rightarrow \mathtt{@cont}$ $\quad \mathtt{now} + 1 \gg \mathtt{@dec}_1 \Rightarrow \mathtt{@dec}_2$ $\quad \mathtt{now} + 2 \gg \mathtt{@ackdec}_2 \Rightarrow \mathtt{@copy}_2$ $\quad \mathtt{now} + 2 \gg \mathtt{@c2notickA} \Rightarrow \mathtt{@cont}$ $\quad \mathtt{now} + 3 \gg \mathtt{@dec}_2 \Rightarrow \mathtt{@Q}$ $\quad \mathtt{now} + 4 \gg \mathtt{@notickA} \Rightarrow \mathtt{@cont}$ $\} \Rightarrow \mathtt{@s1notick}$	$\mathtt{@Q\_start2} : \mathtt{fQstart2}\ \{$ $\quad \mathtt{now} \gg \mathtt{@ackdec}_2 \Rightarrow \mathtt{@copy}_2$ $\quad \mathtt{now} \gg \mathtt{@cont} \Rightarrow \mathtt{@dec}_2$ $\quad \mathtt{now} \gg \mathtt{@c2notickA} \Rightarrow \mathtt{@cont}$ $\quad \mathtt{now} + 1 \gg \mathtt{@dec}_2 \Rightarrow \mathtt{@Q}$ $\quad \mathtt{now} + 2 \gg \mathtt{@notickA} \Rightarrow \mathtt{@cont}$ $\} \Rightarrow \mathtt{@s2notick}$
$\mathtt{@copy}_1 : \mathtt{fAcopy1}\ \{$ $\quad \mathtt{now} \gg \mathtt{@ackdec}_1 \Rightarrow \mathtt{@copy}_1$ $\quad \mathtt{now} \gg \mathtt{@cont} \Rightarrow \mathtt{@dec}_1$ $\quad \mathtt{now} \gg \mathtt{@c1notickB} \Rightarrow \mathtt{@cont}$ $\quad \mathtt{now} + 5 \gg \mathtt{@dec}_1 \Rightarrow \mathtt{@ackdec}_1$ $\} \Rightarrow \mathtt{@c1notickA}$	$\mathtt{@copy}_1 : \mathtt{fBcopy1}\ \{$ $\quad \mathtt{now} \gg \mathtt{@ackdec}_1 \Rightarrow \mathtt{@copy}_1$ $\quad \mathtt{now} \gg \mathtt{@cont} \Rightarrow \mathtt{@dec}_1$ $\quad \mathtt{now} \gg \mathtt{@c1notickA} \Rightarrow \mathtt{@cont}$ $\quad \mathtt{now} + 5 \gg \mathtt{@dec}_1 \Rightarrow \mathtt{@ackdec}_1$ $\} \Rightarrow \mathtt{@c1notickB}$
$\mathtt{@copy}_2 : \mathtt{fAcopy2}\ \{$ $\quad \mathtt{now} \gg \mathtt{@ackdec}_2 \Rightarrow \mathtt{@copy}_2$ $\quad \mathtt{now} \gg \mathtt{@cont} \Rightarrow \mathtt{@dec}_2$ $\quad \mathtt{now} \gg \mathtt{@c2notickB} \Rightarrow \mathtt{@cont}$ $\quad \mathtt{now} + 5 \gg \mathtt{@dec}_2 \Rightarrow \mathtt{@ackdec}_2$ $\} \Rightarrow \mathtt{@c2notickA}$	$\mathtt{@copy}_2 : \mathtt{fBcopy2}\ \{$ $\quad \mathtt{now} \gg \mathtt{@ackdec}_2 \Rightarrow \mathtt{@copy}_2$ $\quad \mathtt{now} \gg \mathtt{@cont} \Rightarrow \mathtt{@dec}_2$ $\quad \mathtt{now} \gg \mathtt{@c2notickA} \Rightarrow \mathtt{@cont}$ $\quad \mathtt{now} + 5 \gg \mathtt{@dec}_2 \Rightarrow \mathtt{@ackdec}_2$ $\} \Rightarrow \mathtt{@c2notickB}$

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. Therefore, 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 $\mu\mathit{Stipula}^{\mathsf{D}}$ 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 $\mu\mathit{Stipula}^{\mathsf{D}}$ 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 $R1$ are at $\mathtt{now} + 1$):

$$
\begin{array}{ll}
\texttt{@Q} : \texttt{fAincQ}\,\{ & \texttt{@Q} : \texttt{fBincQ}\,\{ \\
\quad \texttt{now} + 1 \gg \texttt{@dec}_1 \Rightarrow \texttt{@ackdec}_1 & \quad \texttt{now} + 1 \gg \texttt{@dec}_1 \Rightarrow \texttt{@ackdec}_1 \\
\quad \texttt{now} \gg \texttt{@cont} \Rightarrow \texttt{@Q}' & \quad \texttt{now} \gg \texttt{@cont} \Rightarrow \texttt{@Q}' \\
\quad \texttt{now} \gg \texttt{@notickB} \Rightarrow \texttt{@cont} & \quad \texttt{now} \gg \texttt{@notickA} \Rightarrow \texttt{@cont} \\
\} \Rightarrow \texttt{@notickA} & \} \Rightarrow \texttt{@notickB}
\end{array}
$$

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

Tables 4 and 5 define the encoding of a Minsky machine M into a *μStipula*$^{\texttt{D}}$ contract $\texttt{D}_M$. The reader may notice that the management functions $\texttt{fQstart1}$ and $\texttt{fQstart2}$ in Table 5 do not have sibling functions – they always produce a management event $\texttt{k} \gg \texttt{notickA} \Rightarrow \texttt{cont}$ (with $\texttt{k}$ 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 *μStipula*$^{\texttt{D}}$ follows.

Theorem 3. *State reachability is undecidable in μStipula*$^{\texttt{D}}$*.*

4 Decidability Results for μStipula$^{\texttt{DI}}$

We demonstrate that state reachability is decidable for *μStipula*$^{\texttt{DI}}$ by reasoning on a variant with an alternative [Tick] rule. We recall that, in *μStipula*$^{\texttt{DI}}$, for every $\texttt{Q f Q}'$, $\texttt{Q}'' \texttt{ ev Q}''' \in \texttt{C}$, we have $\texttt{Q} \neq \texttt{Q}''$

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

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

That is, unlike [Tick], [Tick-Plus] may only be used in states that are not initial states of events. Let *μStipula*$^{\texttt{DI}}_{+}$ be the language whose operational semantics uses [Tick-Plus] instead of [Tick]; we denote with $\longrightarrow_{\texttt{tp}}$ the transition relation of *μStipula*$^{\texttt{DI}}_{+}$. We observe that, syntactically, nothing is changed: every *μStipula*$^{\texttt{DI}}$ contract is a *μStipula*$^{\texttt{DI}}_{+}$ 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 2. *Let $\mathbb{C}$ be a possible configuration of a μStipula 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* $\mathtt{C}(\mathtt{Q}\,,\,\Sigma\,,\,\Psi)$ *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* $\Sigma \neq \_$*:*

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

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

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

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

$$\mathtt{C}(\mathtt{Q}\,,\,\_\,,\,\Psi) \;\textit{is stuck} \quad \textit{if and only if} \quad \mathtt{C}(\mathtt{Q}\,,\,\_\,,\,\Psi) \not\longrightarrow_{\mathtt{tp}}\,.$$

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

We begin with some background on well-structured transition systems [18]. 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 3. *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.*, $\uparrow \mathbb{C} = \{\,\mathbb{C}' \mid \mathbb{C} \preceq \mathbb{C}'\,\}$). 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 [18]. With abuse of notation, we will denote with $\mathit{Pred}(\cdot)$ also its natural extension to sets of configurations.

Several properties are decidable for well-structured transition systems (under some conditions discussed below) [2,18], we will consider the following one.

Definition 4. *Let* $(\mathcal{C}, \longrightarrow, \preceq)$ *be a well-structured transition system. The* coverability problem *is to decide, given the initial configuration* $\mathbb{C}_{\mathtt{init}} \in \mathcal{C}$ *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* $\mathbb{C}_{\mathtt{init}} \longrightarrow^* \mathbb{C}'$*.*

In well-structured transition systems the coverability problem is decidable when the transition relation $\longrightarrow$, the ordering $\preceq$ and a finite-basis for the set of configurations $Pred(\uparrow \mathbb{C})$ 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

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

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

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

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

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

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

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

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

Theorem 4. *The state reachability problem is decidable in μStipula$^{\mathtt{DI}}_{+}$.*

As a direct consequence of Proposition 1 and Theorem 4 we have:

Corollary 1. *The state reachability problem is decidable in μStipula$^{\mathtt{DI}}$.*

5 Related Work

The decidability of problems about infinite-state systems has been largely addressed in the literature. We refer to [2] for an overview of the research area.

It turns out that critical features of μ*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., [16]), the above variants of Petri nets are Turing complete, therefore all non-trivial properties become undecidable [15].

It is not obvious whether Petri nets with inhibitor and reset arcs may be modelled by *μStipula* 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 *μStipula*. 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 [26] 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 [18] (for which, under certain computability conditions, the reachability and termination problems are decidable, see Sect. 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 *μStipula* 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) [1,5], 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 [3]. 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 [19]. *μStipula* 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 [22,29]. 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 [20].

6 Conclusions

We have systematically studied the computational power of *μStipula*, 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 μ*Stipula* a problem such as reachability of state is undecidable. The decidable fragment, μ*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 μ*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 μ*Stipula*$^{\mathtt{DI}}$. Another interesting line of research regards the investigation of sound, but incomplete, algorithms for checking state reachability in μ*Stipula*. A preliminary algorithm was investigated in [24]. 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. Abdulla, P.A., Nylén, A.: Timed petri nets and bqos. In: ICATPN'01, LNCS, vol. 2075, pp. 53–70. Springer (2001)
2. Abdulla, P.A., Cerans, K., Jonsson, B., Tsay, Y.K.: General decidability theorems for infinite-state systems. In: Proceedings of the11th Annual IEEE Symposium on Logic in Computer Science, pp. 313–321. IEEE Computer Society (1996)
3. Abdulla, P.A., Delzanno, G., Rezine, O., Sangnier, A., Traverso, R.: Parameterized verification of time-sensitive models of ad HOC network protocols. Theor. Comput. Sci. **612**, 1–22 (2016)
4. Abdulla, P.A., Jonsson, B.: 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, pp. 160–170. IEEE Computer Society (1993)
5. Abdulla, P.A., Jonsson, B.: Model checking of systems with many identical timed processes. TCS **290**(1), 241–264 (2003)
6. Aho, A.V., Lam, M.S., Sethi, R., Ullman, J.D.: Compilers: Principles, Techniques, and Tools, 2nd edn. Addison Wesley, Boston, MA, USA (2006)
7. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. **126**(2), 183–235 (1994)
8. Amadio, R.M., Meyssonnier, C.: On decidability of the control reachability problem in the asynchronous pi-calculus. Nord. J. Comput. **9**(2), 70–101 (2002)
9. Appel, A.W., Ginsburg, M.: Modern Compiler Implementation in C. Cambridge University Press, Cambridge, UK (1997)
10. Cécé, G., Finkel, A., Iyer, S.P.: Unreliable channels are easier to verify than perfect channels. Inf. Comput. **124**(1), 20–31 (1996)
11. Crafa, S., Laneve, C.: 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, pp. 129–146. Springer, Cham (2022)

12. Crafa, S., Laneve, C., Sartor, G., Veschetti, A.: Pacta sunt servanda: legal contracts in Stipula. Sci. Comput. Program. **225**, 102911 (2023)
13. De Boer, F. S., Jaghoori, M.M., Laneve, C., Zavattaro, G.: Decidability problems for actor systems. Log. Methods Comput. Sci. **10**(4) (2014)
14. Delzanno, G., Laneve, C., Sangnier, A., Zavattaro, G.: Decidability problems for micro-Stipula (2025.) arXiv:2504.16703
15. Dufourd, C., Finkel, A., Schnoebelen, P.: Reset nets between decidability and undecidability. In: Larsen, K.G., Skyum, S., Winskel, G. (eds.) ICALP 1998. LNCS, vol. 1443, pp. 103–115. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0055044
16. Esparza, J., Nielsen, M.: Decidability issues for petri nets - a survey. J. Inf. Process. Cybern. **30**(3), 143–160 (1994)
17. Evangelisti, S.: Stipula reachability analyzer (2024). Available on github: https://github.com/stipula-language
18. Finkel, A., Schnoebelen, P.: Well-structured transition systems everywhere! Theor. Comput. Sci. **256**, 63–92 (2001)
19. Finkel, A., Raskin, J.-F., Samuelides, M., Van Begin, L.: Monotonic extensions of petri nets: forward and backward search revisited. Electr. Notes Theor. Comput. Sci. **68**(6), 85–106 (2002)
20. Ganty, P., Majumdar, R.: Algorithmic verification of asynchronous programs. ACM Trans. Program. Lang. Syst. **34**(1), 1–48 (2012)
21. Hansson, H., Jonsson, B.: A calculus for communicating systems with time and probabitilies. In: Proceedings of the Real-Time Systems Symposium - 1990, pp. 278–287. IEEE Computer Society, New York (1990)
22. Jhala, R., Majumdar, R.: Interprocedural analysis of asynchronous programs. In: Hofmann, M., Felleisen, M., (eds.) Proceedings of the 34th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2007, Nice, France, January 17-19, 2007, pp. 339–350. ACM (2007)
23. Karp, R.M., Miller, R.E.: Parallel program schemata. J. Comput. Syst. Sci. **3**(2), 147–195 (1969)
24. Laneve, C.: Reachability analysis in micro-Stipula. In: Proceedings of the 26th International Symposium on Principles and Practice of Declarative Programming, PPDP 2024, pp. 1–12. ACM (2024)
25. Mayr, R.: Undecidability of weak bisimulation equivalence for 1-counter processes. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J., (eds.) 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, pp. 570–583. Springer (2003)
26. Meyer, R.: On boundedness in depth in the pi-calculus. In: IFIP TCS, volume 273 of IFIP, pp. 477–489. Springer (2008)
27. Minsky, M.: Computation: Finite and Infinite Machines. Prentice Hall (1967)
28. Moller, F., Tofts, C.: A temporal calculus of communicating systems. In: Baeten, J., Klop, J.W. (eds.) CONCUR 1990. LNCS, vol. 458, pp. 401–415. Springer, Heidelberg (1990). https://doi.org/10.1007/BFb0039073
29. Sen, K., Viswanathan, M.: Model checking multithreaded programs with asynchronous atomic methods. In: Ball, T., Jones, R.B., (eds.) Computer Aided Verification, 18th International Conference, CAV 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings, volume 4144 of Lecture Notes in Computer Science, pp. 300–314. Springer (2006)