Reachability Analysis of Legal Contracts*

Cosimo Laneve
Department of Computer Science and Engineering, University of Bologna, Italy

Legal contracts are sets of clauses specifying protocols that regulate the legal interactions between different parties. These clauses may contains errors, such as defining rules that can never be applied because of unreachable circumstances or of wrong time constraints. Since these erroneous clauses may jeopardise the agreement of a contract by parties, it is worth to spot and remove them when the contract is drawn up. In this paper we analyze this issue when the contract text is written in *Stipula*, a domain specific language for designing legal contracts. In particular, we discuss both the theory of the reachability analyzer and the prototype that has been integrated in the language toolchain.

1 Introduction

Contracts are sets of clauses that define protocols regulating legal relationships between parties in terms of permissions, obligations and prohibitions. According to modern legal systems, these protocols can be expressed by the parties using the language and medium they prefer (*principle of freedom of form*) [20], including a programming language. The benefits of using programming languages are evident and have been widely recognized in several projects [14, 21, 19]: they lower the overall transaction costs involved in a contract lifecycle by enabling the identification of potential inconsistencies, reducing the complexity and ambiguity of legal texts and allowing the automatic execution of clauses. For these reasons, in [8, 9, 17] we have developed a domain-specific language for legal contracts, called *Stipula*, with a formal operational semantics [9], so that the behaviour of a contract is fully specified, and with a toolchain [10] that includes a prototype implementation, a graphical interface and an extension supporting runtime contract amendments.

This contribution intends to strengthen the *Stipula* toolchain with tools supporting safe contract drawing. Two techniques have been already studied: [9] defines a type inference system that allows to automatically derive types for fields, assets and contracts' functions, thus preventing basic errors with contract's data and assets; [16] reports an analyzer that statically checks the presence of executions leaving assets frozen into the contract without being redeemable by any party (liquidity). Now we analyze the presence of unreachable clauses in contracts, *i.e.* clauses that can never be applied because of unreachable circumstances or of wrong time constraints. In the legal contract domain, removing unreachable clauses when the contract is drawn up is substantial because they might be considered too oppressive by parties and make the legal relationship fail.

Spotting and removing unreachable code is a very common optimization in compiler construction of programming languages and the literature already reports techniques for affording this issue [1, 5]. However *Stipula* and, in general, legal contracts have an additional difficulty: the presence of clauses that are triggered by time expressions – these clauses are called *events*, in *Stipula*. If the time expressions are logically inconsistent with the contract behaviour, the event itself and its continuation become unreachable and can be safely removed.

*Supported by the SERICS project (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union – NextGenerationEU

Submitted to:
≫

To address time anomalies, we define an evaluator that computes the *logical times* of clauses by means of variables representing the times of function clauses. Then the analyzer generates constraints on logical times and verifies that they are solvable. When constraints are unsolvable, the list of the corresponding clauses (which are therefore unreachable) is returned. The critical point is when a clause can be executed cyclically because, in this case, reachability may be achieved by several instances of the same clause. In this respect, since *Stipula* contracts may display very subtle reachability dependencies, we decided to adopt a combination of time analysis and standard, untimed one: time analysis is used for acyclic clauses, while time expressions are overlooked in the other clauses.

In order to ease our arguments, in the first part of the paper we consider a lightweight version of *Stipula*, called μ*Stipula* – read *micro-Stipula* –, that already highlights all the relevant issues of the reachability analysis. We define an algorithm for the reachability analysis in μ*Stipula* and demonstrate the correctness of the algorithm. Then the technique has been extended to *Stipula* where time expressions have a further complexity: the presence of names whose values are unknown when the contract is drawn (their values will be determined upon the agreement between parties). To cope with this feature, we consider logical times with expressions on field names and the analyzer returns *constraints* that guarantee reachability. In this last case, the analyzer takes advantage of the (additive) format of *Stipula* time expressions to simplify and partially evaluate the constraints in order to obtain more intelligible (unreachability) messages. The reachability analyzer has been prototyped for the (full) *Stipula* language. The prototype, with all the code samples discussed in this paper (and many others), is available at [12].

The paper is organized as follows. Section 2 gives a light introduction to *Stipula* and to the reachability analyzer through a bunch of simple examples. The syntax and semantics of μ*Stipula* are defined in Section 3, while the theory underneath the analyzer is developed in Section 4. Section 5 covers the analysis of features of *Stipula* that are not in μ*Stipula*. We report the related literature in Section 6 and conclude in Section 7. Because of page constraints, the demonstrations of main statements have been moved to the appendix that will be removed from the extended abstract.

2 μ*Stipula* and the reachability analyzer

Let us introduce μ*Stipula* with a simple contract, the `PingPong`:

```
1 stipula PingPong {
2     init StartM
3     @StartM Mary:ping()[] {
4         now + 1 ≫ @Go { } ⇒ @StartB
5     } ⇒ @Go
6     @StartB Bob:pong()[] {
7         now + 2 ≫ @Cont { } ⇒ @StartM
8     } ⇒ @Cont
9 }
```

The contract follows a *state-machine programming* style for modelling the normative elements regulating the interactions between parties. In this case there are two parties, `Mary` and `Bob`. In the initial state `StartM`, once `Mary` invokes the function `ping`, the contract transits to the state `Go` where the unique possibility is the execution of the event at line `4`. The expression `now + 1` indicates that this clause can be executed after `1` minute (one clock tick must elapse). Then the state becomes `StartB` indicating that `pong` may be invoked by `Bob`, thus letting the contract transit to `Cont`. In `Cont`, after `2` minutes (the expression `now + 2`), the event at line `7` can be executed and the contract returns to `StartM`. In `PingPong`, every clause is reachable; in fact the analyzer returns

```
1  stipula SampleTime {
2     init Init
3     @Init A:f()[] {
4        now + 1 ≫ @Cont { } ⇒ @Run
5        now + 2 ≫ @Comp { } ⇒ @End
6     } ⇒ @Cont
7
8     @Run B:g()[] {
9        now + 2 ≫ @Go { } ⇒ @Comp
10    } ⇒ @Go
11 }
```

```
1  stipula Ugly {
2     init Q0
3     @Q0 A: f()[] {
4        now + 1 ≫ @Q3 {} ⇒ @Q4
5        now + 2 ≫ @Q2 {} ⇒ @Q3
6     } ⇒ @Q1
7     @Q1 B: g()[] {
8        now + 1 >> @Q1 {} ⇒ @Q2
9     } ⇒ @Q0
10    @Q4 C: h()[] { } ⇒ @Q5
11 }
```

Figure 1: The contracts `SampleTime` and `Ugly`

```
"unreachable_code": []
```

Next, let us consider the contract `Sample`:

```
1  stipula Sample {
2     init Init
3     @Init A:f()[] {
4        now + 5 ≫ @Go { } ⇒ @End
5     } ⇒ @Run
6     @Init B:g()[] { } ⇒ @Go
7  }
```

This contract has the two functions at lines 3 and 6, noted `A.f` and `B.g`, respectively, that may be invoked in `Init`. We observe that the invocation of one of them excludes the other because of their final states. Therefore the event in line 4 is unreachable since it can run only if `B.g` is executed (because the final state of `B.g` matches with the initial state of the event). Therefore the analyzer returns:

```
"unreachable_code": [ Go ev.4 End ] .
```

The difficult cases are those where unreachability is due to time expressions that are incompatible. Consider `SampleTime` in Figure 1. This contract has an unreachable event: the one at line 5. To explain this, let us discuss the flow of execution by computing the times of the clauses. Let the time of `A.f` be $\zeta_{\texttt{A.f}}$; therefore the times of `ev.4` and `ev.5` are $\zeta_{\texttt{A.f}}+1$ and $\zeta_{\texttt{A.f}}+2$, respectively. Because of the matching final state/initial state, the clause that is executed after `A.f` is `ev.4` and then `B.g`. Therefore `B.g` may start at $\zeta_{\texttt{A.f}}+\zeta_{\texttt{B.g}}+1$ and `ev.9` may be triggered at $\zeta_{\texttt{A.f}}+\zeta_{\texttt{B.g}}+3$. Since the final state of `ev.9` matches with the initial state of `ev.5`, we derive the constraint on time expressions $\zeta_{\texttt{A.f}}+\zeta_{\texttt{B.g}}+3 \leq \zeta_{\texttt{A.f}}+2$. This constraint is clearly *unsolvable* because all the time variables must be nonnegative. Said otherwise, when `ev.5` might be executed, the time is already elapsed. Hence the analyzer gives

```
"unreachable_code": [ Comp ev.5 End ] .
```

The above arguments may lead to wrong conclusions when contracts have cycles. For example, in the contract `Ugly`, the `ev.4` seems unreachable because it is caused by `ev.5` that happens at a later time (as a consequence `C.h` is unreachable, as well). In fact, we derive the constraint $\zeta_{\texttt{A.f}}+\zeta_{\texttt{B.g}}+3 \leq \zeta_{\texttt{A.f}}+\zeta_{\texttt{B.g}}+2$, which is unsolvable. However, consider the following flow of execution assuming that the contract starts at time 0. The first clause that is executed is `A.f`, which creates `ev.4` to be executed at time 1 (we note it $\texttt{ev.4}_1$) and `ev.5` to be executed at time 2 (we note it $\texttt{ev.5}_2$). `A.f` ends in `Q1` and the next clause that can be executed is `B.g`, still at time 0. `B.g` creates `ev.8` to be executed at 1 and the state go back to `Q0`. Now there is the critical moment: a tick occurs and `A.f` is executed again at time 1. `A.f` creates `ev.4` to

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

$$\begin{array}{llll} \textit{Functions} & F & ::= & \_ \mid \texttt{@Q A:f()[]}\{W\} \Rrightarrow \texttt{@Q}' \ F \\ \textit{Events} & W & ::= & \_ \mid \texttt{t} \gg \texttt{@Q}\{\,\} \Rrightarrow \texttt{@Q}' \ W \\ \textit{Time expressions} & \texttt{t} & ::= & \texttt{now} + \kappa \qquad (\kappa \in \mathsf{Nat}) \end{array}$$

Figure 2: Syntax of μ*Stipula*

be executed at time 2 (we note it $\texttt{ev.4}_2$) and ev.5 to be executed at time 3 (we note it $\texttt{ev.5}_3$). Then we can execute ev.8, a tick, $\texttt{ev.5}_2$, $\texttt{ev.4}_2$ and C.h. That is, C.h is reached by events created in two different invocations of $\texttt{A}.f$. Hence, every clause of Ugly is reachable. Therefore, when a function is cyclic, we have decided to drop the time analysis and to deem "reachable" its events if they are so with arguments that do not take into account time expressions.

3 The syntax and semantics of μ*Stipula*

A legal contract is written in μ*Stipula* according to the syntax of Figure 2, where C is the name of the contract. Contracts have

- a set of *parties*, ranged over A, B, $\texttt{A}', \cdots$, which are the entities involved in the contract;
- a set of *states*, ranged over Q, $\texttt{Q}', \cdots$; the initial state is defined by the init expression;
- a sequence F of *functions* f, g, $\cdots$.

A μ*Stipula* contract may transit from one state to another either by invoking a *function* or by running an *event*. Functions are invoked by a *party* and define the state when the invocation is admitted.

Events W are sequences of *timed continuations* that schedule some code for future execution. More precisely, the term $\texttt{t} \gg \texttt{@Q}\{\,\} \Rrightarrow \texttt{@Q}'$ schedules an execution that is triggered at a time that is the value of t. When triggered, if the contract's state is Q, a transition to $\texttt{Q}'$ occurs. The time expressions are additions $\texttt{now} + \kappa$, where κ is a natural constant (representing *minutes*); now is a place-holder that will be replaced by the current global time during the execution, see rule [STATE-CHANGE] in Figure 3. We always shorten $\texttt{now} + 0$ into now.

Restriction and notations. We assume that *a function is uniquely determined by the tuple* $\texttt{Q A.f Q}'$, that is the initial and final states, the party that can invoke the function and the function name. Similarly, an event is uniquely determined by the tuple $\texttt{Q ev.n Q}'$, where n is the line-code of the event. We use H.c to range over A.f and ev.n and tuples $\texttt{Q H.c Q}'$ are called *clauses*.

With an abuse of notation, the contract code is addressed by using the contract name and we write $\texttt{Q A.f Q}' \in \texttt{C}$ if the code $\texttt{@Q A:f()[]}\{W\} \Rrightarrow \texttt{@Q}'$ is in the contract C (also noted $\texttt{@Q A:f()[]}\{W\} \Rrightarrow \texttt{@Q}' \in \texttt{C}$). We also write $\texttt{Q}_1 \ \texttt{ev.n} \ \texttt{Q}_2 \in \texttt{Q A.f Q}'$ if there is an event $\texttt{t} \gg \texttt{@Q}_1\{\,\} \Rrightarrow \texttt{@Q}_2$ in the function $\texttt{@Q A:f()[]}\{W\} \Rrightarrow \texttt{@Q}'$ that starts at line-code n. In this case we also write $\texttt{Q}_1 \ \texttt{ev.n} \ \texttt{Q}_2 \in \texttt{C}$.

3.1 The operational semantics

The meaning of μ*Stipula* primitives is defined operationally by means of a transition relation. Let $\texttt{C}(\texttt{Q}, \Sigma, \Psi)$ be a tuple where

- C is the contract name;

$$\frac{\texttt{@Q A:f ()}\ [\,]\,\{W\} \Rightarrow \texttt{@Q}' \in \texttt{C} \quad W' = \texttt{LC}_{\texttt{Q A.f Q}'}(W) \quad \Psi, \mathbb{t} \not\rightarrow}{\texttt{C}(\texttt{Q}, \_, \Psi), \mathbb{t} \xrightarrow{\texttt{A.f}} \texttt{C}(\texttt{Q}, W' \Rightarrow \texttt{Q}', \Psi), \mathbb{t}}\ [\textsc{Function}]$$

$$\frac{W = \left(\texttt{t}_i \gg_{\texttt{n}_i} \texttt{@Q}_i\,\{\,\} \Rightarrow \texttt{@Q}'_i\right)^{i\in 1..h} \quad \left(\mathbb{t}_i = \texttt{t}_i\{{}^{\mathbb{t}}/_{\texttt{now}}\}\right)^{i\in 1..h} \quad \Psi' = \mathbb{t}_1 \gg_{\texttt{n}_1} \texttt{Q}_1\,\{\,\} \Rightarrow \texttt{Q}'_1 \mid \cdots \mid \mathbb{t}_h \gg_{\texttt{n}_h} \texttt{Q}_h\,\{\,\} \Rightarrow \texttt{Q}'_h}{\texttt{C}(\texttt{Q}, W \Rightarrow \texttt{Q}', \Psi), \mathbb{t} \longrightarrow \texttt{C}(\texttt{Q}', \_, \Psi' \mid \Psi), \mathbb{t}}\ [\textsc{State-Change}]$$

$$\frac{\Psi = \mathbb{t} \gg_{\texttt{n}} \texttt{Q}\ \{\,\} \Rightarrow \texttt{Q}' \mid \Psi'}{\texttt{C}(\texttt{Q}, \_, \Psi), \mathbb{t} \xrightarrow{\texttt{ev.n}} \texttt{C}(\texttt{Q}, \_ \Rightarrow \texttt{Q}', \Psi'), \mathbb{t}}\ [\textsc{Event-Match}]$$

$$\frac{\Psi, \mathbb{t} \not\rightarrow}{\texttt{C}(\texttt{Q}, \_, \Psi), \mathbb{t} \longrightarrow \texttt{C}(\texttt{Q}, \_, \Psi), \mathbb{t}+1}\ [\textsc{Tick}]$$

Figure 3: The operational semantics of μ*Stipula*

- `Q` is the current state of the contract;
- Σ is either `_` or a term $W \Rightarrow \texttt{Q}$;
- 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 $W_1 \mid \cdots \mid W_n$ where each $W_i = \mathbb{t}_i \gg_{\texttt{n}_i} \texttt{Q}_i\ \{\,\} \Rightarrow \texttt{Q}'_i$. The *time guard* $\mathbb{t}_i$ is the *absolute time*: it is the evaluation of the time expression $\texttt{t}_i$ of the event when `now` is replaced by the value of the global clock (se below and rule [State-Change]). The index $\texttt{n}_i$ is the *line-code* of the event; it is set by the function $\texttt{LC}_{\texttt{Q A.f Q}'}(W)$ (see rule [Function], the definition is omitted).

Tuples $\texttt{C}(\texttt{Q}, \Sigma, \Psi)$ are ranged over by $\mathbb{C}, \mathbb{C}', \cdots$. A *configuration* is a pair $\mathbb{C}, \mathbb{t}$, where $\mathbb{t}$ is the time value of the system's global clock. The transition relation of μ*Stipula* is $\mathbb{C}, \mathbb{t} \xrightarrow{\mu} \mathbb{C}', \mathbb{t}'$, where μ is either empty `_` or `A.f` or `ev.n` (the label `ev.n` indicates the event at line `n`; it has been added in this paper for easing the arguments in Section 4). The formal definition of $\mathbb{C}, \mathbb{t} \xrightarrow{\mu} \mathbb{C}', \mathbb{t}'$ is given in Figure 3 using the following auxiliary predicate:

- the predicate $\Psi, \mathbb{t} \not\rightarrow$ is *true* whenever $\Psi = \mathbb{t}_1 \gg_{\texttt{n}_1} \texttt{Q}_1\{\,\} \Rightarrow \texttt{Q}'_1 \mid \cdots \mid \mathbb{t}_k \gg_{\texttt{n}_k} \texttt{Q}_k\{\,\} \Rightarrow \texttt{Q}'_k$ and, for every $1 \leq i \leq k$, $\mathbb{t}_i \neq \mathbb{t}$; *false* otherwise.

A discussion about the four rules follows. Rule [Function] defines invocations: the label specifies the party `A` performing the invocation and the function name `f`. The transition may occur provided (*i*) the contract is in the state `Q` that admits invocations of `f` from `A` and (*ii*) no event can be triggered – *cf.* the premise $\Psi, \mathbb{t} \not\rightarrow$ (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 their time expressions have been evaluated (`now` is replaced by the current value of the clock). Rule [Event-Match] specifies that an event handler may run provided Σ is `_` and the time guard of the event has exactly the value of the global clock $\mathbb{t}$, which is evaluated when the event is scheduled – *cf.* rule [State Change]. Rule [Tick] defines the elapsing of time. This happens when the contract has an empty Σ and no event can be triggered. It is worth to observe that μ*Stipula* has three *causes for nondeterminism*: (*i*) two functions can be invoked in a state, (*ii*) either a function may be invoked or the time may elapse (in this case the function may be invoked at a later time), and (*iii*) two events may be invoked at the same time and in the same state.

The initial configuration of a μ*Stipula* contract

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

is $\texttt{C}(\texttt{Q}, \_, \_), \mathbb{t}$, where $\mathbb{t}$ can be any value because it corresponds to the absolute time. We write $\mathbb{C}, \mathbb{t} \longrightarrow^* \mathbb{C}', \mathbb{t}'$, called *computation*, if there are $\mu_1, \cdots, \mu_k$ such that $\mathbb{C}, \mathbb{t} \xrightarrow{\mu_1} \cdots \xrightarrow{\mu_k} \mathbb{C}', \mathbb{t}'$.

Definition 1 *Let* C *be a* μ*Stipula contract with initial configuration* $\mathbb{C}, \mathbb{t}$ *and let*

$$\mathbb{C}, \mathbb{t} \longrightarrow^* \xrightarrow{\mu} \mathtt{C}(\mathtt{Q}, W \Rightarrow \mathtt{Q}', \Psi), \mathbb{t}' .$$

If $\mu = \mathtt{A.f}$ *then we say that the function* $\mathtt{Q}\ \mathtt{A.f}\ \mathtt{Q}' \in \mathtt{C}$ *is* reachable *(from* $\mathbb{C}, \mathbb{t}$*); if* $\mu = \mathtt{ev.n}$ *then we say that the event* $\mathtt{Q}\ \mathtt{ev.n}\ \mathtt{Q}'$ *is* reachable *(from* $\mathbb{C}, \mathbb{t}$*; in this case* $W = \_$*).*

We remark that, according to Definition 1, the reachability predicate uses an *existential quantification* on computations. The notion of *underlying clause of a transition* will be often used in the following sections:

- the *underlying clause* of $\mathtt{C}(\mathtt{Q}, \_, \Psi), \mathbb{t} \xrightarrow{\mathtt{A.f}} \mathtt{C}(\mathtt{Q}, W \Rightarrow \mathtt{Q}', \Psi), \mathbb{t}$ is $\mathtt{Q}\ \mathtt{A.f}\ \mathtt{Q}'$; the *underlying clause* of $\mathtt{C}(\mathtt{Q}, \_, \Psi), \mathbb{t} \xrightarrow{\mathtt{ev.n}} \mathtt{C}(\mathtt{Q}, \_ \Rightarrow \mathtt{Q}', \Psi), \mathbb{t}$ is $\mathtt{Q}\ \mathtt{ev.n}\ \mathtt{Q}'$.

4 The theory of the reachability analyzer

We use *sequences of clauses without repetitions* $\mathtt{Q}_1\ \mathtt{H}_1.\mathtt{c}_1\ \mathtt{Q}'_1\ ;\ \cdots\ ;\ \mathtt{Q}_n\ \mathtt{H}_n.\mathtt{c}_n\ \mathtt{Q}'_n$. These sequences, ranged over by $\mathbb{A}, \mathbb{A}', \cdots$, are called *abstract computations* because they represents the (sequence of) clauses used in computations of a μ*Stipula* contract. The notion is quite rude: while every computation in μ*Stipula* has a corresponding abstract computation (see Theorem 1), there are abstract computations that do not correspond to any computation. For example, in the contract `Sample`, the sequence containing all the clauses does not correspond to any computation. The purpose of this section is to restrict the set of abstract computations by dropping those containing unreachable clauses.

We will use the following auxiliary operations and notions:

- $\mathtt{Q}\ \mathtt{H.c}\ \mathtt{Q}' \in \mathbb{A}$ if there is an element of $\mathbb{A}$ that is equal to $\mathtt{Q}\ \mathtt{H.c}\ \mathtt{Q}'$;
- let $\mathscr{A}$ be a set of clauses. Then

$$\mathbb{A}|_{\mathscr{A}} \stackrel{\text{def}}{=} \begin{cases} \varnothing & \text{if } \mathbb{A} = \varepsilon \\ \mathbb{A}'|_{\mathscr{A}} & \text{if } \mathbb{A} = \mathbb{A}'\ ;\ \mathtt{Q}\ \mathtt{H.c}\ \mathtt{Q}' \text{ and } \mathtt{Q}\ \mathtt{H.c}\ \mathtt{Q}' \notin \mathscr{A} \\ \mathbb{A}'|_{\mathscr{A}} \cup \{\mathtt{Q}\ \mathtt{H.c}\ \mathtt{Q}'\} & \text{if } \mathbb{A} = \mathbb{A}'\ ;\ \mathtt{Q}\ \mathtt{H.c}\ \mathtt{Q}' \text{ and } \mathtt{Q}\ \mathtt{H.c}\ \mathtt{Q}' \in \mathscr{A} \end{cases}$$

$$\mathbb{A} \lhd \mathtt{Q}\ \mathtt{H.c}\ \mathtt{Q}' \stackrel{\text{def}}{=} \begin{cases} \mathbb{A}\ ;\ \mathtt{Q}\ \mathtt{H.c}\ \mathtt{Q}' & \text{if } \mathtt{Q}\ \mathtt{H.c}\ \mathtt{Q}' \notin \mathbb{A} \\ \mathbb{A} & \text{if } \mathtt{Q}\ \mathtt{H.c}\ \mathtt{Q}' \in \mathbb{A} \end{cases}$$

  We also write $\{\mathbb{A}_1, \cdots, \mathbb{A}_n\} \lhd \mathtt{Q}\ \mathtt{H.c}\ \mathtt{Q}' \stackrel{\text{def}}{=} \{\mathbb{A}_1 \lhd \mathtt{Q}\ \mathtt{H.c}\ \mathtt{Q}', \cdots, \mathbb{A}_n \lhd \mathtt{Q}\ \mathtt{H.c}\ \mathtt{Q}'\}$.

- let $\mathbb{R}'$ and $\mathbb{R}''$ be two maps from clauses to sets of abstract computations. We define $\mathbb{R}' \leq \mathbb{R}''$ if, for every $\mathtt{Q}$, $\mathbb{R}'(\mathtt{Q}\ \mathtt{H.c}\ \mathtt{Q}') \subseteq \mathbb{R}''(\mathtt{Q}\ \mathtt{H.c}\ \mathtt{Q}')$. It turns out that the domain of maps with the partial order $\leq$ is a *lattice* [11] and this lattice, given a contract, is always *finite*.

In the following notations, in order to have a lighter notation, we always omit the reference to the contract C.

Definition 2 *Let* C *be a* μ*Stipula contract with initial state* $\mathtt{Q}$*. The sequence of maps* $\mathbb{R}_{\mathtt{Q}}{}^{(0)}$, $\mathbb{R}_{\mathtt{Q}}{}^{(1)}, \cdots$ *that take clauses in* C *and return sets of abstract computations is defined as follows:*

1. $\mathbb{R}_{\mathtt{Q}}{}^{(0)}(\mathtt{Q}_1\ \mathtt{H.c}\ \mathtt{Q}_2) = \begin{cases} \{\ \mathtt{Q}_1\ \mathtt{H.c}\ \mathtt{Q}_2\ \} & \textit{if } \mathtt{Q} = \mathtt{Q}_1 \textit{ and } \mathtt{H.c} = \mathtt{A.f} \textit{ and } \mathtt{Q}\ \mathtt{A.f}\ \mathtt{Q}_2 \in \mathtt{C} \\ \varnothing & \textit{otherwise} \end{cases}$

2. $\mathbb{R}_{\mathtt{Q}}{}^{(i+1)}(\mathtt{Q}_1\ \mathtt{A.f}\ \mathtt{Q}_2) = \mathbb{R}_{\mathtt{Q}}{}^{(i)}(\mathtt{Q}_1\ \mathtt{A.f}\ \mathtt{Q}_2) \cup \Big(\bigcup_{\mathtt{Q}_3\ \mathtt{H}'.\mathtt{c}'\ \mathtt{Q}_1\in\mathtt{C}} \{\,\mathbb{A} \lhd \mathtt{Q}_1\ \mathtt{A.f}\ \mathtt{Q}_2 \mid \mathbb{A}\in\mathbb{R}_{\mathtt{Q}}{}^{(i)}(\mathtt{Q}_3\ \mathtt{H}'.\mathtt{c}'\ \mathtt{Q}_1)\,\}\Big)$

3. *if* $\mathtt{Q}_1\ \mathtt{ev.n}\ \mathtt{Q}_2 \in \mathtt{Q}'\ \mathtt{A.f}\ \mathtt{Q}''$, *then*

$$\begin{array}{rl}\mathbb{R}_{\mathtt{Q}}{}^{(i+1)}(\mathtt{Q}_1\ \mathtt{ev.n}\ \mathtt{Q}_2) = & \mathbb{R}_{\mathtt{Q}}{}^{(i)}(\mathtt{Q}_1\ \mathtt{ev.n}\ \mathtt{Q}_2)\\ & \cup\Big(\bigcup_{\mathtt{Q}_3\ \mathtt{H}'.\mathtt{c}'\ \mathtt{Q}_1\in\mathtt{C}}\{\mathbb{A}\lhd\mathtt{Q}_1\ \mathtt{ev.n}\ \mathtt{Q}_2 \mid \mathbb{A}\in\mathbb{R}_{\mathtt{Q}}{}^{(i)}(\mathtt{Q}_3\ \mathtt{H}'.\mathtt{c}'\ \mathtt{Q}_1)\ \textit{and}\ \mathtt{Q}'\ \mathtt{A.f}\ \mathtt{Q}''\in\mathbb{A}\}\Big)\end{array}$$

By definition $\mathbb{R}_{\mathtt{Q}}{}^{(i)} \le \mathbb{R}_{\mathtt{Q}}{}^{(i+1)}$. *Since the set of possible functions from clauses to sets of abstract computations is a finite lattice, there exists k such that* $\mathbb{R}_{\mathtt{Q}}{}^{(k)} = \mathbb{R}_{\mathtt{Q}}{}^{(k+1)}$ *[11]. Let* $\mathbb{R}_{\mathtt{Q}}$ *be such fixpoint.*

The map $\mathbb{R}_{\mathtt{Q}}$ takes a clause $\mathtt{Q}'\ \mathtt{H.c}\ \mathtt{Q}''$ and returns the set of abstract computations starting at the initial state $\mathtt{Q}$ and ending at $\mathtt{Q}'\ \mathtt{H.c}\ \mathtt{Q}''$. Notice that every set $\mathbb{A}\in\mathbb{R}_{\mathtt{Q}}(\mathtt{Q}'\ \mathtt{H.c}\ \mathtt{Q}'')$ owns a *consistency property* (*cf.* the item 2 of Definition 2): if an event is in $\mathbb{A}$ then the function that contains it is also in $\mathbb{A}$.

The clauses that will be deemed "unreachable" (from $\mathtt{Q}$) are those such that $\mathbb{R}_{\mathtt{Q}}(\mathtt{Q}'\ \mathtt{H.c}\ \mathtt{Q}'') = \varnothing$. In particular, $\mathbb{R}_{\mathtt{Q}}$ allows us to discard

- those clauses such that there is no path in the *control flow graph* of $\mathtt{C}$ (the finite automaton whose states are the contract's states and transitions $\mathtt{Q}' \xrightarrow{\mathtt{H.c}} \mathtt{Q}''$ are the functions and events $\mathtt{Q}'\ \mathtt{H.c}\ \mathtt{Q}''$) from $\mathtt{Q}$ to them;
- and those events such that every path in the control flow graph from $\mathtt{Q}$ to them has no transition labelled with the function containing the event.

For example, in `Sample`:

$$\mathbb{R}_{\mathtt{Init}}(\mathtt{Init\ A.f\ Run}) = \{\,\mathtt{Init\ A.f\ Run}\,\} \qquad \mathbb{R}_{\mathtt{Init}}(\mathtt{Init\ B.g\ Go}) = \{\,\mathtt{Init\ B.g\ Go}\,\} \qquad \mathbb{R}_{\mathtt{Init}}(\mathtt{Go\ ev.4\ End}) = \varnothing$$

The reader is encouraged to verify that, in `SampleTime`,

$$\mathbb{R}_{\mathtt{Init}}(\mathtt{Comp\ ev.5\ End}) = \{\,\mathtt{Init\ A.f\ Cont\ ;\ Cont\ ev.4\ Run\ ;\ Run\ B.g\ Go\ ;\ Go\ ev.9\ Comp\ ;\ Comp\ ev.5\ End}\,\}$$

therefore $\mathbb{R}_{\mathtt{Init}}(\mathtt{Comp\ ev.5\ End}) \neq \varnothing$, which means that `Comp ev.5 End` is reachable according to $\mathbb{R}_{\mathtt{Init}}$.

Definition 3 *Let* $\mathtt{C}$ *be a* μStipula *contract with initial configuration* $\mathbb{C}, \mathbb{t}$. *The* underlying abstract computation *of* $\mathbb{C}, \mathbb{t} \xrightarrow{\mu_1} \cdots \xrightarrow{\mu_n} \mathbb{C}', \mathbb{t}'$ *is the sequence of clauses* $\mathtt{Q}_1\ \mathtt{H}_1.\mathtt{c}_1\ \mathtt{Q}'_1\ ;\ \cdots\ ;\ \mathtt{Q}_k\ \mathtt{H}_k.\mathtt{c}_k\ \mathtt{Q}'_k$ *such that*

1. *if* $\mathtt{Q}\ \mathtt{H.c}\ \mathtt{Q}'$ *is the underlying clause of* μ_i *then there is* j *with* $\mathtt{Q}\ \mathtt{H.c}\ \mathtt{Q}' = \mathtt{Q}_j\ \mathtt{H}_j.\mathtt{c}_j\ \mathtt{Q}'_j$;
2. *for every i, if* μ_i *is the first label whose underlying clause is* $\mathtt{Q}_j\ \mathtt{H}_j.\mathtt{c}_j\ \mathtt{Q}'_j$ *then* $\{\mathtt{Q}_1\ \mathtt{H}_1.\mathtt{c}_1\ \mathtt{Q}'_1, \cdots, \mathtt{Q}_{j_1}\ \mathtt{H}_{j-1}.\mathtt{c}_{j-1}\ \mathtt{Q}'_{j-1}\}$ *is the set of underlying clauses of* $\mu_1, \cdots, \mu_{i-1}$.

Theorem 1 (Correctness of $\mathbb{R}_{\mathtt{Q}}$) *Let* $\mathtt{C}$ *be a* μStipula *contract with initial state* $\mathtt{Q}$ *and initial configuration* $\mathbb{C}, \mathbb{t}$. *Let also*

$$\mathbb{C}, \mathbb{t} \longrightarrow^* \mathbb{C}', \mathbb{t}' \xrightarrow{\mu} \mathbb{C}'', \mathbb{t}' \tag{1}$$

and $\mathtt{Q}'\ \mathtt{H.c}\ \mathtt{Q}''$ *be the underlying clause of* μ. *Then there is* $\mathbb{A}\in\mathbb{R}_{\mathtt{Q}}(\mathtt{Q}'\ \mathtt{H.c}\ \mathtt{Q}'')$ *that is the underlying abstract computation of* (1). *Therefore* $\mathbb{R}_{\mathtt{Q}}(\mathtt{Q}'\ \mathtt{H.c}\ \mathtt{Q}'') \neq \varnothing$.

We use $\mathbb{R}_{\mathtt{Q}}$ to define the cyclic behaviours of contract functions. This predicate will be relevant in the following because we spot timed-out events when the corresponding functions are acyclic.

Definition 4 *Let* C *be a* μ*Stipula contract with initial state* Q*. Let*

$$\mathit{Cyclic}_{\mathtt{Q}}(\mathtt{Q}_1\ \mathtt{A.f}\ \mathtt{Q}_2) = \begin{cases} \mathit{true} & \text{if } \mathbb{A}\ \mathtt{;}\ \mathtt{Q}_1\ \mathtt{A.f}\ \mathtt{Q}_2\ \mathtt{;}\ \mathbb{A}' \in \mathbb{R}_{\mathtt{Q}}(\mathtt{Q}_1\ \mathtt{A.f}\ \mathtt{Q}_2) \text{ and } \mathtt{Q}_1'\ \mathtt{H.c}\ \mathtt{Q}_1'' \in \mathbb{A}\ \mathtt{;}\ \mathtt{Q}_1\ \mathtt{A.f}\ \mathtt{Q}_2 \\ & \text{and } \mathtt{Q}_2'\ \mathtt{H}'\mathtt{.c}'\ \mathtt{Q}_2'' \in \mathtt{Q}_1\ \mathtt{A.f}\ \mathtt{Q}_2\ \mathtt{;}\ \mathbb{A}' \text{ with } \mathtt{Q}_2'' = \mathtt{Q}_1' \\ \mathit{false} & \text{otherwise} \end{cases}$$

($\mathit{Cyclic}_{\mathtt{Q}}$ *is undefined on events.)*

The predicate $\mathit{Cyclic}_{\mathtt{Q}}$ uses the property that, in abstract computations of $\mathbb{R}_{\mathtt{Q}}(\mathtt{Q}_1\ \mathtt{A.f}\ \mathtt{Q}_2)$, clauses are added to the right of the sequences (if they are not already present). Therefore, if in the final states "on the right" of $\mathtt{Q}_1\ \mathtt{A.f}\ \mathtt{Q}_2$ (also including $\mathtt{Q}_2$) there is a state that occurs as initial in some clause "on the left" (also including $\mathtt{Q}_1$) then the function is deemed cyclic. The predicate has not been defined on events because they are cyclic provided the corresponding function is. An immediate consequence of Theorem 1 is the correctness of $\mathit{Cyclic}_{\mathtt{Q}}$.

Corollary 1 *Let* C *be a* μ*Stipula contract with initial state* Q *and initial configuration* $\mathbb{C}, \mathbb{t}$*. Let also*

$$\mathbb{C}, \mathbb{t} \longrightarrow^* \xrightarrow{\mu} \longrightarrow^* \xrightarrow{\mu'} \longrightarrow^* \mathbb{C}', \mathbb{t}'$$

such that the underlying clause of μ *and* μ' *is* $\mathtt{Q}_1\ \mathtt{A.f}\ \mathtt{Q}_2$*. Then* $\mathit{Cyclic}_{\mathtt{Q}}(\mathtt{Q}_1\ \mathtt{A.f}\ \mathtt{Q}_2) = \mathit{true}$*.*

The converse of Corollary 1 is false: since $\mathit{Cyclic}_{\mathtt{Q}}$ is over-approximating, it is possible that $\mathit{Cyclic}_{\mathtt{Q}}(\mathtt{Q}_1\ \mathtt{A.f}\ \mathtt{Q}_2) = \mathit{true}$ and there is no computation manifesting the corresponding cyclic behaviour (see Remark **??**).

4.1 Removing timed-out events

The map $\mathbb{R}_{\mathtt{Q}}$ does not parse time expressions; for this reason, in `SampleTime`, `Comp ev.5 End` is deemed reachable. The following refinement of $\mathbb{R}_{\mathtt{Q}}$ addresses time expressions and allows us to drop those events (and their continuations) whose time expressions are inconsistent with the contract behaviour.

Assessing consistency of time expressions at static time is complex because it is not possible to relay on the notion of computation (which are infinitely many) and on the absolute time $\mathbb{t}$ (which is a runtime concept). To overcome these issues, we use abstract computations as surrogates of computations and logical times as surrogates of absolute times. Logical times are terms that are defined by assigning a time variable to functions and a time variable plus a delta to events (the time variable is the logical time of the function containing the event, the delta is a natural number corresponding to the value of the time expression of the event). The time variable represents the (absolute) time when a function is invoked: it is a variable because its value is unknown statically. However, the critical point is the presence of cyclic behaviours (*cf.* the discussion about the contract `Ugly` in Section 2) because, in these cases, several instances of a function are required and it is not possible to associate a unique value to the corresponding time variable. Since abstract computations and logical times may lead to wrong conclusions about reachability in presence of cycles, we have decided to restrict the refinement of $\mathbb{R}_{\mathtt{Q}}$ to acyclic functions. We begin with the definition of logical time and with two auxiliary functions:

- the *logical time* is a term of the form $\zeta_1 + \cdots + \zeta_h + k$, where ζ_i are variables representing the times when functions are invoked and k is a constant in Nat;
- $\mathtt{EV_C}(\mathtt{Q}\ \mathtt{A.f}\ \mathtt{Q}') \stackrel{\text{def}}{=} \{\ \mathtt{Q}_1\ \mathtt{ev.n}\ \mathtt{Q}_2 \mid \mathtt{Q}_1\ \mathtt{ev.n}\ \mathtt{Q}_2 \in \mathtt{Q}\ \mathtt{A.f}\ \mathtt{Q}'\ \}$

- $\mathtt{TE}_{\mathtt{C}}(\cup_{i\in 1..m}\{\mathtt{Q}_i\ \mathtt{ev.n}_i\ \mathtt{Q}'_i\}) \stackrel{\text{def}}{=} \{\kappa_i \mid i\in 1..m \text{ and } \mathtt{now}+\kappa_i \gg_{\mathtt{n}_i} \mathtt{@Q}_i\{\,\} \Rightarrow \mathtt{@Q}'_i \in \mathtt{C}\}$.

Definition 5 *Let* C *be a* μ*Stipula contract and* ν *be an injctive mapping from functions* $\mathtt{Q\ A.f\ Q'} \in \mathtt{C}$ *to variables. Let*

- Θ_ν *be the function from abstract computations to logical times defined as follows:*

$$\Theta_\nu(\mathbb{A}) \stackrel{def}{=} \begin{cases} \Theta_\nu(\mathbb{A}') + \nu(\mathtt{Q\ A.f\ Q'}) & \text{if } \mathbb{A} = \mathbb{A}'\ ;\ \mathtt{Q\ A.f\ Q'} \\ \Theta_\nu(\mathbb{A}') + \nu(\mathtt{Q\ A.f\ Q'}) + \kappa & \text{if } \mathbb{A} = \mathbb{A}'\ ;\ \mathtt{Q\ A.f\ Q'}\ ;\ \mathbb{A}''\ ;\ \mathtt{Q}_1\ \mathtt{ev.n}\ \mathtt{Q}_2 \text{ and } \mathtt{Q}_1\ \mathtt{ev.n}\ \mathtt{Q}_2 \in \mathtt{Q\ A.f\ Q'} \\ & \text{and } \kappa = \max\left(\mathtt{TE}_{\mathtt{C}}(\mathbb{A}''\ ;\ \mathtt{Q}_1\ \mathtt{ev.n}\ \mathtt{Q}_2|_{\mathtt{EV_C(Q\ A.f\ Q')}})\right) \end{cases}$$

- $\mathbb{T}_\nu$, *called* time function*, be the following function from sets of abstract computations to sets of logical times:*

$$\mathbb{T}_\nu(\{\mathbb{A}_1,\cdots,\mathbb{A}_n\}) \stackrel{def}{=} \{\ \Theta_\nu(\mathbb{A}_1),\cdots,\Theta_\nu(\mathbb{A}_n)\ \}$$

For example, in `SampleTime` (we use $\mathbb{R}_{\mathtt{Init}}(\cdot)$ to specify sets of abstract computations; in this case, these sets are singletons):

$$\mathbb{T}_\nu(\mathbb{R}_{\mathtt{Init}}(\mathtt{Init\ A.f\ Cont})) = \{\zeta_{\mathtt{A.f}}\} \qquad \mathbb{T}_\nu(\mathbb{R}_{\mathtt{Init}}(\mathtt{Cont\ ev.4\ Run})) = \{\zeta_{\mathtt{A.f}}+1\}$$

$$\mathbb{T}_\nu(\mathbb{R}_{\mathtt{Init}}(\mathtt{Run\ B.g\ Go})) = \{\zeta_{\mathtt{A.f}}+\zeta_{\mathtt{B.g}}+1\} \qquad \mathbb{T}_\nu(\mathbb{R}_{\mathtt{Init}}(\mathtt{Go\ ev.9\ Comp})) = \{\zeta_{\mathtt{A.f}}+\zeta_{\mathtt{B.g}}+3\}$$

Definition 6 *Let* T *and* T' *be two sets of logical times. We say that* $T \le T'$ is solvable *if there are* $t \in T$, $t' \in T'$ *and a ground substitution* σ*, such that* $t\sigma \le t'\sigma$ *(*σ *maps variables to naturals). Otherwise we say that* $T \le T'$ is unsolvable.

For example $\{x+y+1\} \le \{x+2\}$ is solvable (replacing y with either 0 or 1), while $\{x+y+1\} \le \{x+y\}$ is not solvable. The next key lemma guarantees that the logical times of clauses are sound with respect to computations.

Everything is now in place for the definition of $\mathbb{R}^{+}_{\mathtt{Q}}$; the definition and its correctness conclude the section.

Definition 7 *Let* C *be a* μ*Stipula contract with initial state* Q *and let* ν *be an injctive mapping from functions* $\mathtt{Q\ A.f\ Q'} \in \mathtt{C}$ *to variables. The sequence of maps* $\mathbb{R}^{+(0)}_{\mathtt{Q}}$, $\mathbb{R}^{+(1)}_{\mathtt{Q}}, \cdots$ *is defined as follows:*

1. $\mathbb{R}^{+(0)}_{\mathtt{Q}}(\mathtt{Q}_1\ \mathtt{H.c}\ \mathtt{Q}_2) = \begin{cases} \{\ \mathtt{Q}_1\ \mathtt{H.c}\ \mathtt{Q}_2\ \} & \text{if } \mathtt{Q} = \mathtt{Q}_1 \text{ and } \mathtt{H.c} = \mathtt{A.f} \text{ and } \mathtt{Q\ A.f\ Q}_2 \in \mathtt{C} \\ \varnothing & \text{otherwise} \end{cases}$

2. $\mathbb{R}^{+(i+1)}_{\mathtt{Q}}(\mathtt{Q}_1\ \mathtt{A.f}\ \mathtt{Q}_2) = \quad \mathbb{R}_{\mathtt{Q}}{}^{(i)}(\mathtt{Q}_1\ \mathtt{A.f}\ \mathtt{Q}_2) \cup \left(\bigcup_{\mathtt{Q}_3\ \mathtt{H'.c'}\ \mathtt{Q}_1 \in \mathtt{C}} \{\ \mathbb{A} \lhd \mathtt{Q}_1\ \mathtt{A.f}\ \mathtt{Q}_2 \mid \mathbb{A} \in \mathbb{R}^{+(i)}_{\mathtt{Q}}(\mathtt{Q}_3\ \mathtt{H'.c'}\ \mathtt{Q}_1)\ \}\right)$

3. *if* $\mathtt{Q}_1\ \mathtt{ev.n}\ \mathtt{Q}_2 \in \mathtt{Q'\ A.f\ Q''}$ *and* $Cyclic_{\mathtt{C}}(\mathtt{Q'\ A.f\ Q''}) = true$*, then*

$$\mathbb{R}^{+(i+1)}_{\mathtt{Q}}(\mathtt{Q}_1\ \mathtt{ev.n}\ \mathtt{Q}_2) = \quad \mathbb{R}^{+(i)}_{\mathtt{Q}}(\mathtt{Q}_1\ \mathtt{ev.n}\ \mathtt{Q}_2) \cup \left(\bigcup_{\mathtt{Q}_3\ \mathtt{H'.c'}\ \mathtt{Q}_1 \in \mathtt{C}} \{\mathbb{A} \lhd \mathtt{Q'\ ev.n\ Q''} \mid \mathbb{A} \in \mathbb{R}^{+(i)}_{\mathtt{Q}}(\mathtt{Q}_3\ \mathtt{H'.c'}\ \mathtt{Q}_1) \text{ and } \mathtt{Q'\ A.f\ Q''} \in \mathbb{A}\}\right)$$

4. *if* $\mathtt{Q}_1\ \mathtt{ev.n}\ \mathtt{Q}_2 \in \mathtt{Q'\ A.f\ Q''}$ *and* $Cyclic_{\mathtt{C}}(\mathtt{Q'\ A.f\ Q''}) = false$*, then*

$$\begin{aligned} \mathbb{R}^{+(i+1)}_{\mathtt{Q}}(\mathtt{Q}_1\ \mathtt{ev.n}\ \mathtt{Q}_2) = \\ \quad & \mathbb{R}^{+(i)}_{\mathtt{Q}}(\mathtt{Q}_1\ \mathtt{ev.n}\ \mathtt{Q}_2) \\ \quad \cup \Big(\bigcup_{\mathtt{Q}_3\ \mathtt{H'.c'}\ \mathtt{Q}_1 \in \mathtt{C}} \{\mathbb{A} \lhd \mathtt{Q}_1\ \mathtt{ev.n}\ \mathtt{Q}_2 \mid\ & \mathbb{A} \in \mathbb{R}^{+(i)}_{\mathtt{Q}}(\mathtt{Q}_3\ \mathtt{H'.c'}\ \mathtt{Q}_1) \text{ and } \mathtt{Q'\ A.f\ Q''} \in \mathbb{A} \\ & \text{and } \mathbb{T}_\nu(\{\mathbb{A}\}) \le \mathbb{T}_\nu(\mathbb{R}^{+(i)}_{\mathtt{Q}}(\mathtt{Q'\ A.f\ Q''}) \lhd \mathtt{Q}_1\ \mathtt{ev.n}\ \mathtt{Q}_2) \text{ is solvable}\}\Big) \end{aligned}$$

By definition $\mathbb{R}^{+(i)}_{\mathtt{Q}} \subseteq \mathbb{R}^{+(i+1)}_{\mathtt{Q}}$. *Since the maps from clauses to sets of abstract computations of a contract* C *is a finite lattice, there exists* κ *such that* $\mathbb{R}^{+(\kappa)}_{\mathtt{Q}} = \mathbb{R}^{+(\kappa+1)}_{\mathtt{Q}}$ *[11]. Let* $\mathbb{R}^{+}_{\mathtt{Q}}$ *be such fixpoint.*

The definition of $\mathbb{R}^{+}_{\mathtt{Q}}$ is clearly a refinement of $\mathbb{R}_{\mathtt{Q}}$, adding more constraints in case of events $\mathtt{now} + \kappa \gg_{\mathtt{n}} @\mathtt{Q}_1\{S\} \Rrightarrow @\mathtt{Q}_2$ that do not belong to cyclic functions. In particular, in these cases, we must verify that there is an abstract computation $\mathbb{A}$ leading to $\mathtt{Q}_1$ ev.n $\mathtt{Q}_2$ (without the event) such that its logical time is smaller or equal to that of some abstract computations of $\mathtt{Q}'$ A.f $\mathtt{Q}''$ (the function containing the event) plus the value κ. In case the constraint is not solvable and the abstract computation is not added. For example, in SampleTime where every function is not cyclic, $\mathbb{R}^{+}_{\mathtt{Init}}(\mathtt{Comp}\ \mathtt{ev.5}\ \mathtt{End}) = \varnothing$ because

- $\mathbb{R}^{+}_{\mathtt{Init}}(\mathtt{Go}\ \mathtt{ev.9}\ \mathtt{Comp}) = \{\ \mathtt{Init\ A.f\ Cont},\ \mathtt{Cont\ ev.4\ Run},\ \mathtt{Run\ B.g\ Go},\ \mathtt{Go\ ev.9\ Comp}\ \}$;
- then $\mathbb{T}_v(\mathbb{R}^{+}_{\mathtt{Init}}(\mathtt{Go}\ \mathtt{ev.9}\ \mathtt{Comp})) = \zeta_{\mathtt{A.f}} + \zeta_{\mathtt{B.g}} + 3$ and $\mathbb{T}_v(\mathbb{R}^{+}_{\mathtt{Init}}(\mathtt{Init\ A.f\ Cont}) \triangleleft \mathtt{Comp\ ev.5\ End}) = \zeta_{\mathtt{A.f}} + 2$;
- and there is no ground substitution such that $\zeta_{\mathtt{A.f}} + \zeta_{\mathtt{B.g}} + 3 \leq \zeta_{\mathtt{A.f}} + 2$.

Theorem 2 (Correctness of $\mathbb{R}^{+}_{\mathtt{Q}}$) *Let* C *be a* μ*Stipula contract with initial state* Q *and initial configuration* $\mathbb{C}, \mathbb{t}$. *Let also*

$$\mathbb{C}, \mathbb{t} \longrightarrow^{*} \mathbb{C}', \mathbb{t}' \xrightarrow{\mu} \mathbb{C}'', \mathbb{t}' \tag{2}$$

and $\mathtt{Q}'$ H.c $\mathtt{Q}''$ *be the underlying clause of* μ. *Then there is* $\mathbb{A} \in \mathbb{R}^{+}_{\mathtt{Q}}(\mathtt{Q}'\ \mathtt{H.c}\ \mathtt{Q}'')$ *that is the underlying abstract computation of* (2). *Therefore* $\mathbb{R}^{+}_{\mathtt{Q}}(\mathtt{Q}'\ \mathtt{H.c}\ \mathtt{Q}'') \neq \varnothing$.

5 Extensions

The language μ*Stipula* of Section 3 is a micro subset of *Stipula* [9]. In the full language, contracts also define fields and assets, functions have arguments that may be assets (in the notation, the curved bracket group standard arguments, the square brackets group assets) and bodies may also contain statements that can update fields and move assets. While these features do not affect the analysis of reachability, there are others that have a relevant effect. In particular, *Stipula* contracts have the *agreement clause* that may initialize fields occurring in *(more expressive) time expressions*. The reachability analyzer also covers these features and, in this section, we discuss the upgrade of the technique of Section 4.

Agreements and time expressions with names. *Stipula* time expressions also contain names, *i.e.*

$$\mathtt{t} ::= \mathtt{now} \ \mid\ \mathtt{t} + \kappa \ \mid\ \mathtt{t} + \mathtt{x}$$

where these x are *fields* of the contract. The theory of reachability remains the same when these fields are initialized at the beginning (*cf.* the init clause) and their value never changes. However, *Stipula* contract may be drawn with values of fields left unspecified and the values of fields (as well as the actual instances of parties' names) are specified in the *agreement clause* that is performed when the contract is executed. This means that fields are undefined when the contract is drawn up (in particular those occurring in time expressions). For example, in the code

```
1 stipula Agree {
2    fields x,y
3    agreement(A,B)(x,y) { A,B : x,y } ⇛ @Init
4    @Init A: f()[] {
5       now + x ≫ @Run {} ⇛ @Comp
```

```
 6      } ⇒ @Cont
 7      @Cont B: g()[] {
 8         now + y ≫ @Comp {} ⇒ @End
 9      } ⇒ @Run
10  }
```

the agreement clause defines the parties (`A` and `B`) that are involved in the contract and the fields' values (`x` and `y`) they are required to agree upon (notice that the `init` clause is now part of the agreement). It is expected that, when the functions will be invoked, the fields `x` and `y` have been already initialized.

To cover these situations, which are very common in practice, the analyzer also deals with symbolic names and with constraints over them. This is not simple and we discuss our solution below.

Without loss of generality, we assume that no field occurring in time expressions has been initialized. We use an alternative form of the function Θ_ν in Section 4. The new function, noted $\Theta_\nu^{\mathtt{fd}}$ (the superscript `fd` stands for field names), returns *sets* T of terms $\zeta_{\mathtt{A}_1.\mathtt{f}_1}+\cdots+\zeta_{\mathtt{A}_m.\mathtt{f}_m}+\kappa_1\times\mathtt{x}_1+\cdots+\kappa_h\times\mathtt{x}_h+\kappa$, where $\zeta_{\mathtt{A}_1.\mathtt{f}_1}+\cdots+\zeta_{\mathtt{A}_m.\mathtt{f}_m}$ is a logical time, $\mathtt{x}_1,\cdots,\mathtt{x}_h$ are fields of the contract and $\kappa_1,\cdots,\kappa_h,\kappa\in\mathsf{Nat}$. We also redefine $\mathtt{TE}_\mathtt{C}$ and $\mathbb{T}_\nu$. Let `C` be a *Stipula* contract; then

- $\mathtt{TE}_\mathtt{C}^{\mathtt{fd}}(\bigcup_{i\in 1..m}(\{\mathtt{Q}_i\ \mathtt{ev.n}_i\ \mathtt{Q}_i'\})\stackrel{\text{def}}{=}\{t_i\mid i\in 1..m \text{ and } \mathtt{now}+t_i\gg_{\mathtt{n}_i}\mathtt{@Q}_i\{\,\}\Rightarrow\mathtt{@Q}_i'\in\mathtt{C}\}$;
- $\Theta_\nu^{\mathtt{fd}}$ be the following function from abstract computations to logical times:

$$\Theta_\nu^{\mathtt{fd}}(\mathbb{A})\stackrel{\text{def}}{=}\begin{cases}\Theta_\nu^{\mathtt{fd}}(\mathbb{A}')+\nu(\mathtt{Q\ A.f\ Q}') & \text{if } \mathbb{A}=\mathbb{A}'\ ;\ \mathtt{Q\ A.f\ Q}'\\ \bigcup_{t\in T}\Big(\Theta_\nu^{\mathtt{fd}}(\mathbb{A}')+\ \nu(\mathtt{Q\ A.f\ Q}')+t\Big) & \text{if } \mathbb{A}=\mathbb{A}'\ ;\ \mathtt{Q\ A.f\ Q}'\ ;\ \mathbb{A}''\ ;\ \mathtt{Q}_1\ \mathtt{ev.n\ Q}_2 \text{ and } \mathtt{Q}_1\ \mathtt{ev.n\ Q}_2\in\mathtt{Q\ A.f\ Q}'\\ & \text{and } T=\mathtt{TE}_\mathtt{C}^{\mathtt{fd}}(\mathbb{A}''\ ;\ \mathtt{Q}_1\ \mathtt{ev.n\ Q}_2|_{\mathtt{EV_C(Q\ A.f\ Q')}})\end{cases}$$

- $\mathbb{T}_\nu^{\mathtt{fd}}$ be the following function from sets of abstract computations to sets of logical times:

$$\mathbb{T}_\nu^{\mathtt{fd}}(\{\mathbb{A}_1,\cdots,\mathbb{A}_n\})\stackrel{\text{def}}{=}\bigcup_{i\in 1..n}\Theta_\nu^{\mathtt{fd}}(\mathbb{A}_i)$$

Using the foregoing functions, the analyzer computes $\mathbb{R}^+_\mathtt{Q}$, by gathering constraints between fields. For example, if $\mathbb{T}_\nu^{\mathtt{fd}}(\mathbb{R}^+_{\mathtt{Q}_0}(\mathtt{Q\ H.c\ Q}'))=\{t_1,t_2\}$ and $\mathbb{T}_\nu^{\mathtt{fd}}(\mathbb{R}^+_{\mathtt{Q}_0}(\mathtt{Q}'\ \mathtt{H}'.\mathtt{c}'\ \mathtt{Q}''))=\{t_1',t_2'\}$ then the analyzer highlights the four constraints $t_1\le t_1'$, $t_1\le t_2'$, $t_2\le t_1'$, $t_2\le t_2'$ in the returned message. For instance, in the contract `Agree`, $\mathbb{T}_\nu^{\mathtt{fd}}(\mathbb{R}^+_{\mathtt{Init}}(\mathtt{Run\ ev.5\ Comp}))=\{\zeta_{\mathtt{A.f}}+\zeta_{\mathtt{B}.g}+1\times\mathtt{x}+0\times\mathtt{y}+0\}$ and $\mathbb{T}_\nu^{\mathtt{fd}}(\mathbb{R}^+_{\mathtt{Init}}(\mathtt{Comp\ ev.8\ End}))=\{\zeta_{\mathtt{A.f}}+\zeta_{\mathtt{B.g}}+0\times\mathtt{x}+1\times\mathtt{y}+0\}$. Therefore in order to have a larger $\mathbb{R}^+_{\mathtt{Init}}$ we need that $\mathtt{x}\le\mathtt{y}$. In fact, the tool returns

```
"reachability constraint": [ x <= y ] .
```

Of course, the precision of the reachability analyzer may be augmented if it is *also* run *after* the agreement, once the fields' values are known (henceforth we are back to the basic theory). We are considering to extend the *Stipula* toolchain with this additional feature.

A full-fledged set of time expressions. *Stipula* time expressions are more generic than those in Section 3. In particular, the full syntax of `t` in *Stipula* is

$$\mathtt{t}::=\mathtt{now}\ \mid\ \mathit{date}\ \mid\ \mathtt{x}\ \mid\ \mathtt{t}+\mathtt{x}\ \mid\ \mathtt{t}+\kappa(\mathtt{'Y'}\mid\mathtt{'M'}\mid\mathtt{'D'}\mid\mathtt{'h'}\mid\mathtt{'m'})?$$

where *date* has the format "*year-month-day*". When `t` is `x`, it is intended that `x` stores a *date* and, for instance, $\mathtt{t}+5\mathtt{M}$ means "add 5 months to `t`" (`Y`, `D`, `h` and `m` stand for years, days, hours and minutes, respectively); when we write $\mathtt{t}+5$, as in the whole paper, it is intended $\mathtt{t}+5\mathtt{m}$.

The reachability analyzer covers these generic time-expressions by means of a preprocessor that transforms the above time expressions into those in the restricted syntax of Section 3. We discuss the case of an event $date \gg \texttt{@Q}\ \{S\} \Rightarrow \texttt{@Q}'$ in a function $\texttt{Q}_1\ \texttt{A.f}\ \texttt{Q}_2$, the other cases are similar. In this case, the preprocessor (*i*) replaces every *date* with an expression $\texttt{now} + \texttt{\_x}$, where $\texttt{\_x}$ is added to the fields, and (*ii*) returns a warning message if the computer clock is greater than *date*. Then the analyzer computes $\mathbb{T}_v^{\texttt{fd}}$ considering all these new fields $\texttt{\_x}$. The set of returned constraints are then extended with those of the form $t + \texttt{\_x} = date$, for every $t \in \mathbb{T}_v^{\texttt{fd}}(\mathbb{R}^{+}_{\texttt{Q}_0}(\texttt{Q}_1\ \texttt{A.f}\ \texttt{Q}_2))$. For instance, if the code

```
1  stipula OutofTime {
2      init Init
3      @Init A:f()[] {
4          "2024-01-01" ≫ @Cont { } ⇒ @End
5      } ⇒ @Cont
6  }
```

is executed today, the analyzer (actually the preprocessor) returns the warning `"expired code": [ Cont ev.5 End ]` . We notice that this warning is returned when the variables in time expressions are initialized either in the agreement or in the `field` clause. Otherwise, no warning or a "reachability constraint" message is returned.

6 Related works

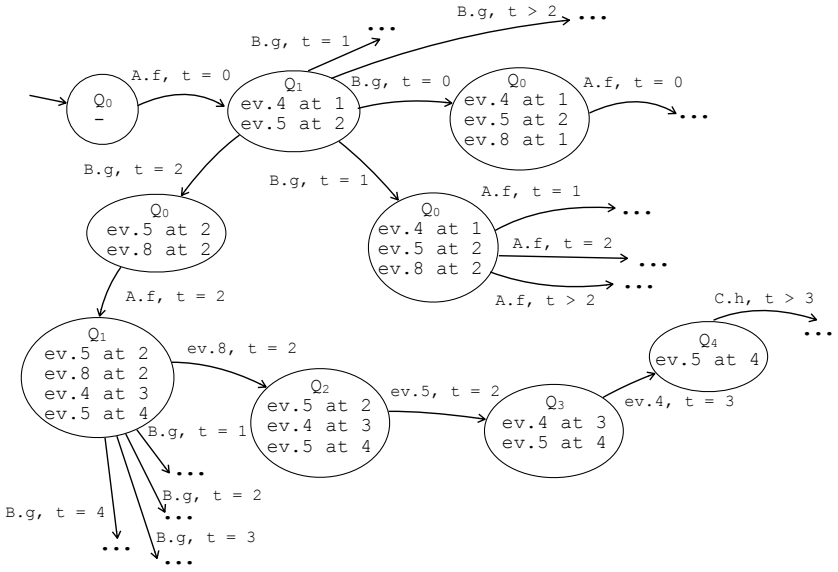
Reachability analysis is a very common optimization in compiler construction that uses data-flow analysis over the control flow graph of programs [1, 5]. In turn, this data-flow analysis employs fixpoint techniques to pinpoint parts of codes that cannot be reached. In our case, the control flow graph is the state-transition system of a *Stipula* contract and the fixpoint techniques (in the data-flow analysis) use dependencies event-function and perform the symbolic executions to derive logical times of clauses that might spot unreachable code units (functions or events). In particular, in our case, the symbolic executions return a set of constraints on (field) names that are evaluated by the analyzer and used to signal wrong instances of names. The technique we have followed is similar to the one described in [4]. Currently, our analyzer uses a very basic constraint solver that performs partial evaluations taking advantage of the additive format of time expressions in *Stipula*. Integrating the analyzer with an off-the-shelf constraint solver is in the to-do list of the prototype development.

Another technique that is used for reachability analysis is model checking. It undertakes a systematic exploration of the computations to verify a temporal logic formula expressing a reachability property ([15] and [7] are two well-known tools, the latter uses a symbolic technique to reduce the state explosion problem). While model checking is good at catching difficult corner cases, it is critical because of the state space, which may be infinite in *Stipula* (even if the control flow graph is finite). In particular, time expressions may generate very subtle reachability dependencies (*cf.* the `Ugly` contract) that require several instances of a function. At the time of writing, it is unclear whether there is a relationship between the number of instances of functions and the time expressions of a contract that might bound the reachability analysis.

The standard reference model of systems with clocks is *timed automata* [3]. As regards reachability analysis, one can think to define a compilation pattern from μ*Stipula* contracts to timed automata with one clock and without the reset operation. Henceforth, the reachability analysis of a μ*Stipula* is reduced to the reachability problem of this subclass of timed automata that has been proved to be *NLogSpace*-complete [18]. Actually we started our research by defining a compilation pattern in timed automata without obtaining valuable results. Let us discuss it briefly. The compilation gives automata whose

- states are contract states *plus* the events that have been produced and are live (timed-out events are garbage-collected);
- transitions are labelled with clauses and with a constraint on the time; we do not consider transitions to states that differ for the value of the clock.

For example, in the case of the contract `Ugly`, part of the corresponding timed automata is (we assume to start with the clock equal to 0)



As a pros, the above automata shows that $\mathtt{Q}_4$ `C.h` $\mathtt{Q}_5$ is reachable. The downside is that there is an explosion of states because of the value of the clock and the events that are *via via* collected (one might stick to timed automata with the reset operation but the situation does not change in a sensible way). Even more worse, we have no clear clue to stop the generation of states (the current one stops the generation when either every clause has been generated or path lengths reach a given upper bound). To this respect we observe that the above technique is not different from model checking the μ*Stipula* automata up-to a given length of computations. Finally, the compilation in timed automata do not seem to support the extension with variables discussed in Section 5 and the corresponding generation of constraints. Because of these remarks, we decided to develop the theory that has been presented in this paper.

Finally, reachability has also been demonstrated to be decidable for a large class of transition systems – the well-structured ones [13]. However, our attempts to prove that the μ*Stipula* transition system matches with the well-structured constraints have failed. It turns out that the expressive power of μ*Stipula* is an open problem: apparently, the combination of automata, time, and function invocations is a novel matter.

7 Conclusions

We have studied the reachability of clauses in legal contracts written in *Stipula*, a formal language with a precise syntax and semantics. This problem is knotty because clauses may be triggered by time constraints that must be partially evaluated. We have defined an algorithm, demonstrated its correctness and prototyped it.

The current prototype is a trade-off between coverage of cases and complexity of the theory. We have already observed that the precision of the analyzer is reduced in presence of cyclic functions. There

are also two other sources of imprecision that we have noticed. μ*Stipula* semantics gives precedence to events when, in a configuration, both functions can be invoked and events can be performed. This means that in the contract `UglyNow`

```
1  stipula UglyNow {
2     init Q0
3     @Q0 A: f()[] {
4        now ≫ @Q1 {} ⇒ @Q2
5     } ⇒ @Q1
6     @Q1 B: g()[] {
7     } ⇒ @Q3
8  }
```

```
1  stipula TwoEvents {
2     init Q0
3     @Q0 A: f()[] {
4        now + 1 ≫ @Q1 { } ⇒ @Q3
5        now + 2 ≫ @Q1 { } ⇒ @Q2
6     } ⇒ @Q1
7     @Q2 B: g()[] { } ⇒ @Q3
8  }
```

`B.g` is unreachable and the analyzer does not spot it. The imprecision of `UglyNow` may be easily recognized because it only appears in contracts that have a time expression `now` (the fix has not been implemented in order to avoid a *distinguo* that further entangle the theory). In another case, fixing the imprecision is more difficult. Consider the contract `TwoEvents`; there, `ev.4` and `ev.5` start in the same state at two different times. It turns out that `ev.5` starting at a later time (and its continuation `B.g`) will be unreachable and not detected by the analyzer. While this case is easy to spot, the general one where the competing events might belong to different functions or even to different instances of the same function is difficult. A more precise analysis should require to record the events that do not occur in the abstract computation and that compete with each other. The definition of $\mathbb{R}^{+}_{\mathtt{Q}}$ should also use this additional set to reduce the imprecision. All these precision issues are relevant research problems that are already in our agenda. However, it is worth to remark that the pattern on the left never showed up in the (more than 30 real) legal contracts we have encoded in *Stipula*, while we found only once the pattern on the right (the `Bakery` contract in the repository [12]).

As we said in Section 6, using a constraint solver to obtain more accurate outputs (such as messages about unreachability because constraints are unsolvable) is also in our to-do list. There is an additional reason for using constraint solvers, which is related to a last feature of *Stipula* that has not been covered in this paper. The complete format of a *Stipula* function is $\mathtt{@Q\ A{:}f}(\overline{\mathtt{y}})[\overline{\mathtt{k}}]\,(E)\,\{S\ W\} \Rightarrow \mathtt{@Q}'$, with the meaning that, when the function is invoked, the body can be executed *provided* the condition E holds. That is, if E is always false, the function is never executed and becomes unreachable. Since E may contain field names, asset names and formal parameters of the function, understanding whether *every* computation ending at `Q` has E unsolvable is pretty difficult.

We are aware of two techniques that might be considered. One is *dynamic symbolic execution* [2] that combines symbolic execution with the collection of constraints on paths and the analysis by means of a solver. here, the criticality is to gather all the possible computations; perhaps a saturation technique using some (approximation of the) fixpoint might be helpful. Another technique is to encode the operational semantics of *Stipula* contracts into the formal model of Abstract State Machines and then uses the corresponding tools for verifying reachability [6]. In this case, while the encoding in Abstract State Machines should be feasible, covering all the possible executions might require over-approximations that threaten the precision of the technique. The detailed study of this solution is under current investigation.

Acknowledgements. I thank Samuele Evangelisti and Alessandro Parenti for the long discussions we had about the reachability analysis. In particular Samuele, which developed the prototype [12] in his master thesis, has been very patient in rewriting the algorithm several times while I developed the theory and very accurate in spotting errors. Alessandro has triggered this research by underlying the relevance of finding errors in legal contracts when they are drawn.

References

[1] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools (2nd Edition)*. Addison Wesley, 2006.

[2] Eman Alatawi, Harald Søndergaard, and Tim Miller. Leveraging abstract interpretation for efficient dynamic symbolic execution. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*, ASE '17, pages 619–624. IEEE Press, 2017.

[3] Rajeev Alur and David L. Dill. A Theory of Timed Automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.

[4] Roberto Amadini, Graeme Gange, Peter Schachte, Harald Søndergaard, and Peter J. Stuckey. Abstract interpretation, symbolic execution and constraints. In *Recent Developments in the Design and Implementation of Programming Languages*, volume 86 of *OASIcs*, pages 7:1–7:19. Schloss Dagstuhl, 2020.

[5] Andrew W. Appel and Maia Ginsburg. *Modern Compiler Implementation in C*. Cambridge University Press, 1997.

[6] Paolo Arcaini, Andrea Bombarda, Silvia Bonfanti, Angelo Gargantini, Elvinia Riccobene, and Patrizia Scandurra. The ASMETA approach to safety assurance of software systems. In *Logic, Computation and Rigorous Methods*, volume 12750 of *Lecture Notes in Computer Science*, pages 215–238. Springer, 2021.

[7] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking. In *Proc. International Conference on Computer-Aided Verification (CAV 2002)*, volume 2404 of *LNCS*. Springer, 2002.

[8] 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. Springer, 2022.

[9] Silvia Crafa, Cosimo Laneve, Giovanni Sartor, and Adele Veschetti. Pacta sunt servanda: Legal contracts in *Stipula*. *Sci. Comput. Program.*, 225:102911, 2023.

[10] Silvia Crafa, Cosimo Laneve, and Adele Veschetti. The Stipula Project, July 2022. Available on github: `https://github.com/stipula-language`.

[11] Brian A. Davey and Hilary A. Priestley. *Introduction to lattices and order*. Cambridge University Press, 1990.

[12] Samuele Evangelisti. Stipula Reachability Analyzer, February 2024. Available on github: `https://github.com/stipula-language`.

[13] A. Finkel and Ph. Schnoebelen. Well-structured transition systems everywhere! *Theor. Comput. Sci.*, 256:63–92, 2001.

[14] Lexon Foundation. Lexon Home Page. `http://www.lexon.tech`, 2019.

[15] Gerard Holzmann. *Spin model checker, the: primer and reference manual*. Addison-Wesley Professional, 2003.

[16] Cosimo Laneve. Liquidity analysis in resource-aware programming. *J. Log. Algebraic Methods Program.*, 135:100889, 2023.

[17] Cosimo Laneve, Alessandro Parenti, and Giovanni Sartor. Legal contracts amending with Stipula. In *Proceedings of COORDINATION'23*, volume 13908 of *Lecture Notes in Computer Science*, pages 253–270. Springer, 2023.

[18] Franccois Laroussinie, Nicolas Markey, and Philippe Schnoebelen. Model checking timed automata with one or two clocks. In *Proceedings of CONCUR 2004, 15th International Conference in Concurrency Theory*, volume 3170 of *Lecture Notes in Computer Science*, pages 387–401. Springer, 2004.

[19] Open Source Contributors. The Accord Project. `https://accordproject.org`, 2018.

[20] Research Group on EC Private Law (Acquis Group) Study Group on a European Civil Code. *Principles, Definitions and Model Rules of European Private Law: Draft Common Frame of Reference (DCFR), Outline Edition*. Sellier, 2009.

[21] Aaron Wright, David Roon, and ConsenSys AG. OpenLaw Web Site. `https://www.openlaw.io`, 2019.

A Technical material

The appendix contains the technical material that has not been included in the paper for length constraints. To ease the reading, we report the proof with the corresponding statements.

We begin with the correctness of the map $\mathbb{R}_{\mathtt{Q}}$.

Theorem 1 (Correctness of $\mathbb{R}_{\mathtt{Q}}$) *Let* $\mathtt{C}$ *be a* μ*Stipula contract with initial state* $\mathtt{Q}$ *and initial configuration* $\mathtt{C}(\mathtt{Q},\_,\_),\mathbb{t}$. *Let also*

$$\mathbb{C},\mathbb{t} \longrightarrow^* \mathbb{C}',\mathbb{t}' \xrightarrow{\mu} \mathbb{C}'',\mathbb{t}' \tag{1}$$

and $\mathtt{Q}'\ \mathtt{H.c}\ \mathtt{Q}''$ *be the underlying clause of* μ. *Then there is* $\mathbb{A} \in \mathbb{R}_{\mathtt{Q}}(\mathtt{Q}'\ \mathtt{H.c}\ \mathtt{Q}'')$ *that is the underlying abstract computation of* (1). *Therefore* $\mathbb{R}_{\mathtt{Q}}(\mathtt{Q}'\ \mathtt{H.c}\ \mathtt{Q}'') \neq \varnothing$.

Proof: Let the computation (1) be $\mathtt{C}(\mathtt{Q},\_,\_),\mathbb{t} \longrightarrow^* \mathtt{C}(\mathtt{Q}',\_,\Psi),\mathbb{t}' \xrightarrow{\mu} \mathtt{C}(\mathtt{Q}',\Sigma,\Psi),\mathbb{t}'$. By induction on the length of $\mathtt{C}(\mathtt{Q},\_,\_),\mathbb{t} \longrightarrow^* \mathtt{C}(\mathtt{Q}',\_,\Psi),\mathbb{t}'$.

The basic case (the length is 0) is obvious. Assuming the theorem and the property hold for computations of length h, we demonstrate them for computations of length $h+1$. Consider the last transition of $\mathtt{C}(\mathtt{Q},\_,\_),\mathbb{t} \longrightarrow^* \mathtt{C}(\mathtt{Q}',\_,\Psi),\mathbb{t}'$ that is an instance either of [FUNCTION] or of [EVENT-MATCH]. Let $\mathtt{Q}'''\ \mathtt{H}'.\mathtt{c}'\ \mathtt{Q}'$ be its underlying clause. By inductive hypotheses, there is $\mathbb{A} \in \mathbb{R}_{\mathtt{Q}}(\mathtt{Q}'''\ \mathtt{H}'.\mathtt{c}'\ \mathtt{Q}')$ that is the underlying abstract computation of $\mathtt{C}(\mathtt{Q},\_,\_),\mathbb{t} \longrightarrow^* \mathtt{C}(\mathtt{Q}',\_,\Psi),\mathbb{t}'$.

When $\mu = \mathtt{A.f}$, by definition of $\mathbb{R}_{\mathtt{Q}}$, $\mathbb{A} \triangleleft \mathtt{Q}'\ \mathtt{A.f}\ \mathtt{Q}'' \in \mathbb{R}_{\mathtt{Q}}(\mathtt{Q}'\ \mathtt{A.f}\ \mathtt{Q}'')$.

When $\mu = \mathtt{ev.n}$, let $\mathtt{Q}'\ \mathtt{ev.n}\ \mathtt{Q}'' \in \mathtt{Q}_1'\ \mathtt{A.f}\ \mathtt{Q}_2'$. By the operational semantics, the computation $\mathtt{C}(\mathtt{Q},\_,\_),\mathbb{t} \longrightarrow^* \mathtt{C}(\mathtt{Q}',\_,\Psi),\mathbb{t}'$ must contain a transition $\mathtt{C}(\mathtt{Q}_1',\_,\Psi_1),\mathbb{t}_1 \xrightarrow{\mathtt{A.f}} \mathtt{C}(\mathtt{Q}_1',\ W \Rightarrow \mathtt{Q}_2',\Psi_1),\mathbb{t}_1$ and $W = \mathbb{t}' \gg_{\mathtt{n}} \mathtt{Q}'\{\,\} \Rightarrow \mathtt{Q}'' \,|\, W'$. Therefore, by the inductive hypotheses, $\mathtt{Q}_1'\ \mathtt{A.f}\ \mathtt{Q}_2' \in \mathbb{A}$. Hence, by definition of $\mathbb{R}_{\mathtt{Q}}$, $\mathbb{A} \triangleleft \mathtt{Q}'\ \mathtt{ev.n}\ \mathtt{Q}'' \in \mathbb{R}_{\mathtt{Q}}(\mathtt{Q}'\ \mathtt{ev.n}\ \mathtt{Q}'')$. □

The proof of Corollary 1is omitted because it is similar to the one of the following Lemma 1.

The correctness of $\mathbb{R}^+_{\mathtt{Q}}$ requires two preliminary statements about acyclic functions. The first one, Lemma 1, is a property about the format of underlying abstract computations containing acyclic functions. The second one, Lemma 2, is about time expressions of abstract computations containing events of acyclic functions.

Lemma 1 *Let* $\mathtt{C}$ *be a* μ*Stipula contract with initial state* $\mathtt{Q}$ *and initial configuration* $\mathbb{C},\mathbb{t}$. *Let also*

$$\mathbb{C},\mathbb{t} \longrightarrow^* \mathbb{C}_1,\mathbb{t}_1 \xrightarrow{\mu} \mathbb{C}_2,\mathbb{t}_1 \longrightarrow^* \mathbb{C}_2',\mathbb{t}_2 \tag{3}$$

with $\mathtt{Q}_1\ \mathtt{A.f}\ \mathtt{Q}_2$ *being the underlying clause of* μ *and* $\mathit{Cyclic}_{\mathtt{Q}}(\mathtt{Q}_1\ \mathtt{A.f}\ \mathtt{Q}_2) = \mathit{false}$. *Then the underlying abstract computation of* (3) *is* $\mathbb{A}\ ;\ \mathtt{Q}_1\ \mathtt{A.f}\ \mathtt{Q}_2\ ;\ \mathbb{A}'$ *where* $\mathbb{A}$ *and* $\mathbb{A}'$ *contain the underlying clauses in* $\mathbb{C},\mathbb{t} \longrightarrow^* \mathbb{C}_1,\mathbb{t}_1$ *and in* $\mathbb{C}_1',\mathbb{t}_1 \longrightarrow^* \mathbb{C}_2,\mathbb{t}_2$, *respectively;*

As a consequence, no clause in $\mathbb{A}$ *occurs in* $\mathbb{A}'$ *and, conversely, no clause in* $\mathbb{A}'$ *occurs in* $\mathbb{A}$.

Proof: Since $\mathit{Cyclic}_{\mathtt{Q}}(\mathtt{Q}_1\ \mathtt{A.f}\ \mathtt{Q}_2) = \mathit{false}$ then, by Corollary 1, (3) has a unique transition whose underlying clause is $\mathtt{Q}_1\ \mathtt{A.f}\ \mathtt{Q}_2$. Therefore, the underlying abstract computation of (3), which exists by Theorem 1, may be written $\mathbb{A}\ ;\ \mathtt{Q}_1\ \mathtt{A.f}\ \mathtt{Q}_2\ ;\ \mathbb{A}'$. By definition of underlying abstract computation, $\mathbb{A}$ contains the underlying clauses in $\mathbb{C},\mathbb{t} \longrightarrow^* \mathbb{C}_1,\mathbb{t}_1$.

We demonstrate that every underlying clause $\mathtt{Q}_1'\ \mathtt{H.c}\ \mathtt{Q}_2'$ of transitions in $\mathbb{C}_2,\mathbb{t}_1 \longrightarrow^* \mathbb{C}_2',\mathbb{t}_2$ does not occur in $\mathbb{A}$. By contradiction, assume that $\mathtt{Q}_1'\ \mathtt{H.c}\ \mathtt{Q}_2' \in \mathbb{A}$. Therefore $\mathbb{A} = \mathbb{A}_1\ ;\ \mathtt{Q}_1'\ \mathtt{H.c}\ \mathtt{Q}_2'\ ;\ \mathbb{A}_2$ and the computation $\mathbb{C},\mathbb{t} \longrightarrow^* \mathbb{C}_1,\mathbb{t}_1$ may be decomposed as follows:

$$\mathbb{C},\mathbb{t} \longrightarrow^* \mathbb{C}_1',\mathbb{t}_1' \xrightarrow{\mu'} \mathbb{C}_1'',\mathbb{t}_1' \longrightarrow^* \mathbb{C}_1,\mathbb{t}_1$$

where $\mathbb{C}_1', \mathbb{t}_1' \xrightarrow{\mu'} \mathbb{C}_1'', \mathbb{t}_1'$ is the leftmost transition whose underlying clause is $\mathtt{Q}_1'\ \mathtt{H.c}\ \mathtt{Q}_2'$. According to the operational semantics, there are two subcases: either (*a*) there is a transition in $\mathbb{C}_1'', \mathbb{t}_1' \longrightarrow^* \mathbb{C}_1, \mathbb{t}_1$ whose underlying clause is $\mathtt{Q}_2'\ \mathtt{H}'\mathtt{.c}'\ \mathtt{Q}_2''$ or (*b*) $\mathtt{Q}_2' = \mathtt{Q}_1$ ($\mathtt{Q}_1$ the initial state of $\mathtt{Q}_1\ \mathtt{A.f}\ \mathtt{Q}_2$). Notice that, in case (*a*), $\mathtt{Q}_2'\ \mathtt{H}'\mathtt{.c}'\ \mathtt{Q}_2'' \in \mathbb{A}$. We conclude by observing that, by Definition 4, both in case (*a*) and in case (*b*), $Cyclic_{\mathtt{Q}}(\mathtt{Q}_1\ \mathtt{A.f}\ \mathtt{Q}_2) = true$, which is absurd. □

Lemma 2 *Let* C *be a Stipula contract with initial configuration* $\mathtt{C}(\mathtt{Q}, \_, \_), \mathbb{t}$. *Let also*

$$\begin{aligned} \mathtt{C}(\mathtt{Q}, \_, \_), \mathbb{t} \quad &\longrightarrow^* \mathtt{C}(\mathtt{Q}_1, \_, \Psi), \mathbb{t}_1 \xrightarrow{\mu} \mathtt{C}(\mathtt{Q}_1, \Sigma, \Psi), \mathbb{t}_1 \\ &\longrightarrow^* \mathtt{C}(\mathtt{Q}_1', \_, \Psi), \mathbb{t}_1' \xrightarrow{\mu'} \mathtt{C}(\mathtt{Q}_1', \Sigma, \Psi'), \mathbb{t}_1' \end{aligned} \tag{4}$$

where $\mathtt{Q}_1\ \mathtt{A.f}\ \mathtt{Q}_2$ *is the underlying clause of* μ, $\mathtt{Q}_1'\ \mathtt{ev.n}\ \mathtt{Q}_2'$ *is the underlying clause of* μ', *with* $\mathtt{Q}_1'\ \mathtt{ev.n}\ \mathtt{Q}_2' \in \mathtt{Q}_1\ \mathtt{A.f}\ \mathtt{Q}_2$ *and* $Cyclic_{\mathtt{C}}(\mathtt{Q}_1\ \mathtt{A.f}\ \mathtt{Q}_2) = false$. *Then let* $\mathbb{A}$ *be the underlying abstract computation of* (4)*:*

(*a*) $\mathbb{A} = \mathbb{A}'\ ;\ \mathtt{Q}_1'\ \mathtt{ev.n}\ \mathtt{Q}_2'$;

(*b*) $\mathbb{T}_\nu(\mathbb{A}') \leq \mathbb{T}_\nu(\mathbb{A})$ *is solvable.*

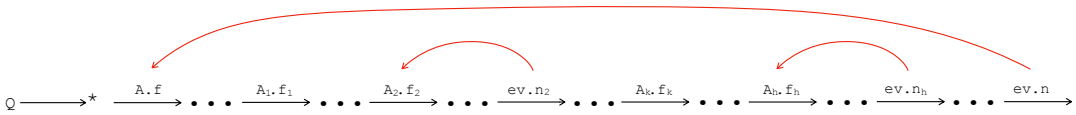
Proof: As regards (*a*), since $Cyclic_{\mathtt{C}}(\mathtt{Q}_1\ \mathtt{A.f}\ \mathtt{Q}_2) = false$, the computation (4) has exactly one transition that is an instance of $\mathtt{Q}_1\ \mathtt{A.f}\ \mathtt{Q}_2$ and, consequently, one transition that is an instance of $\mathtt{Q}_1'\ \mathtt{ev.n}\ \mathtt{Q}_2'$. By Definition 3, $\mathtt{Q}_1'\ \mathtt{ev.n}\ \mathtt{Q}_2'$ must be the last clause of $\mathbb{A}$ hence the thesis.

As regards (*b*), we may decompose $\mathbb{A}$ as follows

$$\mathbb{A} = \mathbb{A}'\ ;\ \mathtt{Q}_1\ \mathtt{A.f}\ \mathtt{Q}_2\ ;\ \mathbb{A}''\ ;\ \mathtt{Q}_1'\ \mathtt{ev.n}\ \mathtt{Q}_2'\ .$$

By Lemma 1, the underlying clauses of $\mathtt{C}(\mathtt{Q}, \_, \_), \mathbb{t} \longrightarrow^* \mathtt{C}(\mathtt{Q}_1, \_, \Psi), \mathbb{t}_1$ occur in $\mathbb{A}'$ and those of $\mathtt{C}(\mathtt{Q}_1, \Sigma, \Psi), \mathbb{t}_1 \longrightarrow^* \mathtt{C}(\mathtt{Q}_1', \_, \Psi), \mathbb{t}_1'$ occur in $\mathbb{A}''$. There are two subcases: (*b1*) the events in $\mathbb{A}''$ refer to functions in $\mathbb{A}''$ and (*b2*) there is an event in $\mathbb{A}''$ that refers to a function in $\mathbb{A}'$. We discuss them separately.

Subcase (*b1*) may be displayed by the drawing

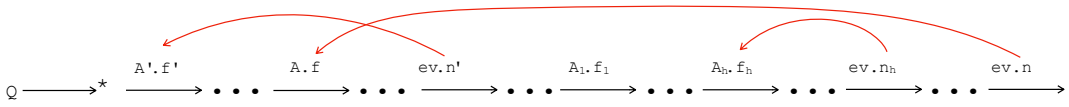


where the labels indicate the functions (we omit states) and events that are used in the computation of Θ_ν and the backwards red arrow indicate the connection of an event with its own function. In this case $\Theta_\nu(\mathbb{A}) = \Theta_\nu(\mathbb{A}') + \nu(\mathtt{Q}_1\ \mathtt{A.f}\ \mathtt{Q}_2) + k$, where $\mathtt{Q}_1'\ \mathtt{ev.n}\ \mathtt{Q}_2' = \mathtt{now} + k \gg_{\mathtt{n}} @\mathtt{Q}_1'\{S\} \Rightarrow @\mathtt{Q}_2'$. (Since the function is acyclic and because of the operational semantics, the other events of $\mathtt{A.f}$ in $\mathbb{A}''$, if any, have smaller time expressions.) Next, let us compute $\Theta_\nu(\mathbb{A}'\ ;\ \mathtt{Q}_1\ \mathtt{A.f}\ \mathtt{Q}_2\ ;\ \mathbb{A}'')$; we obtain

$$\Theta_\nu(\mathbb{A}'\ ;\ \mathtt{Q}_1\ \mathtt{A.f}\ \mathtt{Q}_2\ ;\ \mathbb{A}'') = \Theta_\nu(\mathbb{A}') + \nu(\mathtt{Q}_1\ \mathtt{A.f}\ \mathtt{Q}_2) + \Big(\sum_{i \in 1..m} \nu(\mathtt{Q}_i\ \mathtt{A}_i\mathtt{.f}_i\ \mathtt{Q}_i')\Big) + \Big(\sum_{j \in 1..m'} k_j\Big)\ .$$

where $\mathtt{Q}_i\ \mathtt{A}_i\mathtt{.f}_i\ \mathtt{Q}_i'$, $i \in 1..m$, are the functions of $\mathbb{A}''$ used by Θ_ν and k_j, $j \in 1..m'$ are the events corresponding to a subset of that functions. By the operational semantics $\sum_{j \in 1..m'} k_j \leq k$; therefore $\Theta_\nu(\mathbb{A}) \leq \Theta_\nu(\mathbb{A}'\ ;\ \mathtt{Q}_1\ \mathtt{A.f}\ \mathtt{Q}_2\ ;\ \mathbb{A}'')$ by instantiating every variable in $\sum_{i \in 1..m} \nu(\mathtt{Q}_i\ \mathtt{A}_i\mathtt{.f}_i\ \mathtt{Q}_i)$ with 0. Hence $\mathbb{T}_\nu(\mathbb{A}'\ ;\ \mathtt{Q}_1\ \mathtt{A.f}\ \mathtt{Q}_2\ ;\ \mathbb{A}'') \leq \mathbb{T}_\nu(\mathbb{A})$ is solvable.

Subcase (*b2*) may be displayed by the drawing



where, again, the labels indicate the functions (we omit states) and events that are used in the computation of Θ_ν and the backwards red arrow indicate the connection of an event with its own function. Let $\mathtt{Q}'_3\ \mathtt{ev.n}'\ \mathtt{Q}'_4$ be the *rightmost* event in $\mathbb{A}''$ that belongs to a function $\mathtt{Q}_3\ \mathtt{A}'.\mathtt{f}'\ \mathtt{Q}_4$ in $\mathbb{A}'$. In this case we may further decompose $\mathbb{A}'$ into $\mathbb{A}'_1\ ;\ \mathtt{Q}_3\ \mathtt{A}'.\mathtt{f}'\ \mathtt{Q}_4\ ;\ \mathbb{A}'_2$ and $\mathbb{A}''$ into $\mathbb{A}''_1\ ;\ \mathtt{Q}'_3\ \mathtt{ev.n}'\ \mathtt{Q}'_4\ ;\ \mathbb{A}''_2$. Then, by definition of Θ_ν, we have

$$\Theta_\nu(\mathbb{A}'\ ;\ \mathtt{Q}_1\ \mathtt{A.f}\ \mathtt{Q}_2\ ;\ \mathbb{A}'') = \Theta_\nu(\mathbb{A}'_1\ ;\ \mathtt{Q}_3\ \mathtt{A}'.\mathtt{f}'\ \mathtt{Q}_4) + k' + \Big(\sum_{i\in 1..m} \nu(\mathtt{Q}_i\ \mathtt{A}_i.\mathtt{f}_i\ \mathtt{Q}'_i)\Big) + \Big(\sum_{j\in 1..m'} k_j\Big)\ .$$

where k' is the maximum time expression of events of $\mathtt{Q}_3\ \mathtt{A}'.\mathtt{f}'\ \mathtt{Q}_4$ in $\mathbb{A}'_2\ ;\ \mathtt{Q}_1\ \mathtt{A.f}\ \mathtt{Q}_2\ ;\ \mathbb{A}''_1$ and the two summands are as in the subcase (*b1*). Notice that the variable $\nu(\mathtt{Q}_1\ \mathtt{A.f}\ \mathtt{Q}_2)$ does not occur in the above logical time, while it occurs in $\Theta_\nu(\mathbb{A})$. Therefore there exists a ground substitution of $\nu(\mathtt{Q}_1\ \mathtt{A.f}\ \mathtt{Q}_2)$ such that $\Theta_\nu(\mathbb{A}'\ ;\ \mathtt{Q}_1\ \mathtt{A.f}\ \mathtt{Q}_2\ ;\ \mathbb{A}'') \le \Theta_\nu(\mathbb{A})$ is solvable. Hence $\mathbb{T}_\nu(\mathbb{A}'\ ;\ \mathtt{Q}_1\ \mathtt{A.f}\ \mathtt{Q}_2\ ;\ \mathbb{A}'') \le \mathbb{T}_\nu(\mathbb{A})$ is solvable, as well. □

Every property that is necessary for the proof of the correctness of $\mathbb{R}^+_{\mathtt{Q}}$ has been proved. Hence the theorem.

Theorem 2 (Correctness of $\mathbb{R}^+_{\mathtt{Q}}$). *Let* $\mathtt{C}$ *be a Stipula contract with initial state* $\mathtt{Q}$ *and initial configuration* $\mathbb{C}, \mathbb{t}$. *Let also*

$$\mathbb{C}, \mathbb{t} \longrightarrow^* \mathbb{C}', \mathbb{t}' \xrightarrow{\mu} \mathbb{C}'', \mathbb{t}' \tag{2}$$

and $\mathtt{Q}'\ \mathtt{H.c}\ \mathtt{Q}''$ *be the underlying clause of* μ. *Then there is* $\mathbb{A} \in \mathbb{R}^+_{\mathtt{Q}}(\mathtt{Q}'\ \mathtt{H.c}\ \mathtt{Q}'')$ *that is the underlying abstract computation of* (2). *Therefore* $\mathbb{R}^+_{\mathtt{Q}}(\mathtt{Q}'\ \mathtt{H.c}\ \mathtt{Q}'') \neq \varnothing$.

Proof: Let the computation (2) be $\mathtt{C}(\mathtt{Q}, \_, \_), \mathbb{t} \longrightarrow^* \mathtt{C}(\mathtt{Q}', \_, \Psi), \mathbb{t}' \xrightarrow{\mu} \mathtt{C}(\mathtt{Q}', \Sigma, \Psi), \mathbb{t}'$. By induction on the length of $\mathtt{C}(\mathtt{Q}, \_, \_), \mathbb{t} \longrightarrow^* \mathtt{C}(\mathtt{Q}', \_, \Psi), \mathbb{t}' \xrightarrow{\mu} \mathtt{C}(\mathtt{Q}', \Sigma, \Psi'), \mathbb{t}'$. The proof is similar to that of Theorem 1 except for the case of $\mu = \mathtt{ev.n}$ and $\mathtt{Q}'_1\ \mathtt{ev.n}\ \mathtt{Q}'_2 \in \mathtt{Q}_1\ \mathtt{A.f}\ \mathtt{Q}_2$ and $\mathit{Cyclic}_{\mathtt{Q}}(\mathtt{Q}_1\ \mathtt{A.f}\ \mathtt{Q}_2) = \mathit{false}$.

In this case, the underlying abstract computation of (2) is $\mathbb{A} = \mathbb{A}'\ ;\ \mathtt{Q}_1\ \mathtt{A.f}\ \mathtt{Q}_2\ ;\ \mathbb{A}''\ ;\ \mathtt{Q}'_1\ \mathtt{ev.n}\ \mathtt{Q}'_2$ and, by Lemma 2, $\mathbb{T}_\nu(\mathbb{A}'\ ;\ \mathtt{Q}_1\ \mathtt{A.f}\ \mathtt{Q}_2\ ;\ \mathbb{A}'') \le \mathbb{T}_\nu(\mathbb{A})$ is solvable. Hence, by definition of $\mathbb{R}^+_{\mathtt{Q}}$, $\mathbb{A} \in \mathbb{R}^+_{\mathtt{Q}}(\mathtt{Q}'_1\ \mathtt{ev.n}\ \mathtt{Q}'_2)$. □