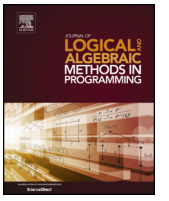
Journal of Logical and Algebraic Methods in Programming 135 (2023) 100889

Contents lists available at ScienceDirect

Journal of Logical and Algebraic Methods in Programming

journal homepage: www.elsevier.com/locate/jlamp



Liquidity analysis in resource-aware programming

Cosimo Laneve

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

ARTICLE INFO

Article history:
Received 15 November 2022
Received in revised form 4 May 2023
Accepted 12 June 2023
Available online 16 June 2023

Keywords:
Resource-aware programming
Assets
Liquidity
Type system
Symbolic analysis

ABSTRACT

Liquidity is a liveness property of programs managing resources that pinpoints those programs not freezing any resource forever. We consider a simple stateful language whose resources are assets (digital currencies, non fungible tokens, etc.). Then we define a type system that tracks in a symbolic way the input-output behavior of functions with respect to assets. These types and their composition, which define types of computations, allow us to design two algorithms for liquidity that have different precisions and costs. We also demonstrate the correctness of the algorithms.

1. Introduction

The proliferation of programming languages that explicitly feature resources has become more and more significant in the last decades. Cloud computing, with the need of providing an elastic amount of resources, such as memories, processors, bandwidth and applications, has pushed the definition of a number of formal languages with explicit primitives for acquiring and releasing them (see [1] and the references therein). More recently, a number of smart contracts languages have been proposed for managing and transferring resources that are assets (usually, in the form of digital currencies, like Bitcoin), such as the Bitcoin Scripting [5], Solidity [10], Vyper [12] and Scilla [17]. Even new programming languages are defined with (linear) types for resources, such as Rust [13].

In all these contexts, the efficient analysis of properties about the usage of resources is central to avoid flaws and bugs of programs that may also have relevant costs at runtime. In this paper, we focus on the *liquidity property*: a program is liquid when no resource remains frozen forever inside it, *i.e.* it is not redeemable by any party interacting with the program. For example, a program is not liquid if the body of a function does not use the resources transferred during the invocation by the caller. A program is also not liquid if, when it terminates, there is a resource that has not been emptied (*i.e.* its value is not 0). Liquidity has been studied in the past and different notions exist in the literature. The notion of liquidity that we study is the so-called *multiparty strategyless liquidity* in the taxonomy of [3,2] where it is assumed that all the contract's parties cooperate by actually calling the functions provided by the contract. Section 6 contains a detailed discussion.

We analyze liquidity for a simple programming language, a lightweight version of *Stipula*, which is a domain-specific language that has been designed for programming *legal contracts* [9,8,14]. In *Stipula*, programs are *contracts* that transit from state to state and a control logic specifies what functionality can be invoked by which caller; the set of callers is defined when the contract is instantiated. Resources are assets (digital currencies, smart keys, non-fungible tokens, etc.) that may be moved with ad-hoc operators from one to another.

E-mail address: cosimo.laneve@unibo.it.

https://doi.org/10.1016/j.jlamp.2023.100889

Table 1
Syntax of *Stipula* (X are assets, fields and parameters names).

Functions	F ::=	$@\mathtt{Q}\,\mathtt{A}:\mathtt{f}(\overline{\mathtt{y}})[\overline{\mathtt{k}}]\{S\}\Rightarrow @\mathtt{Q}'$
Prefixes	P ::=	$E\to\mathtt{x} \mid E\to\mathtt{A} \mid c\times\mathtt{h}\multimap\mathtt{h}' \mid c\times\mathtt{h}\multimap\mathtt{A} \quad // 0<c\leq 1$
Statements	S ::=	$\_ \mid P\,S \mid \mathtt{if}(E)\{S\}\mathtt{else}\{S\}\,S$
Expressions	E ::=	$v \mid X \mid E\,\mathtt{op}\,E \mid \mathtt{uop}\,E$
Values	v ::=	$n \mid \mathtt{false} \mid \mathtt{true}$

Our analyzer is built upon a type system that records the effects of functions on assets by using symbolic names. Then a *correctness* property, whereby the (liquidity) type of the final state of a computation is always an over-approximation of the actual state, allows us to safely reduce our analysis arguments to verifying if liquidity types of computations have assets that are empty.

We identify two liquidity properties. The first one is k-*separate liquidity*: *if an asset* k *becomes not-empty in a state then there is a continuation where* k *is empty in its final state*. While k-separate liquidity is satisfactory in contracts where assets are separated (no asset field is moved to another asset field, for instance when pairwise different assets have different categories, *e.g.* euros, cars, houses), it is inadequate in unrestricted contracts that move assets between asset fields. In these cases, the following stronger property is more reasonable: *if an asset becomes not-empty in a state then there is a continuation where all the assets are empty in its final state*.

Finally, we design two analysis algorithms with different computational costs and precision accuracies: one for k-separate liquidity and the other for liquidity. It turns out that the computational cost of such algorithm is quadratic with respect to the number of functions. The second algorithm is more precise because, for instance, it accepts contracts that empty assets by means of several function invocations. However, more precision requires more complexity because one has to analyze *computations*. The crucial issue of the analysis is therefore designing a terminating algorithm given that computations may be *infinitely many* because contracts may have cycles. For this reason we restrict to computations whose length is bound by a value (actually we found more reasonable computations where every function can be invoked a bounded number of times). The computational cost of k-separate liquidity and liquidity algorithms is higher than the previous case: it is exponential with respect to the number of functions.

The structure of the paper is as follows. The lightweight *Stipula* language is introduced in Section 2 and the semantics is defined in Section 3. Section 4 reports the theory underlying our liquidity analyzer and Section 5 illustrates the algorithms for verifying k-separate liquidity and liquidity. We end our contribution by discussing the related work in Section 6 and delivering our final remarks in Section 7. To ease the reading, the technical material has been reported in the Appendix.

This paper is an extended and revised version of [7] that also includes the analysis of k-separate liquidity and the design of the algorithms for k-separate liquidity and liquidity. The paper also contains the proofs of the statements.

2. The *Stipula* language

We use disjoint sets of names: *contract names*, ranged over by C, C′, ⋯; names referring to digital identities, called *parties*, ranged over by A, A′, ⋯; *function names* ranged over by f, g, ⋯; *asset names*, ranged over by h, k, ⋯, to be used both as contract's assets and function's asset parameters; *non asset names*, ranged over by x, y, ⋯, to be used both as contract's fields and function's non asset parameters. Assets and generic contract's fields are syntactically set apart since they have different semantics; similarly for functions' parameters. Names of assets, fields and parameters are generically ranged over by X. Names Q, Q′, ⋯ will range over contract states. To simplify the syntax, we often use the vector notation $\overline{x}$ to denote possibly empty *sequences* of elements. With an abuse of notation, in the following sections, $\overline{x}$ will also represent the *set* containing the elements in the sequence.

The code of a *Stipula* contract is[1]

$$\mathtt{stipula\ C\ \{\ parties}\ \overline{\mathtt{A}}\ \ \mathtt{fields}\ \overline{\mathtt{x}}\ \ \mathtt{assets}\ \overline{\mathtt{h}}\ \ \mathtt{init\ Q}\ \ \overline{F}\ \}$$

where C identifies the *contract name*; $\overline{\mathtt{A}}$ are the *parties* that can invoke contract's functions, $\overline{\mathtt{x}}$ and $\overline{\mathtt{h}}$ are the *fields* and the *assets*, respectively, and Q is the *initial state*. The contract body also includes the sequence $\overline{F}$ of functions, whose syntax is defined in Table 1. It is assumed that the names of parties, fields, assets and functions do not contain duplicates and functions' parameters do not clash with the names of contract's fields and assets.

The declaration of a function highlights the state Q when the invocation is admitted, who is the caller party A, and the list of parameters. Function's parameters are split in two lists: the *formal parameters* $\overline{\mathtt{y}}$ in brackets and the *asset parameters* $\overline{\mathtt{k}}$ in square brackets. The *body* $\{\ S\ \}\Rightarrow @\mathtt{Q}'$ specifies the *statement part* S and the state Q′ of the contract when the function execution terminates. We write $@\mathtt{Q}\,\mathtt{A}:\mathtt{f}(\overline{\mathtt{y}})\ [\overline{\mathtt{k}}]\ \{S\}\Rightarrow @\mathtt{Q}'\in\mathtt{C}$ when $\overline{F}$ contains $@\mathtt{Q}\,\mathtt{A}:\mathtt{f}(\overline{\mathtt{y}})\ [\overline{\mathtt{k}}]\ \{S\}\Rightarrow @\mathtt{Q}'$ and we will often shorten the above predicate by writing $\mathtt{Q\,A.f\,Q}'\in\mathtt{C}$. *Stipula* does not admit *internal nondeterminism*: for every Q, A and f, there is at most a Q′ such that $\mathtt{Q\,A.f\,Q}'\in\mathtt{C}$.

[1] Actually this is a lightweight version of the language in [9].

Statements S include the empty statement _ and different prefixes followed by a continuation. Prefixes P use the two symbols $\rightarrow$ (*update*) and $\multimap$ (*move*) to differentiate operations on fields and assets, respectively. The prefix $E \rightarrow \mathtt{x}$ updates the field or the parameter $\mathtt{x}$ with the value of E – the old value of $\mathtt{x}$ is lost – it is *destroyed*; $E \rightarrow \mathtt{A}$ sends the value of E to the party $\mathtt{A}$. The move operations $c \times \mathtt{h} \multimap \mathtt{h}'$ and $c \times \mathtt{h} \multimap \mathtt{A}$ define actions that *never destroy resources*. In particular, $c \times \mathtt{h} \multimap \mathtt{h}'$ subtracts the value of $c \times \mathtt{h}$ to the asset $\mathtt{h}$ and adds this value to $\mathtt{h}'$, where c is a constant with $0 < c \leq 1$. Notice that, because of this constraint, $\mathtt{h} - c \times \mathtt{h}$ is always non-negative. It is also worth to notice that, according to the syntax, the right-hand side of $\rightarrow$ in $E \rightarrow \mathtt{x}$ is always a field or a non-asset function parameter, while the right-hand side of $\multimap$ in $c \times \mathtt{h} \multimap \mathtt{h}'$ is always an asset (the left-hand side is an expression that indicates part of an asset). The operation $c \times \mathtt{h} \multimap \mathtt{A}$ subtracts the value of $c \times \mathtt{h}$ to the asset $\mathtt{h}$ and transfers it to $\mathtt{A}$.

Statements also include *conditionals* $\mathtt{if}\,(E)\,\{S\}\,\mathtt{else}\,\{S'\}$ with the standard semantics. In the rest of the paper we will always abbreviate $1 \times \mathtt{h} \multimap \mathtt{h}'$ and $1 \times \mathtt{h} \multimap \mathtt{A}$ (which are very usual, indeed) into $\mathtt{h} \multimap \mathtt{h}'$ and $\mathtt{h} \multimap \mathtt{A}$, respectively.

Expressions E include constant values v, names X of either assets, fields or parameters, and both binary and unary operations. Constant values are

- real numbers n, that are written as nonempty sequences of digits, possibly followed by "." and by a sequence of digits (*e.g.* `13` stands for `13.0`). The number may be prefixed by the sign `+` or `-`. Reals come with the standard set of binary arithmetic operations (`+`, `-`, $\times$) and the unary division operation E/c where $c \neq 0$, in order to avoid 0-division errors.
- boolean values `false` and `true`. The operations on booleans are conjunction &&, disjunction `||`, and negation `!`.
- asset values that represent *fungible* resources (*e.g.* digital currencies). Fungible asset constants are assumed to be identical to nonnegative real numbers (assets are always greater or equal to 0).

Relational operations (`<`, `>`, `<=`, `>=`, `==`) are available between any expression.

The standard definition of *free names* of expressions, statements and functions is assumed and will be denoted $fn(E)$, $fn(S)$ and $fn(F)$, respectively. A contract $\mathtt{stipula\ C\ \{\ parties}\ \overline{\mathtt{A}}\ \mathtt{fields}\ \overline{\mathtt{x}}\ \mathtt{assets}\ \overline{\mathtt{h}}\ \mathtt{init\ Q}\ \overline{F}\ \}$ is *closed* if, for every $F \in \overline{F}$, $fn(F) \subseteq \overline{\mathtt{A}} \cup \overline{\mathtt{x}} \cup \overline{\mathtt{h}}$.

We illustrate relevant features of *Stipula* by means of few examples; the examples will be also used to present liquidity and separate-liquidity. Consider the `Fill_Move` contract

```
stipula Fill_Move { parties Alice,Bob   assets hA, hB   init Q0
    @Q0 Alice: fill()[k]{   k ⊸ hA   } ⇒ @Q1
    @Q1 Bob: move()[]{   hA ⊸ hB   } ⇒ @Q0
    @Q0 Bob: end()[]{   hB ⊸ Bob   } ⇒ @Q2
}
```

that regulates interactions between `Alice` and `Bob`. It has two assets and three states `Q0`, `Q1` and `Q2`, with initial state `Q0`. In `Q0`, `Alice` may move part of her asset by invoking `fill`; the asset is stored in the formal parameter `k`. That is, the party `Alice` is assumed to own some asset and the invocation, *e.g.* `Alice.fill()[5.0]`, is removing *5* units from `Alice`'s wallet and storing them in `k`. Said otherwise, the total assets of the system are invariant during the invocation; similarly during the operations $\mathtt{h} \multimap \mathtt{h}'$. The execution of `fill` moves the assets in `k` to $\mathtt{h_A}$) and makes the contract transit to the state `Q1`. In this state, the unique admitted function is `move` by which `Bob` accumulates in $\mathtt{h_B}$ the assets sent by `Alice`. The contract's state becomes `Q0` again and the behavior may *cycle*. `Fill_Move` terminates when, in `Q0`, `Bob` decides to grab the whole content of $\mathtt{h_B}$. Notice the *nondeterministic* behavior when `Fill_Move` is in `Q0`: according to a `fill` or an `end` function is invoked, the contract may transit in `Q1` or `Q2` (this is called *external* nondeterminism in the literature [15]). Notice also that *Stipula* overlooks the details of the interactions with the parties (usually an asset transfer between the parties and the contract is mediated by a bank). [*Stipula* also features *internal nondeterminism* [15], which happens when both $\mathtt{@Q\,A : f}\,(\overline{\mathtt{y}})\ [\overline{\mathtt{k}}]\ \{S\} \Rightarrow \mathtt{@Q'} \in \mathtt{C}$ and $\mathtt{@Q\,A : f}\,(\overline{\mathtt{y}})\ [\overline{\mathtt{k}}]\ \{S'\} \Rightarrow \mathtt{@Q''} \in \mathtt{C}$. In this case, the contract may transit in `Q'` or `Q''` and the choice is casual, *cf.* rule [Function] in Table 2.]

`Fill_Move` has the property that the assets $\mathtt{h_A}$ and $\mathtt{h_B}$ are eventually emptied (whatever it is the state of the contract – in the terminology of the next section, *the contract is liquid*). This property is not always retained by *Stipula* contracts. For example, consider a `Fill_Move` contract with a programming error – call it `Fill_Move_Wrong` – where, in the `move` function, the two assets are swapped:

```
@Q1 Bob: move()[]{   hB ⊸ hA   } ⇒ @Q0
```

In this case, the asset $\mathtt{h_A}$ accumulates `Alice`'s wallet but `move` does not shift this value to $\mathtt{h_B}$ anymore. Therefore `Bob` will grab nothing upon invoking `end`. That is, the amount in $\mathtt{h_A}$ remains frozen and the contract is not liquid. In particular, since $\mathtt{h_A}$ is never emptied, it is not $\mathtt{h_A}$-separate liquid, as well.

Separate-liquidity guarantees that a single asset will be eventually emptied; this property is sometimes inadequate. Consider the following `Ping_Pong` contract

Table 2
The transition relation of *Stipula*.

$$\frac{@\mathtt{Q}\,\mathtt{A}:\mathtt{f}\,(\overline{\mathtt{y}})\;[\overline{\mathtt{k}}]\;\{\,S\,\}\Rightarrow @\mathtt{Q}'\in \mathtt{C} \qquad \ell(\mathtt{A})=A \qquad \ell'=\ell[\overline{\mathtt{y}}\mapsto\overline{u},\overline{\mathtt{k}}\mapsto\overline{v}]}{\mathtt{C}(\mathtt{Q},\,\ell,\,\_)\xrightarrow{A:\mathtt{f}(\overline{u})[\overline{v}]}\mathtt{C}(\mathtt{Q},\,\ell',\,S\Rightarrow @\mathtt{Q}')}\;[\textsc{Function}]$$

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

$$\frac{[\![E]\!]_\ell=v \qquad \ell(\mathtt{A})=A}{\mathtt{C}(\mathtt{Q},\,\ell,\,E\to\mathtt{A}\;\Sigma)\xrightarrow{v\to A}\mathtt{C}(\mathtt{Q},\,\ell,\,\Sigma)}\;[\textsc{Value-Send}]$$

$$\frac{[\![c\times\mathtt{h}]\!]_\ell=v \qquad \ell(\mathtt{A})=A \qquad [\![\mathtt{h}-v]\!]_\ell=v'}{\mathtt{C}(\mathtt{Q},\,\ell,\,c\times\mathtt{h}\multimap\mathtt{A}\;\Sigma)\xrightarrow{v\multimap A}\mathtt{C}(\mathtt{Q},\,\ell[\mathtt{h}\mapsto v'],\,\Sigma)}\;[\textsc{Asset-Send}]$$

$$\frac{[\![E]\!]_\ell=v}{\mathtt{C}(\mathtt{Q},\,\ell,\,E\to\mathtt{x}\;\Sigma)\longrightarrow\mathtt{C}(\mathtt{Q},\,\ell[\mathtt{x}\mapsto v],\,\Sigma)}\;[\textsc{Field-Update}]$$

$$\frac{[\![c\times\mathtt{h}]\!]_\ell=v \qquad [\![\mathtt{h}-v]\!]_\ell=v' \qquad [\![\mathtt{h}'+v]\!]_\ell=v'' \qquad \ell'=\ell[\mathtt{h}\mapsto v',\mathtt{h}'\mapsto v'']}{\mathtt{C}(\mathtt{Q},\,\ell,\,c\times\mathtt{h}\multimap\mathtt{h}'\;\Sigma)\longrightarrow\mathtt{C}(\mathtt{Q},\,\ell',\,\Sigma)}\;[\textsc{Asset-Update}]$$

$$\frac{[\![E]\!]_\ell\neq 0}{\mathtt{C}(\mathtt{Q},\,\ell,\,\mathtt{if}\,(E)\{S\}\mathtt{else}\{S'\}\;\Sigma)\longrightarrow\mathtt{C}(\mathtt{Q},\,\ell,\,S\;\Sigma)}\;[\textsc{Cond-true}]$$

$$\frac{[\![E]\!]_\ell=0}{\mathtt{C}(\mathtt{Q},\,\ell,\,\mathtt{if}\,(E)\{S\}\mathtt{else}\{S'\}\;\Sigma)\longrightarrow\mathtt{C}(\mathtt{Q},\,\ell,\,S'\;\Sigma)}\;[\textsc{Cond-false}]$$

```
stipula Ping_Pong { parties Amy,Mary  assets h_A, h_M  init Q0
    @Q0 Mary: ping()[u]{  h_M ⊸ Mary  u ⊸ h_A  } ⇒ @Q1
    @Q1 Amy: pong()[v]{  h_A ⊸ Amy  v ⊸ h_M  } ⇒ @Q0
}
```

It has a cyclic behavior where `Mary` and `Amy` exchange asset values. By invoking `ping`, `Mary` moves part of her wallet into `u` (and then into the asset field $\mathtt{h}_\mathtt{A}$) and grabs the value stored in $\mathtt{h}_\mathtt{M}$; conversely, by invoking `pong`, `Amy` moves part of her wallet into `v` (and then into the asset field $\mathtt{h}_\mathtt{M}$) and grabs the value stored in $\mathtt{h}_\mathtt{A}$. (Since asset fields are initially empty, the first invocation of `ping` does not deliver anything to `Mary`.) Apart the initial state, `Ping_Pong` never reaches a state where $\mathtt{h}_\mathtt{A}$ and $\mathtt{h}_\mathtt{M}$ are both empty at the same time (if `Mary` and `Amy` invocations carry nonempty assets) – *the contract is not liquid*. States have either $\mathtt{h}_\mathtt{A}$ empty – `Q0` – or $\mathtt{h}_\mathtt{M}$ empty – `Q1`. However the assets will be always grabbed by either `Mary` or `Amy`: *the contract is both* $\mathtt{h}_\mathtt{A}$*-separate liquid and* $\mathtt{h}_\mathtt{M}$*-separate liquid.*

3. Semantics

Let a *configuration*, ranged over by $\mathbb{C}$, $\mathbb{C}'$, $\cdots$, be a tuple $\mathtt{C}(\mathtt{Q},\,\ell,\,\Sigma)$ where

- `C` is the contract name and `Q` is one of its states;
- ℓ, called *memory*, is a mapping from names (parties, fields, assets and function's parameters) to values. The values of parties are noted $A, A', \cdots$. These values cannot be passed as function's parameters and cannot be hard-coded into the source contracts, since they do not belong to expressions. We write $\ell[\mathtt{h}\mapsto u]$ to specify the memory that binds `h` to u and is equal to ℓ otherwise;
- Σ is the (possibly empty) residual of a function body, *i.e.* Σ is either `_` or a term $S\Rightarrow @\mathtt{Q}$.

Configurations such as $\mathtt{C}(\mathtt{Q},\,\ell,\,\_)$, *i.e.* there is no statement to execute, are called *idle*.
We will use the auxiliary *evaluation function* $[\![E]\!]_\ell$ that returns the value of E in the memory ℓ such that:

- $[\![v]\!]_\ell = v$ for real numbers and asset values (asset values are always nonnegative), $[\![\mathtt{true}]\!]_\ell = 1$ and $[\![\mathtt{false}]\!]_\ell = 0$ (booleans are converted to reals); $[\![X]\!]_\ell = \ell(X)$ for names of assets, fields and parameters.
- let *uop* and *op* be the semantic operations corresponding to `uop` and `op`, then $[\![\mathtt{uop}\,E]\!]_\ell = uop\,v$, $[\![E\,\mathtt{op}\,E']\!]_\ell = v\;op\;v'$ with $[\![E]\!]_\ell = v$, $[\![E']\!]_\ell = v'$. In case of boolean operations, every non-null real corresponds to `true` and `0.0` corresponds to `false`; the operations return the reals for `true` and `false`. Because of the restrictions on the language, `uop` and `op` are always defined.

The semantics of *Stipula* is defined by a *transition relation*, noted $\mathbb{C}\xrightarrow{\mu}\mathbb{C}'$, that is given in Table 2, where μ is either empty or $A:\mathtt{f}(\overline{u})[\overline{v}]$ or $v\to A$ or $v\multimap A$. Rule [Function] defines invocations: the label specifies the party A performing the invocation and the function name `f` with the actual parameters. The transition may occur provided (*i*) the contract is in the state `Q` that admits invocations of `f` from A such that $\ell(\mathtt{A})=A$ and (*ii*) the configuration is *idle*. Rule [State-Change] says that a contract changes state when the execution of the statement in the function's body terminates. To keep *Stipula*'s operational semantics simple, we do not remove garbage names in the memories (the formal parameters of functions once

the functions have terminated). Therefore memories retain such names and the formal parameters keep the value they have at the end of the function execution. These values are lost when the function is called again (*cf.* rule [Function]: in ℓ', the assets $\overline{\mathtt{k}}$ are updated with $\overline{v}$). A function that does not empty asset formal parameters is clearly incorrect and the following analysis will catch such errors.

Regarding statements, we only discuss [Asset-Send] and [Asset-Update] because the other rules are standard. Rule [Asset-Send] delivers part of an asset $\mathtt{h}$ to A. This part, named v, is removed from the asset, *cf.* the memory of the right-hand side state in the conclusion. In a similar way, [Asset-Update] moves a part v of an asset $\mathtt{h}$ to an asset $\mathtt{h}'$. For this reason, the final memory becomes $\ell[\mathtt{h}\mapsto v', \mathtt{h}'\mapsto v'']$, where $v' = \ell(\mathtt{h}) - v$ and $v'' = \ell(\mathtt{h}') + v$.

A contract `stipula C { parties` $\overline{\mathtt{A}}$ `fields` $\overline{\mathtt{x}}$ `assets` $\overline{\mathtt{h}}$ `init Q` $\overline{F}$ `}` is invoked by $\mathtt{C}(\overline{A}, \overline{u})$ that corresponds to the initial configuration

$$\mathtt{C}(\mathtt{Q},\ [\overline{\mathtt{A}}\mapsto \overline{A}, \overline{\mathtt{x}}\mapsto \overline{u}, \overline{\mathtt{h}}\mapsto \overline{0}],\ \_)\ .$$

We remark that no field and asset is left uninitialized, which means that no *undefined-value* error can occur during the execution by accessing to field and assets. Notice that the initial value of assets is 0. In order to keep the notation light we always assume that parties $\overline{\mathtt{A}}$ are always instantiated by the corresponding names $\overline{A}$ written with italic fonts.

For example, a sequence of transitions of the `Fill_Move` contract is the following one. Let $\ell = [\mathtt{Alice}\mapsto \mathit{Alice}, \mathtt{Bob}\mapsto \mathit{Bob}, \mathtt{h_A}\mapsto 0, \mathtt{h_B}\mapsto 0]$. Then

$$\begin{array}{lll}
\mathtt{Fill_Move}(\mathtt{Q0},\ \ell,\ _) & & \\
\quad \xrightarrow{\mathit{Alice}:\mathtt{fill}[123]} & \mathtt{Fill_Move}(\mathtt{Q0},\ \ell[\mathtt{k}\mapsto 123],\ \mathtt{k}\multimap \mathtt{h_A}\Rightarrow @\mathtt{Q1}) & [\textsc{Function}]\\
\quad \longrightarrow & \mathtt{Fill_Move}(\mathtt{Q0},\ \ell[\mathtt{k}\mapsto 0, \mathtt{h_A}\mapsto 123],\ _\Rightarrow @\mathtt{Q1}) & [\textsc{Asset-Update}]\\
\quad \longrightarrow & \mathtt{Fill_Move}(\mathtt{Q1},\ \ell[\mathtt{k}\mapsto 0, \mathtt{h_A}\mapsto 123],\ _) & [\textsc{State-Change}]\\
\quad \xrightarrow{\mathit{Bob}:\mathtt{move}} & \mathtt{Fill_Move}(\mathtt{Q1},\ \ell[\mathtt{k}\mapsto 0, \mathtt{h_A}\mapsto 123],\ \mathtt{h_A}\multimap \mathtt{h_B}\Rightarrow @\mathtt{Q0}) & [\textsc{Function}]\\
\quad \longrightarrow & \mathtt{Fill_Move}(\mathtt{Q1},\ \ell[\mathtt{k}\mapsto 0, \mathtt{h_A}\mapsto 0, \mathtt{h_B}\mapsto 123],\ _\Rightarrow @\mathtt{Q0}) & [\textsc{Asset-Update}]\\
\quad \longrightarrow & \mathtt{Fill_Move}(\mathtt{Q0},\ \ell[\mathtt{k}\mapsto 0, \mathtt{h_A}\mapsto 0, \mathtt{h_B}\mapsto 123],\ _) & [\textsc{State-Change}]
\end{array}$$

Notice that, as discussed above, formal parameters of functions are not garbage collected when the function terminates.

Below we use the following notation and terminology:

- We write $\mathbb{C} \overset{A.\mathtt{f}(\overline{u})[\overline{v}]}{\Longrightarrow} \mathbb{C}'$ if $\mathbb{C} \overset{A.\mathtt{f}(\overline{u})[\overline{v}]}{\longrightarrow} \overset{\mu_1}{\longrightarrow} \cdots \overset{\mu_n}{\longrightarrow} \mathbb{C}'$ and μ_i are either empty or $v \multimap A$ or $v \to A$ and $\mathbb{C}'$ is idle.
- We write $\mathbb{C} \Longrightarrow \mathbb{C}'$ if $\mathbb{C} \overset{A_1.\mathtt{f}_1(\overline{u_1})[\overline{v_1}]}{\Longrightarrow} \cdots \overset{A_n.\mathtt{f}_n(\overline{u_n})[\overline{v_n}]}{\Longrightarrow} \mathbb{C}'$, for some $A_1.\mathtt{f}_1(\overline{u_1})[\overline{v_1}], \cdots, A_n.\mathtt{f}_n(\overline{u_n})[\overline{v_n}]$. $\mathbb{C} \Longrightarrow \mathbb{C}'$ will be called *computation*.

Two important properties of *Stipula* contracts follow. The first one guarantees that, in closed contracts, the invocation of a function never fails. This property immediately follows by the fact that, in such contracts, the evaluation of expressions and statements can never rise an error (operations are total, names are always bound to values and type errors cannot occur because values are always converted to reals).

Theorem 1 *(Progress). Let $\mathtt{C}$ be a closed* Stipula *contract with fields $\overline{\mathtt{x}}$, assets $\overline{\mathtt{h}}$, parties $\overline{\mathtt{A}}$ and $@\mathtt{Q}\ \ \mathtt{A}:\mathtt{f}(\overline{\mathtt{y}})\ [\overline{k}]\ \{\ S\ \}\Rightarrow @\mathtt{Q}' \in \mathtt{C}$. For every ℓ such that $\overline{\mathtt{x}}, \overline{\mathtt{h}}, \overline{\mathtt{A}} \subseteq dom(\ell)$, there is ℓ' such that $\mathtt{C}(\mathtt{Q}, \ell, \_) \overset{A.\mathtt{f}(\overline{u})[\overline{v}]}{\Longrightarrow} \mathtt{C}(\mathtt{Q}', \ell', \_)$.*

The second property guarantees the soundness of assets, *i.e.* assets always retain nonnegative values if the functions' asset parameters are nonnegative. We also remind that assets are initially zero. We say that a memory ℓ is *sound* if, for every asset $\mathtt{h}' \in dom(\ell)$, $\ell(\mathtt{h}') \geq 0$.

Theorem 2 *(Soundness of assets). Let $\mathtt{C}$ be a* Stipula *contract and $@\mathtt{Q}\ \ \mathtt{A}:\mathtt{f}(\overline{\mathtt{y}})\ [\overline{k}]\ \{\ S\ \}\Rightarrow @\mathtt{Q}' \in \mathtt{C}$. If ℓ is sound and $\overline{v} \geq \overline{0}$ and $\mathtt{C}(\mathtt{Q}, \ell, \_) \overset{A.\mathtt{f}(\overline{u})[\overline{v}]}{\Longrightarrow} \mathtt{C}(\mathtt{Q}', \ell', \_)$ then ℓ' is sound.*

We conclude with the definitions of separate liquidity and liquidity. We use the following notation:

- we write $\ell(\overline{\mathtt{h}}) > \overline{0}$ if and only if there is $\mathtt{k} \in \overline{\mathtt{h}}$ such that $\ell(\mathtt{k}) > 0$; similarly $\ell(\overline{\mathtt{h}}) = \overline{0}$ if and only if, for every $\mathtt{k} \in \overline{\mathtt{h}}$, $\ell(\mathtt{k}) = 0$.

Definition 1 *(Liquidity).* A *Stipula* contract $\mathtt{C}$ with assets $\overline{\mathtt{h}}$ and initial configuration $\mathbb{C}$ is $\mathtt{k}$-*separate liquid* ($\mathtt{k} \in \overline{\mathtt{h}}$) if, for every $\mathbb{C} \Longrightarrow \mathtt{C}(\mathtt{Q}, \ell, \_)$, then

1. $\ell(\overline{\mathtt{h}'}) = \overline{0}$ with $\overline{\mathtt{h}'} = dom(\ell) \setminus \overline{\mathtt{h}}$;
2. if $\ell(\mathtt{k}) > 0$ then there is $\mathtt{C}(\mathtt{Q}, \ell, \_) \Longrightarrow \mathtt{C}(\mathtt{Q}', \ell', \_)$ such that $\ell(\mathtt{k}) = 0$.

Table 3
The Liquidity type system of *Stipula*.

$$\frac{\mathtt{A}, fn(E) \subseteq \mathtt{X} \cup dom(\Xi)}{\Xi \vdash_{\mathtt{X}} E \to \mathtt{A} : \Xi}\ [\text{L-SEND}] \qquad \frac{\mathtt{x}, fn(E) \subseteq \mathtt{X} \cup dom(\Xi)}{\Xi \vdash_{\mathtt{X}} E \to \mathtt{x} : \Xi}\ [\text{L-UPDATE}]$$

$$\frac{\mathtt{h} \in dom(\Xi) \quad \mathtt{A} \in \mathtt{X}}{\Xi \vdash_{\mathtt{X}} \mathtt{h} \multimap \mathtt{A} : \Xi[\mathtt{h} \mapsto \mathbb{0}]}\ [\text{L-ASEND}] \qquad \frac{\mathtt{h} \in dom(\Xi) \quad c \neq 1 \quad \mathtt{A} \in \mathtt{X}}{\Xi \vdash_{\mathtt{X}} c \times \mathtt{h} \multimap \mathtt{A} : \Xi}\ [\text{L-EXPASEND}]$$

$$\frac{\mathtt{h}, \mathtt{h}' \in dom(\Xi) \quad \mathbb{e} = \Xi(\mathtt{h}) \sqcup \Xi(\mathtt{h}')}{\Xi \vdash_{\mathtt{X}} \mathtt{h} \multimap \mathtt{h}' : \Xi[\mathtt{h} \mapsto \mathbb{0}, \mathtt{h}' \mapsto \mathbb{e}]}\ [\text{L-AUPDATE}] \qquad \frac{\mathtt{h}, \mathtt{h}' \in dom(\Xi) \quad c \neq 1 \quad \mathbb{e} = \Xi(\mathtt{h}) \sqcup \Xi(\mathtt{h}')}{\Xi \vdash_{\mathtt{X}} c \times \mathtt{h} \multimap \mathtt{h}' : \Xi[\mathtt{h}' \mapsto \mathbb{e}]}\ [\text{L-EXPAUPD}]$$

$$\Xi \vdash_{\mathtt{X}} \_ : \Xi\ [\text{L-ZERO}] \qquad \frac{\Xi \vdash_{\mathtt{X}} P : \Xi' \quad \Xi' \vdash_{\mathtt{X}} S : \Xi''}{\Xi \vdash_{\mathtt{X}} P\ S : \Xi''}\ [\text{L-SEQ}]$$

$$\frac{fn(E) \subseteq \mathtt{X} \cup dom(\Xi) \quad \Xi \vdash_{\mathtt{X}} S : \Xi' \quad \Xi \vdash_{\mathtt{X}} S' : \Xi'' \quad \Xi' \sqcup \Xi'' \vdash_{\mathtt{X}} S'' : \Xi'''}{\Xi \vdash_{\mathtt{X}} \mathtt{if}\,(E)\,\{S\}\mathtt{else}\,\{S'\}\ S'' : \Xi'''}\ [\text{L-COND}]$$

$$\frac{\mathtt{A}, fn(S) \subseteq \mathtt{X} \cup \overline{\mathtt{y}} \qquad \overline{\xi}' \text{ fresh} \qquad \Xi[\overline{\mathtt{k}} \mapsto \overline{\xi}'] \vdash_{\mathtt{X} \cup \overline{\mathtt{y}}} S : \Xi'}{\Xi \vdash_{\mathtt{X}} @\mathtt{Q}\ \mathtt{A} : \mathtt{f}(\overline{\mathtt{y}})[\overline{\mathtt{k}}]\{S\} \Rightarrow @\mathtt{Q}' : \mathtt{Q}\ \mathtt{A.f}\ \mathtt{Q}' : \Xi[\overline{\mathtt{k}} \mapsto \overline{\mathbb{1}}] \to \Xi'\{\overline{\mathbb{1}}/_{\overline{\xi}'}\}}\ [\text{L-FUNCTION}]$$

$$\frac{\overline{\xi} \text{ fresh} \quad \left([\overline{\mathtt{h}} \mapsto \overline{\xi}] \vdash_{\overline{\mathtt{A}} \cup \overline{\mathtt{x}}} F : \mathcal{L}_F\right)^{F \in \overline{F}}}{\vdash \mathtt{stipula\ C}\,\{\mathtt{parties}\ \overline{\mathtt{A}}\ \mathtt{fields}\ \overline{\mathtt{x}}\ \mathtt{assets}\ \overline{\mathtt{h}}\ \mathtt{init\ Q}\ \overline{F}\,\} : \bigcup_{F \in \overline{F}} \mathcal{L}_F}\ [\text{L-CONTRACT}]$$

The contract is *liquid* if, for every computation $\mathbb{C} \Longrightarrow \mathtt{C}(\mathtt{Q}, \ell, \_)$, then

1. $\ell(\overline{\mathtt{h}'}) = \overline{0}$ with $\overline{\mathtt{h}'} = dom(\ell) \setminus \overline{\mathtt{h}}$;
2. if $\ell(\overline{\mathtt{h}}) > \overline{0}$ then there is $\mathtt{C}(\mathtt{Q}, \ell, \_) \Longrightarrow \mathtt{C}(\mathtt{Q}', \ell', \_)$ such that $\ell'(\overline{\mathtt{h}}) = \overline{0}$.

Clearly k-separate liquidity is weaker than liquidity: a k-separate liquid contract may be not liquid. For instance, the Ping_Pong contract in Section 2 is both $\mathtt{h}_{\mathtt{A}}$-separate liquid and $\mathtt{h}_{\mathtt{M}}$-separate liquid but it is not liquid. (Separate liquidity and liquidity coincide on contracts with only one asset.) It turns out that k-separate liquidity is sufficient when assets are separated, *i.e.* contracts that do not move asset fields to other assets. In fact, in this case, no asset remains frozen (assets are separated) and *the reachability of a configuration where all the assets are 0 at the same time* is not relevant.

We notice that Progress is critical for reducing (separate) liquidity to some form of reachability analysis (otherwise we should also deal with function invocations that terminate into a stuck state because of an error). In the following sections, using a symbolic technique, we define two algorithms for assessing separate liquidity and liquidity and demonstrate their correctness.

4. The theory of liquidity

We begin with the definition of the *liquidity type system* that returns an abstraction of the input-output behavior of functions with respect to assets. These abstractions record whether an asset is zero – notation $\mathbb{0}$ – or not – notation $\mathbb{1}$. The values $\mathbb{0}$ and $\mathbb{1}$ are called *liquidity values* and we use the following notation:

- *liquidity expressions* $\mathbb{e}$ are defined as follows, where ξ, ξ', $\cdots$ range over (symbolic) liquidity names:

  $$\mathbb{e} ::= \mathbb{0} \mid \mathbb{1} \mid \xi \mid \mathbb{e} \sqcup \mathbb{e} \mid \mathbb{e} \sqcap \mathbb{e}\,.$$

  They are ordered as $\mathbb{0} \leq \mathbb{e}$ and $\mathbb{e} \leq \mathbb{1}$; the operations $\sqcup$ and $\sqcap$ respectively return the maximum and the minimum value of the two arguments; they are monotonic with respect to $\leq$ (that is $\mathbb{e}_1 \leq \mathbb{e}'_1$ and $\mathbb{e}_2 \leq \mathbb{e}'_2$ imply $\mathbb{e}_1 \sqcup \mathbb{e}_2 \leq \mathbb{e}'_1 \sqcup \mathbb{e}'_2$ and $\mathbb{e}_1 \sqcap \mathbb{e}_2 \leq \mathbb{e}'_1 \sqcap \mathbb{e}'_2$). A liquidity expression that does not contain liquidity names is called *ground*. Two tuples are ordered $\leq$ if they are element-wise ordered by $\leq$.
- *environments* Ξ map contract's assets and asset parameters to liquidity expressions. Environments that map names to ground liquidity expressions are called *ground environments*.
- *liquidity function types* $\mathtt{Q\,A.f\,Q'} : \Xi \to \Xi'$ where $\Xi \to \Xi'$ records the liquidity effects of fully executing the body of $\mathtt{Q\,A.f\,Q'}$.
- *judgments* $\Xi \vdash_{\mathtt{X}} S : \Xi'$ for statements and $\Xi \vdash_{\mathtt{X}} @\mathtt{Q}\ \mathtt{A} : \mathtt{f}(\overline{\mathtt{x}})[\overline{\mathtt{h}'}]\ \{S\} \Rightarrow @\mathtt{Q}' : \mathcal{L}$ for function definitions, where $\mathcal{L}$ is a liquidity function type. The set X contains party and field names.

The liquidity type system is defined in Table 3; below we discuss the most relevant rules. Asset movements have four rules – [L-ASEND], [L-EXPASEND] [L-AUPDATE] and [L-EXPAUPD] – according to whether the constant factor is 0 or not and whether

the asset is moved to an asset or a party. According to [L-AUPDATE], the final asset environment of $\mathtt{h} \multimap \mathtt{h}'$ (which is an abbreviation for $1 \times \mathtt{h} \multimap \mathtt{h}'$) has $\mathtt{h}$ that is emptied and $\mathtt{h}'$ that gathers the value of $\mathtt{h}$, henceforth the liquidity expression $\Xi(\mathtt{h}) \sqcup \Xi(\mathtt{h}')$. Notice that, when both $\mathtt{h}$ and $\mathtt{h}'$ are $\mathbb{0}$, the overall result is $\mathbb{0}$. In the rule [L-EXPAUPD], the asset $\mathtt{h}$ is decreased by an amount that is moved to $\mathtt{h}'$. Since $c \neq 1$, the static analysis (which is independent of the runtime value of $\mathtt{h}$) can only safely assume that the asset $\mathtt{h}$ is not emptied by this operation (if it was not empty before). Therefore, after the withdraw, the liquidity value of $\mathtt{h}$ has not changed. On the other hand, the asset $\mathtt{h}'$ is increased of some amount if both c and $\mathtt{h}$ have a non zero liquidity value, henceforth the expression $\Xi(\mathtt{h}) \sqcup \Xi(\mathtt{h}')$. In particular, as before, when both $\Xi(\mathtt{h})$ and $\Xi(\mathtt{h}')$ are $\mathbb{0}$, the overall result is $\mathbb{0}$.

The rule for conditionals is [L-COND], where the operation $\sqcup$ on environments is defined pointwise by $(\Xi' \sqcup \Xi'')(\mathtt{h}) = \Xi'(\mathtt{h}) \sqcup \Xi''(\mathtt{h})$. That is, the liquidity analyzer over-approximates the final environments of $\mathtt{if}\,(E)\,\{S\}\,\mathtt{else}\,\{S'\}$ by taking the maximum values between the results of parsing S (that corresponds to a true value of E) and those of S' (that corresponds to a false value of E). Regarding E, the analyzer only verifies that its names are bound in the contract.

The rule for *Stipula* contracts is [L-CONTRACT]; it collects the liquidity function types $\mathcal{L}_i$ that describe the liquidity effects of each contract's function; each function assumes injective environments that respectively associate contract's assets with fresh symbolic names. In turn, the type produced by [L-FUNCTION] says that the complete execution of $\mathtt{Q\ A.f\ Q'}$ has liquidity effects $\Xi[\overline{\mathtt{h}'} \mapsto \overline{\mathbb{1}}] \to \Xi'\{{}^{\overline{\mathbb{1}}}/_{\overline{\xi'}}\}$, assuming that the body S of the function is typed as $\Xi[\overline{\mathtt{h}'} \mapsto \overline{\xi'}] \vdash S : \Xi'$. That is, in the conclusion of [L-FUNCTION] we replace the symbolic values of the liquidity names representing formal parameters with $\mathbb{1}$, because they may be any value when the function will be called.

Example 1. The set $\mathcal{L}$ of the `Fill_Move` contract contains the following liquidity types:

$$\begin{aligned}
\mathtt{Q0\ Alice.fill\ Q1}&: [\mathtt{h_A} \mapsto \xi_1, \mathtt{h_B} \mapsto \xi_2, \mathtt{k} \mapsto \mathbb{1}] \to [\mathtt{h_A} \mapsto \xi_1 \sqcup \mathbb{1}, \mathtt{h_B} \mapsto \xi_2, \mathtt{k} \mapsto \mathbb{0}]\\
\mathtt{Q1\ Bob.move\ Q0}&: [\mathtt{h_A} \mapsto \xi_1, \mathtt{h_B} \mapsto \xi_2] \to [\mathtt{h_A} \mapsto \mathbb{0}, \mathtt{h_B} \mapsto \xi_1 \sqcup \xi_2]\\
\mathtt{Q0\ Bob.end\ Q2}&: [\mathtt{h_A} \mapsto \xi_1, \mathtt{h_B} \mapsto \xi_2] \to [\mathtt{h_A} \mapsto \xi_1, \mathtt{h_B} \mapsto \mathbb{0}]
\end{aligned}$$

In the following we will always shorten $\vdash \mathtt{stipula\ C}\,\{\mathtt{parties}\ \overline{\mathtt{A}}\ \mathtt{fields}\ \overline{\mathtt{x}}\ \mathtt{assets}\ \overline{\mathtt{h}}\ \mathtt{init\ Q}\ \overline{F}\,\} : \mathcal{L}$ into $\vdash \mathtt{C} : \mathcal{L}$. A first property of the liquidity type system is that typed contracts are closed.

Proposition 1. *If $\vdash C : \mathcal{L}$ then C is closed.*

Therefore typed contracts own the progress property (Theorem 1). The correctness of the system in Table 3 requires the following notions:

- A *(liquidity) substitution* is a map from liquidity names to liquidity expressions (that may contain names, as well). Substitutions will be noted either σ, σ', $\cdots$ or $\{{}^{\overline{\mathbb{e}}}/_{\overline{\chi}}\}$. A substitution is *ground* when it maps liquidity names to ground liquidity expressions. For example $\{{}^{\mathbb{0},\mathbb{1}}/_{\chi,\xi}\}$ and $\{{}^{\mathbb{0}\sqcup\mathbb{1},\mathbb{1}\sqcap\mathbb{0}}/_{\chi,\xi}\}$ are ground substitutions, $\{{}^{\mathbb{0}\sqcup\chi'}/_{\chi}\}$ is not.
  We let $\sigma(\Xi)$ be the environment where $\sigma(\Xi)(\mathtt{x}) = \sigma(\Xi(\mathtt{x}))$.
- Let $[\![\mathbb{e}]\!]$ be the *partial evaluation* of $\mathbb{e}$ by applying the commutativity axioms of $\sqcup$ and $\sqcap$ and the axioms $\mathbb{0} \sqcup \mathbb{e} = \mathbb{e}$, $\mathbb{0} \sqcap \mathbb{e} = \mathbb{0}$, $\mathbb{1} \sqcup \mathbb{e} = \mathbb{1}$, $\mathbb{1} \sqcap \mathbb{e} = \mathbb{e}$. More precisely

$$[\![\mathbb{e}]\!] = \begin{cases}
\mathbb{e} & \text{if } \mathbb{e} = \mathbb{0} \text{ or } \mathbb{e} = \mathbb{1} \text{ or } \mathbb{e} = \xi\\
[\![\mathbb{e}']\!] & \text{if } (\mathbb{e} = \mathbb{e}' \sqcup \mathbb{e}'' \text{ or } \mathbb{e} = \mathbb{e}'' \sqcup \mathbb{e}') \text{ and } [\![\mathbb{e}'']\!] = \mathbb{0}\\
[\![\mathbb{e}']\!] & \text{if } (\mathbb{e} = \mathbb{e}' \sqcap \mathbb{e}'' \text{ or } \mathbb{e} = \mathbb{e}'' \sqcap \mathbb{e}') \text{ and } [\![\mathbb{e}'']\!] = \mathbb{1}\\
\mathbb{0} & \text{if } \mathbb{e} = \mathbb{e}' \sqcap \mathbb{e}'' \text{ and either } [\![\mathbb{e}']\!] = \mathbb{0} \text{ or } [\![\mathbb{e}'']\!] = \mathbb{0}\\
\mathbb{1} & \text{if } \mathbb{e} = \mathbb{e}' \sqcup \mathbb{e}'' \text{ and either } [\![\mathbb{e}']\!] = \mathbb{1} \text{ or } [\![\mathbb{e}'']\!] = \mathbb{1}\\
[\![\mathbb{e}']\!] \# [\![\mathbb{e}'']\!] & \text{if } \mathbb{e} = \mathbb{e}' \# \mathbb{e}'' \text{ and no-one of the above cases applies}\\
 & (\# \text{ is either } \sqcup \text{ or } \sqcap)
\end{cases}$$

  Notice that if $\mathbb{e}$ is ground then $[\![\mathbb{e}]\!]$ is either $\mathbb{0}$ or $\mathbb{1}$.
- When Ξ and Ξ' are ground, we write $\Xi \leq \Xi'$ if and only if, for every $\mathtt{h} \in dom(\Xi)$, $[\![\Xi(\mathtt{h})]\!] \leq [\![\Xi'(\mathtt{h})]\!]$. Observe that this implies that $dom(\Xi) \subseteq dom(\Xi')$.
- $\Xi|_{\overline{\mathtt{h}}}$ is the environment Ξ restricted to the names $\overline{\mathtt{h}}$, defined as follows

$$\Xi|_{\overline{\mathtt{h}}}(\mathtt{k}) = \begin{cases} \Xi(\mathtt{k}) & \text{if } \mathtt{k} \in \overline{\mathtt{h}} \\ \text{undefined} & \text{otherwise} \end{cases}$$

- let $\ell = [\overline{\mathtt{A}} \mapsto \overline{A}, \overline{\mathtt{x}'} \mapsto \overline{u}, \overline{\mathtt{h}'} \mapsto \overline{v}]$ be a memory, where $\overline{\mathtt{x}'}$ are contract's fields and non-asset parameters, while $\overline{\mathtt{h}'}$ are contract's assets and the asset parameters. We let $\mathbb{E}(\ell)$ be the ground environment defined as follows:

$$\mathbb{E}(\ell)(\mathtt{k}) = \begin{cases} \mathbb{0} & \text{if } \mathtt{k} \in \overline{\mathtt{h}'} \text{ and } \ell(\mathtt{k}) = 0 \\ \mathbb{1} & \text{if } \mathtt{k} \in \overline{\mathtt{h}'} \text{ and } \ell(\mathtt{k}) \neq 0 \\ \textit{undefined} & \text{otherwise} \end{cases}$$

Liquidity types are correct, as stated by the following theorem.

Theorem 3 *(Correctness of liquidity function types). Let* $\vdash C : \mathcal{L}$ *and* $\overline{\mathtt{h}}$ *be the assets of* C *and* $\mathtt{@Q\,A : f\,(\overline{y})\ [\overline{k}]\ \{\ S\ \} \Rightarrow @Q' \in C}$ *and* $\mathtt{Q\,A.f\ Q'} : \Xi \to \Xi'$ *in* $\mathcal{L}$*. If* $\overline{\mathtt{h}} \subseteq dom(\ell)$ *and* $C(\mathtt{Q}\,,\ \ell\,,\ \_) \overset{A.\mathtt{f}(\overline{u})[\overline{v}]}{\Longrightarrow} C(\mathtt{Q'}\,,\ \ell'\,,\ \_)$ *then there are* X *and* Ξ'' *such that:*

1. $\mathbb{E}(\ell[\overline{y} \mapsto \overline{u}, \overline{k} \mapsto \overline{v}]) \vdash_X S : \Xi''$;
2. $\mathbb{E}(\ell[\overline{y} \mapsto \overline{u}, \overline{k} \mapsto \overline{v}])|_{dom(\Xi)} \leq \sigma(\Xi)$ *and* $\Xi''|_{dom(\Xi)} \leq \sigma(\Xi')$*, for a ground substitution* σ*;*
3. $\mathbb{E}(\ell')|_{dom(\Xi)} \leq \Xi''$.

For example, consider the transition

$$\mathtt{Fill\_Move(Q1}\,,\ \ell\,,\ \_) \overset{Bob.\mathtt{move()[]}}{\longrightarrow} \mathtt{Fill\_Move(Q0}\,,\ \ell'\,,\ \_)$$

of `Fill_Move` where $\ell = [\mathtt{h_A} \mapsto \mathtt{25.0}, \mathtt{h_B} \mapsto \mathtt{5.0}]$ and $\ell' = [\mathtt{h_A} \mapsto \mathtt{0.0}, \mathtt{h_B} \mapsto \mathtt{30.0}]$. By definition, $\mathbb{E}(\ell) = [\mathtt{h_A} \mapsto \mathbb{1}, \mathtt{h_B} \mapsto \mathbb{1}]$. Letting $\mathtt{X} = \{\mathtt{Alice}, \mathtt{Bob}\}$, by the liquidity type system we obtain $\mathbb{E}(\ell) \vdash_X \mathtt{h_A} \multimap \mathtt{h_B} : \Xi''$, $\Xi'' = [\mathtt{h_A} \mapsto \mathbb{0}, \mathtt{h_B} \mapsto \mathbb{1} \sqcup \mathbb{1}]$. Since $\mathtt{Q1\ Bob.move\ Q0} : [\mathtt{h_A} \mapsto \xi_1, \mathtt{h_B} \mapsto \xi_2] \to [\mathtt{h_A} \mapsto \mathbb{0}, \mathtt{h_B} \mapsto \xi_1 \sqcup \xi_2]$ (see Example 1), the ground substitution σ that satisfies Theorem 3.2 is $[\xi_1 \mapsto \mathbb{1}, \xi_2 \mapsto \mathbb{1}]$ (actually, in this case, the "$\leq$" are equalities). Regarding the last item, $\mathbb{E}(\ell') = [\mathtt{h_A} \mapsto \mathbb{0}, \mathtt{h_B} \mapsto \mathbb{1}]$ and $\mathbb{E}(\ell') \leq \Xi''$ follows by definition.

A basic notion of our theory is the one of abstract computation and its liquidity type.

Definition 2. An *abstract computation* of a *Stipula* contract, ranged over by $\varphi, \varphi', \cdots$, is a finite sequence $\mathtt{Q}_1\,\mathtt{A}_1.\mathtt{f}_1\,\mathtt{Q}_2\ ;\ \cdots\ ;\ \mathtt{Q}_n\,\mathtt{A}_n.\mathtt{f}_n\,\mathtt{Q}_{n+1}$ of contract's functions, shortened into $\{\ \mathtt{Q}_i\,\mathtt{A}_i.\mathtt{f}_i\,\mathtt{Q}_{i+1}\ \}^{i\in 1..n}$. We use the notation $\mathtt{Q} \overset{\varphi}{\curvearrowright} \mathtt{Q'}$ to highlight the initial and final states of φ and we let $\{\ \mathtt{Q}_i\,\mathtt{A}_i.\mathtt{f}_i\,\mathtt{Q}_{i+1}\ \}^{i\in 1..n}$ be the abstract computation of $\big(C(\mathtt{Q}_i\,,\ \ell_i\,,\ \_) \overset{A_i:\mathtt{f}_i(\overline{u_i})[\overline{v_i}]}{\Longrightarrow} C(\mathtt{Q}_{i+1}\,,\ \ell_{i+1}\,,\ \_)\big)^{i\in 1..n}$.

An abstract computation φ is κ*-canonical* if functions occur at most κ-times in φ.

We notice that abstract computations do not mind of memories. Regarding canonical computations, every prefix of a κ-canonical computation is κ-canonical as well, including the empty computation.

Definition 3 *(Liquidity type of an abstract computation).* Let $\vdash C : \mathcal{L}$ and $\overline{\mathtt{h}}$ be the assets of C. Let also $\mathtt{Q}_i\,\mathtt{A}_i.\mathtt{f}_i\,\mathtt{Q}_{i+1} : \Xi_i \to \Xi'_i \in \mathcal{L}$ for every $i \in 1..n$. The *liquidity type* of $\varphi = \{\ \mathtt{Q}_i\,\mathtt{A}_i.\mathtt{f}_i\,\mathtt{Q}_{i+1}\ \}^{i\in 1..n}$, noted $\mathtt{L}_\varphi$, is $\Xi_1^{(b)}|_{\overline{\mathtt{h}}} \to \Xi_n^{(e)}|_{\overline{\mathtt{h}}}$ where $\Xi_1^{(b)}$ and $\Xi_n^{(e)}$ ("b" stays for *begin*, "e" stays for *end*) are defined as follows

$$\Xi_1^{(b)} = \Xi_1 \qquad \Xi_{i+1}^{(b)} = \Xi_{i+1}\{^{\Xi_i^{(e)}(\overline{\mathtt{h}})}/_{\overline{\xi}}\} \qquad \Xi_i^{(e)} = \Xi'_i\{^{\Xi_i^{(b)}(\overline{\mathtt{h}})}/_{\overline{\xi}}\}\,.$$

Notice that, by definition, the initial environment of the i-th type is updated so that it maps assets to the values computed at the end of the $(i-1)$-th transition. These values are also propagated to the final environment of the i-th transitions by substituting the occurrence of a liquidity name with the computed value of the corresponding asset. Notice also that the domains of the environments $\Xi_i^{(b)}$, $1 \leq i \leq n$, are in general different because they are also defined on the asset parameters of the corresponding function. However, formal parameters are not relevant because they are always replaced by $\mathbb{1}$ and are therefore dropped in the liquidity types of computations.

For example, consider the computation of the `Fill_Move` contract

$$\varphi = \mathtt{Q0\ Alice.fill\ Q1\ ;\ Q1\ Bob.move\ Q0\ ;\ Q0\ Bob.end\ Q2}$$

(we refer to Example 1 for the types of the contract). Let $H = \{\mathtt{h_A}, \mathtt{h_B}\}$ and $\Xi = [\mathtt{h_A} \mapsto \xi_1, \mathtt{h_B} \mapsto \xi_2]$. φ has liquidity type $\Xi_1^{(b)}|_H \to \Xi_3^{(e)}|_H$ where:

$$\begin{array}{lll}
\Xi_1^{(b)} & = & \Xi[\mathtt{k} \mapsto \mathbb{1}] \\
\Xi_2^{(b)} & = & \Xi\{^{\xi_1 \sqcup \mathbb{1}}/_{\xi_1}\} \\
 & = & \Xi[\mathtt{h_A} \mapsto \xi_1 \sqcup \mathbb{1}] \\
\Xi_3^{(b)} & = & \Xi\{^{\mathbb{0}, \xi_1 \sqcup \xi_2 \sqcup \mathbb{1}}/_{\xi_1, \xi_2}\} \\
 & = & [\mathtt{h_A} \mapsto \mathbb{0}, \mathtt{h_B} \mapsto \xi_1 \sqcup \xi_2 \sqcup \mathbb{1}]
\end{array}
\qquad
\begin{array}{lll}
\Xi_1^{(e)} & = & \Xi[\mathtt{h_A} \mapsto \xi_1 \sqcup \mathbb{1}, \mathtt{k} \mapsto \mathbb{0}] \\
\Xi_2^{(e)} & = & [\mathtt{h_A} \mapsto \mathbb{0}, \mathtt{h_B} \mapsto \xi_1 \sqcup \xi_2]\{^{\xi_1 \sqcup \mathbb{1}}/_{\xi_1}\} \\
 & = & [\mathtt{h_A} \mapsto \mathbb{0}, \mathtt{h_B} \mapsto \xi_1 \sqcup \xi_2 \sqcup \mathbb{1}] \\
\Xi_3^{(e)} & = & [\mathtt{h_A} \mapsto \mathbb{0}, \mathtt{h_B} \mapsto \xi_2]\{^{\mathbb{0}, \xi_1 \sqcup \xi_2 \sqcup \mathbb{1}}/_{\xi_1, \xi_2}\} \\
 & = & [\mathtt{h_A} \mapsto \mathbb{0}, \mathtt{h_B} \mapsto \mathbb{0}]
\end{array}$$

Therefore $\mathtt{L}_\varphi = \Xi \to [\mathtt{h_A} \mapsto \mathbb{0}, \mathtt{h_B} \mapsto \mathbb{0}]$. That is, whatever they are the initial values of $\mathtt{h_A}$ and $\mathtt{h_B}$, which are represented in Ξ by the liquidity names ξ_1 and ξ_2, respectively, their liquidity values after the computation φ are $\mathbb{0}$ (henceforth they are 0 by the following Theorem 4). Notice also the differences between $\mathtt{L}_\varphi$ and $\Xi \to [\mathtt{h_A} \mapsto \xi_1, \mathtt{h_B} \mapsto \mathbb{0}]$, which is the type of `Bob.end`: from this last type we may derive that $\mathtt{h_B}$ is $\mathbb{0}$ in the final environment, while the value of $\mathtt{h_A}$ is the same of the initial environment (in fact, `Bob.end` only empties $\mathtt{h_B}$ and does not access to $\mathtt{h_A}$ at all).

We recall that the operational semantics of *Stipula* in Table 2 does not remove garbage names in the memories (the formal parameters of functions once the functions have terminated, see Section 3). However, these names do not exist in environments of the liquidity types of abstract computations. For this reason, in the following statement, we restrict the inequalities to the names of the contract's assets.

Theorem 4 *(Correctness of liquidity types of abstract computations). Let* $\vdash C : \mathcal{L}$ *and* $\left(C(\mathtt{Q}_i\,,\,\ell_i\,,\,\_) \overset{A_i:\mathtt{f}_i(\overline{u_i})[\overline{v_i}]}{\Longrightarrow} C(\mathtt{Q}_{i+1},\ell_{i+1}\,,\,\_)\right)^{i\in 1..n}$ *with* $dom(\ell_1)$ *containing the assets* $\overline{\mathtt{h}}$ *of* C*. Let also* $\varphi = \{\, \mathtt{Q}_i\ \mathtt{A}_i.\mathtt{f}_i\ \mathtt{Q}_{i+1}\,\}^{i\in 1..n}$ *have liquidity type* $\mathtt{L}_\varphi = \Xi \to \Xi'$*.*

Then there is a substitution σ *such that* $\mathbb{E}(\ell_1)|_{\overline{\mathtt{h}}} \le \sigma(\Xi)$ *and* $\mathbb{E}(\ell_{n+1})|_{\overline{\mathtt{h}}} \le \sigma(\Xi')$*.*

The following proposition is used in the definition of our algorithms.

Proposition 2. *Let* $\Xi = [\overline{\mathtt{h}} \mapsto \overline{\xi}]$*, where* $\overline{\xi}$ *is a tuple of pairwise different symbolic names,* $\Xi \vdash_{\mathtt{X}} S : \Xi'$ *and* $[\![\Xi(\mathtt{h})]\!] \neq [\![\Xi'(\mathtt{h})]\!]$*. Then* S *either contains* $\mathtt{h} \multimap \mathtt{h}'$ *or* $\mathtt{h}' \multimap \mathtt{h}$ *or* $c \times \mathtt{h}' \multimap \mathtt{h}$ *or* $\mathtt{h} \multimap \mathtt{A}$ *(we say that* h *has been updated in* S*).*

Similarly, if $\vdash C : \mathcal{L}$ *and* $\mathtt{L}_\varphi = \Xi \to \Xi'$ *for an abstract computation* φ *and* $[\![\Xi(\mathtt{h})]\!] \neq [\![\Xi'(\mathtt{h})]\!]$ *then* h *has been updated by (the body of) at least one of the functions in* φ*.*

By Proposition 2, given a liquidity type $\Xi \to \Xi'$, $[\![\Xi(\mathtt{h})]\!] \neq [\![\Xi'(\mathtt{h})]\!]$ is a sufficient condition of the fact that h is updated in the corresponding computation. Of course, $[\![\Xi(\mathtt{h})]\!] = [\![\Xi'(\mathtt{h})]\!]$ does not imply that h has not been modified by the computation. For example, according to the rules in Table 3, if $\Xi \vdash_{\mathtt{X}} c \times \mathtt{h} \multimap \mathtt{h}' : \Xi'$ then $\Xi(\mathtt{h}) = \Xi'(\mathtt{h})$. That is, the liquidity type system does not record move operations that reduce an asset without emptying it.

5. The algorithms for separate liquidity and liquidity

Analyzing the liquidity of a *Stipula* contract amounts to verifying the two constraints of Definition 1. In both cases, checking constraint 1 is not difficult: for every transition $\mathtt{Q}\,\mathtt{A}.\mathtt{f}\,\mathtt{Q}'$ of the contract with assets $\overline{\mathtt{h}}$, we consider its liquidity type $\Xi \to \Xi'$ and verify whether, for every parameter $\mathtt{k} \notin \overline{\mathtt{h}}$, $[\![\Xi'(\mathtt{k})]\!] = \mathbb{0}$. Since there are finitely many transitions, this analysis is exhaustive. The correctness is the following: if $\mathtt{k} \notin \overline{\mathtt{h}}$ implies $[\![\Xi'(\mathtt{k})]\!] = \mathbb{0}$ then, for every substitution σ, $[\![\sigma(\Xi')]\!](\mathtt{k}) = \mathbb{0}$. Specifically for the substitution σ' such that $\mathbb{E}(\ell') \le [\![\sigma'(\Xi')]\!]$, which is guaranteed by Theorem 3.

On the contrary, verifying the constraints 2 of Definition 1 is harder because the transition system of a *Stipula* contract may be complex (cycles, absence of final states, nondeterminism). We design two algorithms (for both separate liquidity and liquidity) that have different precisions and different computational costs.

We first define the notions of reachable function and reachable state of a *Stipula* contract C. Let $\vdash \mathtt{C} : \mathcal{L}$; $\mathbb{Q}_\mathtt{Q}$ is the least set such that

1. if $\mathtt{Q}\,\mathtt{A}.\mathtt{f}\,\mathtt{Q}' : \Xi \to \Xi' \in \mathcal{L}$ then $\mathtt{Q}\,\mathtt{A}.\mathtt{f}\,\mathtt{Q}' : \Xi \to \Xi' \in \mathbb{Q}_\mathtt{Q}$;
2. if $\mathtt{Q}'\,\mathtt{B}.\mathtt{g}\,\mathtt{Q}'' : \Xi \to \Xi' \in \mathbb{Q}_\mathtt{Q}$ and $\mathtt{Q}''\,\mathtt{B}'.\mathtt{g}'\,\mathtt{Q}''' : \Xi'' \to \Xi''' \in \mathcal{L}$ then $\mathtt{Q}''\,\mathtt{B}'.\mathtt{g}'\,\mathtt{Q}''' : \Xi'' \to \Xi''' \in \mathbb{Q}_\mathtt{Q}$.

That is $\mathbb{Q}_\mathtt{Q}$ contains all the liquidity function types of functions whose initial state is *reachable* through computations starting at Q. In short, when $\mathtt{Q}'\,\mathtt{B}.\mathtt{g}\,\mathtt{Q}'' : \Xi \to \Xi' \in \mathbb{Q}_\mathtt{Q}$, we say that both $\mathtt{Q}'$ and $\mathtt{Q}''$ are *reachable* from Q. For example, in the `Fill_Move` contract, the set $\mathbb{Q}_{\mathtt{Q}0}$ contains the liquidity function types listed in Example 1, while $\mathbb{Q}_{\mathtt{Q}2} = \emptyset$.

Notice that, (*i*) $\mathbb{Q}_\mathtt{Q}$ is finite, (*ii*) if $\mathtt{Q}'$ is reachable from Q, then $\mathbb{Q}_{\mathtt{Q}'} \subseteq \mathbb{Q}_\mathtt{Q}$, (*iii*) if no function starts at Q then $\mathbb{Q}_\mathtt{Q} = \emptyset$ and (*iv*) if Q is the initial state and every state is reachable, then $\mathbb{Q}_\mathtt{Q}$ contains all the functions of the contract.

Below, without loss of generality, we assume that every state in the contract is reachable from the initial state. A straightforward optimization allows us to reduce to this case. We also assume that our contracts satisfy item 1 of separate liquidity and liquidity. Therefore we focus on item 2.

5.1. The efficient algorithms

Our first two algorithms verify a stronger property than separate liquidity and liquidity. We use the following terminology:

- A function $\mathtt{Q}\,\mathtt{A}.\mathtt{f}\,\mathtt{Q}' \in \mathtt{C}$ *empties* $\overline{\mathtt{k}}$ if, for every memory ℓ and every $\mathtt{C}(\mathtt{Q},\ell,\_) \overset{A.f(\overline{u})[\overline{v}]}{\Longrightarrow} \mathtt{C}(\mathtt{Q}',\ell',\_)$, we have $\ell'(\overline{\mathtt{k}}) = \overline{0}$.

Definition 4. A *Stipula* contract C with assets $\overline{\mathtt{h}}$ and initial configuration $\mathbb{C}$ is

k-separate liquid+: if, for every $\mathbb{C} \Longrightarrow \mathtt{C}(\mathtt{Q},\ell,\_)$ with $\ell(\mathtt{k}) > 0$, there are $\mathtt{C}(\mathtt{Q},\ell,\_) \Longrightarrow \mathtt{C}(\mathtt{Q}',\ell',\_)$ and $\mathtt{Q}'\,\mathtt{A}.\mathtt{f}\,\mathtt{Q}'' \in \mathtt{C}$ such that $\mathtt{Q}'\,\mathtt{A}.\mathtt{f}\,\mathtt{Q}''$ empties k.

liquid+: if, for every $\mathbb{C} \Longrightarrow \mathtt{C}(\mathtt{Q},\ell,\_)$ with $\ell(\overline{\mathtt{h}}) > 0$, there are $\mathtt{C}(\mathtt{Q},\ell,\_) \Longrightarrow \mathtt{C}(\mathtt{Q}',\ell',\_)$ and $\mathtt{Q}'\,\mathtt{A}.\mathtt{f}\,\mathtt{Q}'' \in \mathtt{C}$ such that $\mathtt{Q}'\,\mathtt{A}.\mathtt{f}\,\mathtt{Q}''$ empties $\overline{\mathtt{h}}$.

Table 4
The efficient algorithms – $\mathcal{Z}$ contains pairs $(\mathtt{Q}, \mathtt{k})$ for k-separate liquidity+ and pairs $(\mathtt{Q}, \overline{\mathtt{h}})$ for liquidity+.

Let Q be the initial state of C whose assets are $\overline{\mathtt{h}}$:	
step 1.	Compute $\mathbb{Q}_{\mathtt{Q}'}$ for every $\mathtt{Q}'$ reachable from Q; set $\mathcal{Z} = \emptyset$.
step 2.	For every $\mathtt{Q}'\ \mathtt{A.f}\ \mathtt{Q}'' : \Xi \to \Xi' \in \mathbb{Q}_{\mathtt{Q}}$ and $\mathtt{k} \in \overline{\mathtt{h}}$ such that
	(a) $[\![\Xi'(\mathtt{k})]\!] \neq \mathbb{0}$ and $[\![\Xi'(\mathtt{k})]\!] \neq [\![\Xi(\mathtt{k})]\!]$
	(b) $(\mathtt{Q}'', \mathtt{k}) \notin \mathcal{Z}$ (respectively, $(\mathtt{Q}'', \overline{\mathtt{h}}) \notin \mathcal{Z}$):
	2.1 if there is no $\mathtt{Q}'\ \mathtt{A.f}\ \mathtt{Q}'' : \Xi \to \Xi' \in \mathbb{Q}_{\mathtt{Q}}$ and k then exit: *the contract is k-separate liquid+* (respectively, *the contract is liquid+*);
	2.2 otherwise, verify whether there is $\mathtt{Q1\ B.g\ Q2} : \Xi_1 \to \Xi_2 \in \mathbb{Q}_{\mathtt{Q}''}$ such that $[\![\Xi_2(\mathtt{k})]\!] = \mathbb{0}$ (respectively, $[\![\Xi_2(\overline{\mathtt{h}})]\!] = \overline{\mathbb{0}}$). If this is the case, add $(\mathtt{Q}'', \mathtt{k})$ (respectively, add $(\mathtt{Q}'', \overline{\mathtt{h}})$) to $\mathcal{Z}$ and reiterate **step 2**, otherwise exit: *the contract is not k-separate liquid+* (respectively, *the contract is not liquid+*).

k-separate liquidity+ and liquidity+ require that the properties hold *for every memory*, while the corresponding k-separate liquidity and liquidity require the properties hold for *at least one memory*. (The former properties are stronger than the latter ones, respectively.)

Proposition 3. *Let* $\vdash C : \mathcal{L}$*:*

1. *if* $\mathtt{Q\ A.f}\ \mathtt{Q}' : \Xi \to \Xi' \in \mathcal{L}$ *and* $\Xi'(\overline{\mathtt{k}}) \leq \overline{\mathbb{0}}$ *then, for every* $C(\mathtt{Q}, \ell, \_) \overset{A:\mathtt{f}(\overline{u})[\overline{v}]}{\Longrightarrow} C(\mathtt{Q}', \ell', \_)$, $\ell'(\overline{\mathtt{k}}) = \overline{\mathbb{0}}$*;*
2. *if* C *is k-separate liquid+ then it is k-separate liquid; if* C *is liquid+ then it is liquid.*

Table 4 reports the algorithms for k-separate liquidity+ and liquidity+. We comment the k-separate liquidity+ algorithm, the other one is similar. In this case, the set $\mathcal{Z}$ contains all the verifications that have been already done; that is if $(\mathtt{Q}, \mathtt{k}) \in \mathcal{Z}$ then it has been verified that there is a function type in $\mathbb{Q}_{\mathtt{Q}}$ such that k is $\mathbb{0}$ in its final environment. The analysis ends when $\mathcal{Z}$ does not increase anymore with output k-separate liquid+/liquid+.

In the step 2(a), following Proposition 2 in Appendix B, the algorithm verifies $[\![\Xi'(\mathtt{k})]\!] \neq [\![\Xi(\mathtt{k})]\!]$ to check whether a function $\mathtt{Q}'\ \mathtt{A.f}\ \mathtt{Q}''$ *updates* the value of an asset. Additionally, it also verifies that $[\![\Xi'(\mathtt{k})]\!] \neq \mathbb{0}$ because we are interested in updates that do not empty the assets. Notice that there is a mismatch with the definition of separate liquidity+ that looks for assets' values greater than 0. We explain the mismatch with an example: consider the liquidity type $\Xi \to \Xi'$ of `Bob.end` in Example 1. We have $\Xi(\mathtt{h_A}) = \Xi'(\mathtt{h_A}) = \xi_2$ then this function does not modify $\mathtt{h_A}$ (this is a property of the system in Table 3). Therefore, regarding $\mathtt{h_A}$, `Bob.end` does not play any role: if some function updates $\mathtt{h_A}$ then we must look elsewhere for a function turning $\mathtt{h_A}$ to 0. Notice also that $\Xi(\mathtt{h_A})$ and $\Xi'(\mathtt{h_A})$ are both different from $\mathbb{0}$. Finally we remark that the algorithm takes $[\![\Xi'(\mathtt{h})]\!]$ instead of $\Xi'(\mathtt{h})$ in order to discard ground expressions that are syntactically different but semantically equivalent to $\mathbb{0}$.

Once a state $\mathtt{Q}''$ is found where k has been updated and the pair $(\mathtt{Q}'', \mathtt{k})$ does not belong to $\mathcal{Z}$, k-separate liquidity+ amounts to looking for a function in $\mathbb{Q}_{\mathtt{Q}''}$ that empties k. This is what is specified in 2.2. For example, consider the liquidity types of the `Fill_Move` contract in Example 1. There are two problematic types $\mathtt{Q0\ Alice.fill\ Q1} : \Xi_1 \to \Xi_2$ and $\mathtt{Q1\ Bob.move\ Q0} : \Xi'_1 \to \Xi'_2$. In particular, $\Xi_2(\mathtt{h_A}) \neq \Xi_1(\mathtt{h_A})$ and $\Xi'_2(\mathtt{h_B}) \neq \Xi'_1(\mathtt{h_B})$ (all the values are not-$\mathbb{0}$). In these cases, the algorithm finds two liquidity function types, one that is reachable from `Q1` and one that is reachable from `Q0` that empty the corresponding assets. The reader is invited to verify that these are the types of `Q1 Bob.move Q0` and `Q0 Bob.end Q2`, respectively.

Proposition 4. *Let* $\vdash C : \mathcal{L}$*. If the algorithm of Table 4 returns that* C *is k-separate liquid+ (respectively, liquid+) then it is k-separate liquid+ (respectively, liquid+). Additionally, the algorithms always terminate.*

Since k-separate liquidity+ and liquidity+ have been advocated for the efficiency of the analysis, we conclude by computing their cost. Let n be the size of the *Stipula* contract (the number of instructions of the contract), m be the number of states and m' be the number of functions. Then

- the cost of the inference of liquidity types is linear with respect to the size of the contract, *i.e.* $O(n)$;
- the cost of computing $\mathbb{Q}_{\mathtt{Q}}$, for every Q is $O(m \times m')$ (with a straightforward fixpoint technique);
- the cost for verifying step 2 of the algorithm is $O(m' \times m')$ because, for every function of step 2 it looks for another function satisfying 2.2.

Therefore the overall cost of the weak algorithm is $O(n + (m + m') \times m')$, which means it is $O(m'^2)$, assuming that m and m' are in linear relation.

Table 5
The costly algorithm for liquidity – $\mathcal{Z}$ contains pairs $(\mathtt{Q}, \overline{\mathtt{k}})$.

Let $\mathtt{Q}$ be the initial state of $\mathtt{C}$ whose assets are $\overline{\mathtt{h}}$.
step 1. Compute $\mathbb{T}^{\kappa}_{\mathtt{Q}'}$ for every $\mathtt{Q}'$ reachable from $\mathtt{Q}$; let $\mathcal{Z} = \emptyset$.
step 2. For every $\mathtt{Q}'$ and $\mathtt{Q}' \overset{\varphi}{\curvearrowright} \mathtt{Q}'' : \Xi \to \Xi' \in \mathbb{T}^{\kappa}_{\mathtt{Q}'}$ and $\emptyset \subsetneq \overline{\mathtt{k}} \subseteq \overline{\mathtt{h}}$ is the maximal set such that (*a*) for every $\mathtt{k}' \in \overline{\mathtt{k}}$, $[\![\Xi'(\mathtt{k}')]\!] \neq \mathbb{0}$ and $[\![\Xi'(\mathtt{k}')]\!] \neq [\![\Xi(\mathtt{k}')]\!]$ (*b*) $(\mathtt{Q}'', \overline{\mathtt{k}}) \notin \mathcal{Z}$:
2.1 If there is no $\mathtt{Q}'$, $\mathtt{Q}' \overset{\varphi}{\curvearrowright} \mathtt{Q}'' : \Xi \to \Xi'$ and $\overline{\mathtt{k}}$ then exit: *the contract is liquid.*
2.2 otherwise verify whether there is $\mathtt{Q}'' \overset{\varphi'}{\curvearrowright} \mathtt{Q}''' : \Xi'' \to \Xi''' \in \mathbb{T}^{\kappa}_{\mathtt{Q}''}$ such that $[\![\Xi'''(\overline{\mathtt{k}})]\!] = \overline{\mathbb{0}}$ and, for every $\mathtt{k}' \in \overline{\mathtt{h}} \setminus \overline{\mathtt{k}}$, either $[\![\Xi'''(\mathtt{k}')]\!] = \mathbb{0}$ or $[\![\Xi'''(\mathtt{k}')]\!] = [\![\Xi''(\mathtt{k}')]\!]$. If this is the case, add $(\mathtt{Q}'', \overline{\mathtt{k}})$ to $\mathcal{Z}$ and reiterate **step 2**, otherwise exit: *the contract is not liquid.*

5.2. The costly algorithms

Verifying $\mathtt{k}$-separate liquidity and liquidity is more complex because a single asset or a tuple of assets may become 0 during *a computation*, rather than just one transition. Let us discuss the case of liquidity with an example. Consider the `Ugly` contract with assets `w1` and `w2` and functions:

```
@Q0 Mark: get()[u]{   u ⊸ w2   } ⇒ @Q1
@Q1 Sam: shift()[]{   w1 ⊸ Sam   w2 ⊸ w1   } ⇒ @Q1
@Q1 Sam: end()[]{  } ⇒ @Q2
```

`Ugly` has no function that, *for every memory* ℓ, empties both `w1` and `w2`. In fact, the contract is not liquid+ (this contract is not `w1`-separate liquid+, as well. However there is a liquid computation (a computation that empties all the assets), which is the one invoking `shift` two times: `Q1 Sam.shift Q1; Q1 Sam.shift Q1`. In particular, we have

$$\mathtt{Q1\ Sam.shift\ Q1} : [\mathtt{w1} \mapsto \xi_1, \mathtt{w2} \mapsto \xi_2] \to [\mathtt{w1} \mapsto \mathbb{0} \sqcup \xi_2, \mathtt{w2} \mapsto \mathbb{0}]$$

$$\mathtt{Q1\ Sam.shift\ Q1\ ;\ Q1\ Sam.shift\ Q1} : [\mathtt{w1} \mapsto \xi_1, \mathtt{w2} \mapsto \xi_2] \to [\mathtt{w1} \mapsto \mathbb{0}, \mathtt{w2} \mapsto \mathbb{0}]$$

(we have simplified the final environment). That is, in this case, liquidity requires the analysis of 2-canonical computations to be assessed. (When the contract has no cycle, 1-canonical computations are sufficient to verify liquidity.) Since we have to consider cycles, in order to force termination of the analysis, we restrict to κ-canonical abstract computations (with a finite value of κ).

Let $\mathbb{T}^{\kappa}_{\mathtt{Q}}$ be *the set of elements* $\mathtt{Q} \overset{\varphi}{\curvearrowright} \mathtt{Q}' : \mathtt{L}_{\varphi}$ where φ is a κ-canonical computation starting at $\mathtt{Q}$ in the contract (the contract is left implicit).

The costly algorithm for liquidity is reported in Table 5. It uses the set $\mathbb{T}^{\kappa}_{\mathtt{Q}'}$, for every state $\mathtt{Q}'$ of the contract that is reachable from $\mathtt{Q}$ – see step 1 of Step 2 identifies the "critical pairs" $(\mathtt{Q}'', \overline{\mathtt{k}})$ such that there is a computation updating the assets $\overline{\mathtt{k}}$ and terminating in the state $\mathtt{Q}''$. Assume that $(\mathtt{Q}'', \overline{\mathtt{k}}) \notin \mathcal{Z}$. Then we must find $\mathtt{Q}'' \overset{\varphi'}{\curvearrowright} \mathtt{Q}''' : \Xi'' \to \Xi'''$ in $\mathbb{T}^{\kappa}_{\mathtt{Q}''}$ such that $\Xi'''(\overline{\mathtt{k}}) = \overline{\mathbb{0}}$ and the other assets in $\overline{\mathtt{h}} \setminus \overline{\mathtt{k}}$ are either $\mathbb{0}$ or equal to the corresponding value in Ξ''. That is, as for the efficient algorithms, assets $\overline{\mathtt{h}} \setminus \overline{\mathtt{k}}$ have not been modified by φ' and may be overlooked. Notice that these checks are exactly those defined in step 2.2. If no liquidity type $\mathtt{Q}'' \overset{\varphi'}{\curvearrowright} \mathtt{Q}''' : \Xi'' \to \Xi'''$ is found in $\mathbb{T}^{\kappa}_{\mathtt{Q}''}$ such that $\Xi'''(\overline{\mathtt{k}}) = \mathbb{0}$, the liquidity cannot be guaranteed and the algorithm exits stating that the contract is not liquid (which might be a false negative because the liquidity type might exist in $\mathbb{T}^{\kappa+1}_{\mathtt{Q}''}$).

For example, in case of the `Fill_Move` contract, the liquidity algorithm spots $\mathtt{Q0} \overset{\mathtt{Alice.fill}}{\curvearrowright} \mathtt{Q1} : \Xi \to \Xi'$ because $[\![\Xi'(\mathtt{h_A})]\!] \neq [\![\Xi(\mathtt{h_A})]\!]$. Therefore it parses the liquidity types in $\mathbb{T}^{1}_{\mathtt{Q1}}$ and finds the type $\mathtt{Q1} \overset{\varphi'}{\curvearrowright} \mathtt{Q2} : [\mathtt{h_A} \mapsto \xi_1, \mathtt{h_B} \mapsto \xi_2] \to [\mathtt{h_A} \mapsto \mathbb{0}, \mathtt{h_B} \mapsto \mathbb{0}]$, where $\varphi' = \mathtt{Q1\ Bob.move\ Q0\ ;\ Q0\ Bob.end\ Q2}$. Henceforth $(\mathtt{Q1}, \mathtt{h_A})$ is added to $\mathcal{Z}$. There is also another problematic type: $\mathtt{Q1} \overset{\mathtt{Bob.move}}{\curvearrowright} \mathtt{Q0} : \Xi'' \to \Xi'''$, because $[\![\Xi'''(\mathtt{h_A})]\!] \neq [\![\Xi''(\mathtt{h_A})]\!]$. In this case, the liquidity type of the abstract computation `Q0 Bob.end Q2` (still in $\mathbb{T}^{1}_{\mathtt{Q0}}$) satisfies the liquidity constraint. We leave this check to the reader.

The correctness of the costly algorithm follows from Theorem 4 (the formal proof is in Appendix C). For example, assume that step 2.2 returns $\mathtt{Q} \overset{\varphi}{\curvearrowright} \mathtt{Q}' : \Xi \to \Xi'$ where $\Xi'(\overline{\mathtt{k}}) = \overline{\mathbb{0}}$ and $\Xi'(\overline{\mathtt{h}} \setminus \overline{\mathtt{k}}) = \Xi(\overline{\mathtt{h}} \setminus \overline{\mathtt{k}})$. Then, for every initial memory ℓ, the concrete computation corresponding to φ ends in a memory ℓ' such that $\mathbb{E}(\ell')|_{\overline{\mathtt{k}}} \leq [\![\sigma(\Xi'|_{\overline{\mathtt{k}}})]\!] = [\![\sigma([\overline{\mathtt{k}} \mapsto \overline{\mathbb{0}}])]\!] = [\overline{\mathtt{k}} \mapsto \overline{\mathbb{0}}]$ (for every σ) and $\mathbb{E}(\ell')|_{\overline{\mathtt{h}} \setminus \overline{\mathtt{k}}} \leq [\![\sigma(\Xi'|_{\overline{\mathtt{h}} \setminus \overline{\mathtt{k}}})]\!] = [\![\sigma([\overline{\mathtt{h}} \setminus \overline{\mathtt{k}} \mapsto \overline{\xi}])]\!] = [\overline{\mathtt{k}} \mapsto \overline{\mathbb{1}}]$, by taking a $\sigma = [\overline{\xi} \mapsto \mathbb{1}]$. As regards termination, the set $\mathcal{Z}$ increases at every iteration. When no other pair can be added to $\mathcal{Z}$ (and we have not already exited) the algorithm terminates by declaring the contract as liquid.

Proposition 5. *Let* $\vdash \mathtt{C} : \mathcal{L}$. *If the algorithm of Table 5 returns that* $\mathtt{C}$ *is liquid then it is liquid. Additionally, the algorithm always terminates.*

Table 6 reports the costly algorithm for $\mathtt{k}$-separate liquidity. We omit comments about the steps of the algorithm because they are similar to those of the algorithm in Table 5, actually simpler because here we are considering one asset only.

Table 6
The costly algorithm for $\mathtt{k}$-separate liquidity – $\mathcal{Z}$ contains pairs $(\mathtt{Q}, \mathtt{k})$.

Let $\mathtt{Q}$ be the initial state of $\mathtt{C}$ whose assets are $\overline{\mathtt{h}}$.	
step 1.	Compute $\mathbb{T}^{\kappa}_{\mathtt{Q}'}$ for every $\mathtt{Q}'$ reachable from $\mathtt{Q}$; let $\mathcal{Z} = \emptyset$.
step 2.	For every $\mathtt{Q}'$ and $\mathtt{Q}' \overset{\varphi}{\curvearrowright} \mathtt{Q}'' : \Xi \to \Xi' \in \mathbb{T}^{\kappa}_{\mathtt{Q}'}$ such that (a) $[\![\Xi'(\mathtt{k})]\!] \neq \mathbb{0}$ and $[\![\Xi'(\mathtt{k})]\!] \neq [\![\Xi(\mathtt{k})]\!]$ (b) $(\mathtt{Q}'', \mathtt{k}) \notin \mathcal{Z}$:
2.1	If there is no $\mathtt{Q}'$ and $\mathtt{Q}' \overset{\varphi}{\curvearrowright} \mathtt{Q}'' : \Xi \to \Xi'$ then exit: *the contract is* $\mathtt{k}$*-separate liquid*.
2.2	otherwise verify whether there is $\mathtt{Q}'' \overset{\varphi'}{\curvearrowright} \mathtt{Q}''' : \Xi'' \to \Xi''' \in \mathbb{T}^{\kappa}_{\mathtt{Q}''}$ such that $[\![\Xi'''(\mathtt{k})]\!] = \mathbb{0}$. If this is the case, add $(\mathtt{Q}'', \overline{\mathtt{k}})$ to $\mathcal{Z}$ and reiterate **step 2**, otherwise exit: *the contract is not* $\mathtt{k}$*-separate liquid*.

The correctness and termination of the costly algorithm for $\mathtt{k}$-separate liquidity are proved in a similar way to Proposition 5, therefore they are omitted. We only report the statement.

Proposition 6. *If the algorithm of Table 6 returns that a* Stipula *contract is k-separate liquid then it is k-separate liquid. Additionally, the algorithm always terminates.*

The computational costs of liquidity and $\mathtt{k}$-separate liquidity are way larger than the algorithm in Table 4. Let n be the size of the *Stipula* contract (the number of functions, prefixes and conditionals in the code), h be the number of assets, m be the number of states and m' be the number of functions. Then

- the cost of the inference of liquidity types is linear with respect to the size of the contract, *i.e.* $O(n)$;
- the length of κ-canonical traces starting in a state is less than $\kappa \times m'$; therefore the cardinality of $\mathbb{T}^{\kappa}_{\mathtt{Q}}$ is bounded by $\sum_{0 \leq i \leq \kappa \times m'} i!$. The cost of computing $\mathbb{T}^{\kappa}_{\mathtt{Q}}$, for every $\mathtt{Q}$, and the liquidity types of the elements therein are proportional to the number of κ-canonical traces, which are $N = m \times (\sum_{0 \leq i \leq \kappa \times m'} i!)$;
- The cost for verifying step 2 of the algorithm in Table 5 is $O(N \times N \times 2^h)$ because, for every κ-canonical trace and every subset of $\overline{\mathtt{h}}$, we must look for a κ-canonical trace satisfying either step 2.1 or step 2.2. The cost for verifying step 2 of the algorithm in Table 6 is $O(N \times N)$ because, for every κ-canonical trace, we must look for a κ-canonical trace satisfying either step 2.1 or step 2.2.

Therefore the overall cost of the algorithm in Table 5 is $O(n + N + N^2 \times 2^h)$ and the one in Table 6 is $O(n + N + N^2)$. Next, assuming that the number of asset h is a constant upper bound (while *Stipula* does not bound the assets, in the realistic examples we wrote, some of them are reported in [9,8,14], the assets are never more than 3) and assuming that the other values are in linear relation with m', then the computational cost of the algorithms in Table 5 and Table 6 is $O(N^2)$, *i.e.* exponential with respect to m'.

6. Related works

Liquidity properties have been put forward by Tsankov et al. in [18] as the property of a smart contract to always admit a trace where its balance is decreased (so, the funds stored within the contract do not remain frozen). Later, Bartoletti and Zunino in [3] discussed and extended this notion to a general setting – the Bitcoin language – that takes into account the strategy that a participant (which is possibly an adversary) follows to perform contract actions. More precisely, they observe that there are many possible flavors of liquidity, depending on which participants are assumed to be honest and on what are the strategies. In the taxonomy of [3,2], the notion of liquidity that we study in this work is the so-called *multiparty strategyless liquidity*, which assumes that all the contract's parties cooperate by actually calling the functions provided by the contract. We notice that, without cooperation, there is no guarantee that a party that has the permission to call a function will actually call it.

Both [18] and [3,2] adopt a model checking technique to verify properties of contracts. However, while [18] uses finite state models and the Uppaal model checker to verify the properties, [3,2] targets infinite state system and reduces them to finite state models that are consistent and complete with respect to liquidity. This mean that the technique of [3,2] is close to ours (we also target infinite state models and reduce to finite sets of abstract computations that over-approximate the real ones), even if we stick to a symbolic approach. Last, the above contributions and the ones we are aware of in the literature always address programs with one asset only (the contract balance). In this work we have understood that analyzing liquidity in programs with several different assets is way more complex than the case with a single asset.

A number of research projects are currently investigating the subject of resource-aware programming, as the prototype languages Obsidian [6] Nomos [11,4], Marlowe [16] and Scilla [17]. As discussed in the empirical study [6], programming with linear types, ownership and assets is difficult and the presence of strong type systems can be an effective advantage. In fact, the above languages provide type systems that guarantee that assets are not accidentally lost, even if none of them address liquidity. More precisely, Obsidian uses types to ensure that owning references to assets cannot be lost unless they are explicitly disowned by the programmer. Nomos uses a linear type system to prevent the duplication or deletion of

assets and amortized resource analysis to statically infer the resource cost of transactions. Marlowe [16], being a language for financial contracts, does not admit that money be locked forever in a contract. In particular, Marlowe's contracts have a finite lifetime and, at the end of the lifetime, any remaining money is returned to the participants. In other terms, all contracts are liquid by construction. In the extension of *Stipula* with events, the finite lifetime constraint can be explicitly programmed: a contract issues an event at the beginning so that at the timeout all the contract's assets are sent to the parties. Finally, Scilla is an intermediate-level language for safe smart contracts that is based on System F and targets a blockchain. It is unquestionable that a blockchain implementation of *Stipula* would bring in the advantages of a public and decentralized platform, such as traceability and the enforcement of contractual conditions. Being Scilla a minimalistic language with a formal semantics and a powerful type system, it seems an excellent candidate for implementing *Stipula*.

7. Conclusions

We have studied liquidity, a property of programs managing resources that pinpoints those programs not freezing any resource forever. In particular we have designed and demonstrated the correctness of two algorithms that verify two different liquidity properties.

We are currently prototyping the two algorithms. In case of liquidity, our prototype takes in input an integer value κ and verifies liquidity by sticking to types in $\mathbb{T}^{\kappa}_{\mathtt{Q}}$. This allows us to tune the precision of the analysis according to the contract to verify. We are also considering optimizations that improve both the precision of the algorithms and the performance. For example, the precision of the checks $[\![\Xi'(\mathtt{k})]\!] \neq \mathbb{0}$ and $[\![\Xi'(\mathtt{k})]\!] \neq [\![\Xi(\mathtt{k})]\!]$ may be improved by noticing that the algebra of liquidity expressions is a distributive lattice with min ($\mathbb{0}$) and max ($\mathbb{1}$). This algebra has a complete axiomatization that we may implement (for simplicity sake, in this paper we have only used min-max rules – see definition of $[\![\mathtt{e}]\!]$). Other optimizations we are studying allow us to reduce the number of canonical computations to verify (such as avoiding repetition of cycles that modify only one asset).

Another research objective addresses the liquidity analysis in languages featuring *conditional transitions* and *events*, such as the full *Stipula* [9]. These primitives introduce *internal nondeterminism*, which may undermine state reachability and, for this reason, they have been dropped in this paper. In particular, our analysis might synthesize a computation containing a function whose execution depends on values of fields that never hold. Therefore the computation will never be executed (it is a false positive) and must be discarded (and the contract might be not liquid). To overcome these problems, we will try to complement our analysis with an (off-the-shelf) constraint solver technique that guarantees the reachability of states of the computations synthesized by our algorithms.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Cosimo Laneve reports financial support was provided by the SERICS project (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union – NextGenerationEU – and by the H2020-MSCA-RISE project ID 778233 "Behavioural Application Program Interfaces (BEHAPI)".

Data availability

No data was used for the research described in the article.

Acknowledgements

I thank Silvia Crafa that co-authored the extended abstract for the discussions about liquidity. I also thank the FACS 2022 and the JLAMP referees for their careful reading and the many constructive suggestions on the submitted paper.

Appendix A. Progress and soundness of assets

Theorem 1. *(Progress). Let C be a closed* Stipula *contract with fields $\overline{\mathtt{x}}$, assets $\overline{\mathtt{h}}$, parties $\overline{\mathtt{A}}$ and* `@Q A:f(`$\overline{y}$`)[`$\overline{k}$`]{ S } ⇒ @Q'` $\in C$. *For every ℓ such that $\overline{\mathtt{x}}, \overline{\mathtt{h}}, \overline{\mathtt{A}} \subseteq dom(\ell)$, there is ℓ' such that* $C(\mathtt{Q}, \ell, \_) \xrightarrow{A.\mathtt{f}(\overline{u})[\overline{v}]} C(\mathtt{Q}', \ell', \_)$.

Proof. In the thesis we are assuming $\ell(\mathtt{A}) = A$. Because of the thesis and [Function] we have

$$C(\mathtt{Q}, \ell, \_) \xrightarrow{A.\mathtt{f}(\overline{u})[\overline{v}]} C(\mathtt{Q}, \ell[\overline{y} \mapsto \overline{u}, \overline{k} \mapsto \overline{v}], S \Rightarrow \mathtt{Q}')$$

therefore we are reduced to show the existence of

$$C(\mathtt{Q}, \ell[\overline{y} \mapsto \overline{u}, \overline{k} \mapsto \overline{v}], S \Rightarrow \mathtt{Q}') \xrightarrow{\mu_1} \cdots \xrightarrow{\mu_n} C(\mathtt{Q}', \ell', \_)$$

when $\overline{\mathtt{y}}, \overline{\mathtt{k}}, \overline{\mathtt{x}}, \overline{\mathtt{h}}, \overline{\mathtt{A}} \subseteq dom(\ell[\overline{y} \mapsto \overline{u}, \overline{k} \mapsto \overline{v}])$. Let $\ell'' = \ell[\overline{y} \mapsto \overline{u}, \overline{k} \mapsto \overline{v}]$; we reason by induction on S. The base case is $S = \_$, which is immediate by [State-Change]. The inductive cases are (*i*) $S = P\,S'$ and (*ii*) $S = \mathtt{if}\,(E)\,\{S'\}\,\mathtt{else}\,\{S''\}\,S'''$. We only

discuss (*i*), in particular when $P = E \to \mathtt{x}$ and $P = c \times \mathtt{h} \multimap \mathtt{A}$. When $P = E \to \mathtt{x}$, since $\mathtt{C}$ is closed, then $fv(E), \mathtt{x} \subseteq dom(\ell'')$. In order to apply [Field-Update] we need to verify the existence of v such that $[\![E]\!]_{\ell''} = v$. This follows by the fact that every operation in E is total and returns a real number and because names in E are bound in ℓ''. Hence

$$\mathtt{C}(\mathtt{Q}, \ell'', E \to \mathtt{x}\, S \Rightarrow \mathtt{Q}') \longrightarrow \mathtt{C}(\mathtt{Q}, \ell''[\mathtt{x} \mapsto v]\,,\ S \Rightarrow \mathtt{Q}')$$

by [Field-Update]. The thesis follows by induction since $dom(\ell'') = dom(\ell''[\mathtt{x} \mapsto v])$. When $P = c \times \mathtt{h} \multimap \mathtt{A}$, we notice that $\mathtt{h}, \mathtt{A} \in dom(\ell'')$ because $\mathtt{C}$ is closed and $\mathtt{h}$ is either an asset or an asset parameter. Additionally, since c is a constant, there is v such that $[\![c \times \mathtt{h}]\!]_{\ell''} = v$ and v' such that $[\![\mathtt{h} - v]\!]_{\ell''} = v'$. Therefore

$$\mathtt{C}(\mathtt{Q}, \ell'', c \times \mathtt{h} \multimap \mathtt{A}\, S \Rightarrow \mathtt{Q}') \xrightarrow{v' \multimap A} \mathtt{C}(\mathtt{Q}, \ell''[\mathtt{h} \mapsto v']\,,\ S \Rightarrow \mathtt{Q}')$$

by [Asset-Send]. The thesis follows by induction since $dom(\ell'') = dom(\ell''[\mathtt{x} \mapsto v])$.

Theorem 2. *(Soundness of assets). Let C be a* Stipula *contract and $@\mathtt{Q}\ \ \mathtt{A} : \mathtt{f}\ (\overline{\mathtt{y}})\ [\overline{\mathtt{k}}]\ \{\ S\ \} \Rightarrow @\mathtt{Q}' \in C$. If ℓ is sound and $\overline{v} \geq \overline{0}$ and $C(\mathtt{Q}, \ell, \_) \overset{A.\mathtt{f}(\overline{u})[\overline{v}]}{\Longrightarrow} C(\mathtt{Q}', \ell', \_)$ then ℓ' is sound.*

Proof. The computation in the hypothesis is actually

$$\mathtt{C}(\mathtt{Q}, \ell, \_) \xrightarrow{A.\mathtt{f}(\overline{u})[\overline{v}]} \mathtt{C}(\mathtt{Q}, \ell[\overline{\mathtt{y}} \mapsto \overline{u}, \overline{\mathtt{k}} \mapsto \overline{v}], S \Rightarrow \mathtt{Q}') \Longrightarrow \mathtt{C}(\mathtt{Q}', \ell', \_)$$

and, by hypothesis, $\ell[\overline{\mathtt{y}} \mapsto \overline{u}, \overline{\mathtt{k}} \mapsto \overline{v}]$ is sound. We demonstrate that every memory ℓ'' in the computation $\mathtt{C}(\mathtt{Q}, \ell[\overline{\mathtt{y}} \mapsto \overline{u}, \overline{\mathtt{k}} \mapsto \overline{v}], S \Rightarrow \mathtt{Q}') \Longrightarrow \mathtt{C}(\mathtt{Q}', \ell', \_)$ is sound. We do a case analysis on the transitions. In particular, those that modify assets are the instances of [Asset-Send] and [Asset-Update]. Let

$$\mathtt{C}(\mathtt{Q}, \ell_1, c \times \mathtt{h} \multimap \mathtt{A}\, S' \Rightarrow \mathtt{Q}') \xrightarrow{v \multimap A} \mathtt{C}(\mathtt{Q}', \ell_2, S' \Rightarrow \mathtt{Q}')$$

one of the transitions and assume that ℓ_1 is sound. Then, by [Asset-Send], $\mathtt{h} \in dom(\ell_1)$ (otherwise $[\![c \times \mathtt{h}]\!]_{\ell_1}$ should be undefined) and $\mathtt{A} \in dom(\ell_1)$ by the hypothesis of the rule. Since $0 \leq c \leq 1$, we obtain that $0 \leq [\![c \times \mathtt{h}]\!]_{\ell_1} \leq [\![\mathtt{h}]\!]_{\ell_1}$. Let v be the corresponding value; we derive $[\![\mathtt{h} - v]\!]_{\ell_1} = v' \geq 0$. Therefore $\ell_2 = \ell_1[\mathtt{h} \mapsto v']$ is sound as well. The arguments demonstrating that ℓ_2 is sound provided that ℓ_1 is and

$$\mathtt{C}(\mathtt{Q}, \ell_1, c \times \mathtt{h} \multimap \mathtt{h}'\, S' \Rightarrow \mathtt{Q}') \longrightarrow \mathtt{C}(\mathtt{Q}', \ell_2, S' \Rightarrow \mathtt{Q}')$$

are similar to the foregoing ones. Notice that, in this case $[\![c \times \mathtt{h}]\!]_{\ell_1} = v \geq 0$ and $[\![\mathtt{h} - v]\!]_{\ell_1} \geq 0$ and $[\![\mathtt{h}' + v]\!]_{\ell_1} \geq 0$.

Appendix B. The theory of liquidity: technical material

Lemma 1. *The following properties hold true:*

(*Weakening*) *Let $\Xi \vdash_{\mathtt{X}} S : \Xi'$ and Ξ_w be such that $\Xi_w|_{dom(\Xi)} = \Xi$. Then there exists Ξ'_w such that $\Xi_w \vdash_{\mathtt{X}} S : \Xi'_w$ and $\Xi'_w|_{dom(\Xi')} = \Xi'$.*

(*Substitution*) *Let σ be a substitution mapping liquidity names to liquidity expressions. If $\Xi \vdash_{\mathtt{X}} S : \Xi'$ then $\sigma(\Xi) \vdash_{\mathtt{X}} S : \sigma(\Xi')$.*

(*Monotonicity*) *Let Ξ, Ξ'' be ground environments such that $\Xi \leq \Xi''$. If $\Xi \vdash_{\mathtt{X}} S : \Xi'$ then there is Ξ''' such that $\Xi'' \vdash_{\mathtt{X}} S : \Xi'''$ and $\Xi' \leq \Xi'''$.*

Proof. The proofs of Weakening and Substitution are standard and therefore omitted; we discuss the Monotonicity. The proof is by induction on the structure of S. The basic case, *i.e.* $S = \_$, is immediate. The inductive cases are $P\ S$ and $\mathtt{if}\,(E)\,\{S_t\}\,\mathtt{else}\,\{S_e\}\,S$, assuming the Monotonicity holds on S, S_t and S_e. Regarding $P\ S$, the interesting cases are when P is (*i*) $c \times \mathtt{h} \multimap \mathtt{h}'$, or (*ii*) $c \times \mathtt{h} \multimap \mathtt{A}$, or (*iii*) $\mathtt{h} \multimap \mathtt{h}'$.

In case (*i*), monotonicity must be proved for $\Xi \vdash_{\mathtt{X}} c \times \mathtt{h} \to \mathtt{h}'\ \ S : \Xi'$, given $\Xi \leq \Xi''$. By [L-seq], we have $\Xi \vdash_{\mathtt{X}} c \times \mathtt{h} \to \mathtt{h}' : \Xi_1$ and $\Xi_1 \vdash_{\mathtt{X}} S : \Xi'$. By [L-expaupd], $\Xi_1 = \Xi[\mathtt{h}' \mapsto \mathbb{e}]$, where $\mathbb{e} = \Xi(\mathtt{h}) \sqcup \Xi(\mathtt{h}')$. Taking an environment Ξ'' such that $\Xi \leq \Xi''$ and applying [L-expaupd], we obtain $\Xi'' \vdash_{\mathtt{X}} c \times \mathtt{h} \to \mathtt{h}' : \Xi'_1$ such that $\Xi'_1 = \Xi''[\mathtt{h}' \mapsto \mathbb{e}']$, where $\mathbb{e}' = \Xi''(\mathtt{h}) \sqcup \Xi''(\mathtt{h}')$. Therefore, by monotonicity of $\sqcup$, $\Xi_1 \leq \Xi'_1$. The thesis follows by inductive hypotheses on S. Cases (*ii*) and (*iii*) are similar.

When the statement is $\mathtt{if}\,(E)\,\{S_t\}\,\mathtt{else}\,\{S_e\}\,S$ we use [L-cond]. To this aim, we observe that $fn(E) \subseteq \mathtt{X} \cup dom(\Xi'')$ follows by $fn(E) \subseteq \mathtt{X} \cup dom(\Xi)$ and $\Xi \leq \Xi''$. In this case the thesis is a straightforward application of the inductive hypothesis.

Theorem 3. *(Correctness of liquidity function types). Let $\vdash C : \mathcal{L}$ and $\overline{\mathtt{h}}$ be the assets of C and $@\mathtt{Q}\,\mathtt{A} : \mathtt{f}\ (\overline{\mathtt{y}})\ [\overline{\mathtt{k}}]\ \ \{\ S\ \} \Rightarrow @\mathtt{Q}' \in \mathtt{C}$ and $\mathtt{Q}\,\mathtt{A}.\mathtt{f}\ \mathtt{Q}' : \Xi \to \Xi'$ in $\mathcal{L}$. If $\overline{\mathtt{h}} \subseteq dom(\ell)$ and $C(\mathtt{Q}\,,\ \ell\,,\ \_) \overset{A.\mathtt{f}(\overline{u})[\overline{v}]}{\Longrightarrow} C(\mathtt{Q}'\,,\ \ell'\,,\ \_)$ then there are X and Ξ'' such that:*

1. $\mathbb{E}(\ell[\overline{\mathtt{y}} \mapsto \overline{u}, \overline{\mathtt{k}} \mapsto \overline{v}]) \vdash_{\mathtt{X} \cup \overline{\mathtt{y}}} S : \Xi''$;

2. $\mathbb{E}(\ell[\overline{y} \mapsto \overline{u}, \overline{k} \mapsto \overline{v}])|_{dom(\Xi)} \leq \sigma(\Xi)$ *and* $\Xi''|_{dom(\Xi)} \leq \sigma(\Xi')$, *for a ground substitution* σ;
3. $\mathbb{E}(\ell')|_{dom(\Xi)} \leq \Xi''$.

Proof. Regarding item 1, by $\mathtt{Q}\ \mathtt{A.f}\ \mathtt{Q}' : \Xi \to \Xi' \in \mathcal{L}$ and [L-FUNCTION] and rule [L-CONTRACT], we have that $\Xi_0[\overline{k} \mapsto \overline{\xi'}] \vdash_{X \cup \overline{y}} S : \Xi_1$, where $\Xi_0 = [\overline{h} \mapsto \overline{\xi}]$ and $\Xi = \Xi_0[\overline{k} \mapsto \overline{\mathbb{1}}]$ and $\Xi' = \Xi_1\{^{\overline{\mathbb{1}}}/_{\overline{\xi'}}\}$ ($\overline{\xi}$ and $\overline{\xi'}$ are tuples of fresh liquidity names). We first notice that $dom(\Xi) = dom(\Xi_0[\overline{k} \mapsto \overline{\xi'}]) = \overline{h} \cup \overline{k} \subseteq dom(\mathbb{E}(\ell[\overline{y} \mapsto \overline{u}, \overline{k} \mapsto \overline{v}]))$ because of the hypothesis $\overline{h} \subseteq dom(\ell)$. The containment may be strict because $dom(\mathbb{E}(\ell[\overline{y} \mapsto \overline{u}, \overline{k} \mapsto \overline{v}]))$ may have asset names that do not occur in $dom(\Xi_0[\overline{k} \mapsto \overline{\xi'}])$ (these names are the formal parameters of previous invocations, which have not been garbage-collected by the semantics). By definition of $\Xi_0[\overline{k} \mapsto \overline{\xi'}]$, there exists a ground substitution σ such that

$$\mathbb{E}(\ell[\overline{y} \mapsto \overline{u}, \overline{k} \mapsto \overline{v}])|_{dom(\Xi)} = \sigma(\Xi_0[\overline{k} \mapsto \overline{\xi'}])\ .$$

Therefore, by Lemma 1 (Substitution) applied to $\Xi_0[\overline{k} \mapsto \overline{\xi'}] \vdash_{X \cup \overline{y}} S : \Xi_1$, we have

$$\mathbb{E}(\ell[\overline{y} \mapsto \overline{u}, \overline{k} \mapsto \overline{v}])|_{dom(\Xi)} \vdash_{X \cup \overline{y}} S : \sigma(\Xi_1) \tag{B.1}$$

Item 1 follows by applying Lemma 1 (Weakening) to (B.1). The environment Ξ'' of the thesis is $\sigma(\Xi_1)$ after the weakening, *i.e.* $\Xi''|_{dom(\Xi)} = \sigma(\Xi_1)$.

We demonstrate 2. Observe that $\Xi = \Xi_0[\overline{k} \mapsto \overline{\xi'}]\{^{\overline{\mathbb{1}}}/_{\overline{\xi'}}\}$. Therefore, by $\Xi_0[\overline{k} \mapsto \overline{\xi'}] \vdash_{X \cup \overline{y}} S : \Xi_1$ and Lemma 1 (Substitution), we obtain

$$\sigma'(\Xi_0[\overline{k} \mapsto \overline{\xi'}]) \vdash_{X \cup \overline{y}} S : \sigma'(\Xi_1) \tag{B.2}$$

where $\sigma' = \{^{\overline{\mathbb{1}}}/_{\overline{\xi'}}\}$. Next observe that $\sigma'(\Xi_0[\overline{k} \mapsto \overline{\xi'}]) = \Xi$ and $\sigma'(\Xi_1) = \Xi$; hence we rewrite (B.2) as

$$\Xi \vdash_{X \cup \overline{y}} S : \Xi' \tag{B.3}$$

Finally, notice that, using the substitution σ of (B.1) we have

$$\begin{array}{rcl} \mathbb{E}(\ell[\overline{y} \mapsto \overline{u}, \overline{k} \mapsto \overline{v}])|_{dom(\Xi)} & = & \sigma(\Xi_0[\overline{k} \mapsto \overline{\xi'}]) \\ & \leq & \sigma|_{\overline{h}}(\Xi_0[\overline{k} \mapsto \overline{\mathbb{1}}]) \\ & = & \sigma|_{\overline{h}}(\Xi) \end{array}$$

where $\sigma|_{\overline{h}}$ is the substitution σ that is only defined on $\overline{\xi}$; the "$\leq$" follows by the fact that, in $\mathbb{E}(\ell)|_{dom(\Xi)}$, some asset parameter in $\overline{k}$ might be $\mathbb{0}$, while these parameters are all $\mathbb{1}$ in Ξ. We conclude by Monotonicity applied to (B.3) obtaining $\Xi''|_{dom(\Xi)} \leq \sigma|_{\overline{h}}(\Xi')$.

To demonstrate 3, we proceed by induction on the length of the computation $\mathtt{C}(\mathtt{Q}, \ell[\overline{y} \mapsto \overline{u}, \overline{k} \mapsto \overline{v}], S \Rightarrow @\mathtt{Q}') \Longrightarrow \mathtt{C}(\mathtt{Q}', \ell', \_)$, assuming that the properties of items 1 and 2 hold. The base case is immediate. If the property holds for computations of length n, we demonstrate the case of computations of length $n+1$ by reasoning on the first transition and applying inductive hypotheses to the continuation. We discuss the case when the first transition is an instance of [ASSET-UPDATE] with $c \neq 1$. Let $\ell'' = \ell[\overline{y} \mapsto \overline{u}, \overline{k} \mapsto \overline{v}]$ and

$$\begin{array}{rcl} \mathtt{C}(\mathtt{Q}, \ell'', c \times \mathtt{h} \multimap \mathtt{h}'\ S \Rightarrow @\mathtt{Q}') & \longrightarrow & \mathtt{C}(\mathtt{Q}, \ell''[\mathtt{h} \mapsto v', \mathtt{h}' \mapsto v''], S \Rightarrow @\mathtt{Q}') \\ & \Longrightarrow & \mathtt{C}(\mathtt{Q}, \ell', \_ \Rightarrow @\mathtt{Q}') \end{array}$$

where $[\![c \times \mathtt{h}]\!]_{\ell''} = v$ and $[\![\mathtt{h} - v]\!]_{\ell''} = v'$ and $[\![\mathtt{h}' + v]\!]_{\ell''} = v''$. By the property of item 1, $\mathbb{E}(\ell'') \vdash_{X \cup \overline{y}} c \times \mathtt{h} \multimap \mathtt{h}'\ S : \Xi''$, which is obtained by [L-EXPAUPD] and [L-SEQ] with premises

$$\mathbb{E}(\ell'') \vdash_{X \cup \overline{y}} c \times \mathtt{h} \multimap \mathtt{h}' : \mathbb{E}(\ell'')[\mathtt{h}' \mapsto \mathbb{e}] \qquad (\mathbb{e} = \mathbb{E}(\ell'')(\mathtt{h}) \sqcup \mathbb{E}(\ell'')(\mathtt{h}')) \tag{B.4}$$

$$\mathbb{E}(\ell'')[\mathtt{h}' \mapsto \mathbb{e}] \vdash_{X \cup \overline{y}} S : \Xi'' \tag{B.5}$$

Additionally, by inductive hypotheses, we also have

$$\mathbb{E}(\ell''[\mathtt{h} \mapsto v', \mathtt{h}' \mapsto v'']) \vdash_{X \cup \overline{y}} S : \Xi''' \tag{B.6}$$

$$\mathbb{E}(\ell')|_{dom(\Xi)} \leq \Xi''' \tag{B.7}$$

We then observe that, by definition of v', v'' and $\mathbb{e}$: $\mathbb{E}(\ell''[\mathtt{h} \mapsto v', \mathtt{h}' \mapsto v'']) \leq \mathbb{E}(\ell'')[\mathtt{h}' \mapsto \mathbb{e}]$. Therefore, by Lemma 1 (Monotonicity) and (B.5) and (B.6), we derive $\Xi''' \leq \Xi''$. The conclusion follows by this inequality, (B.7) and transitivity of $\leq$.

Theorem 4. *(Correctness of liquidity types of abstract computations). Let* $\vdash C : \mathcal{L}$ *and* $\left(\mathtt{C}(\mathtt{Q}_i, \ell_i, \_) \overset{A_i : \mathtt{f}_i(\overline{u_i})[\overline{v_i}]}{\Longrightarrow} \mathtt{C}(\mathtt{Q}_{i+1}, \ell_{i+1}, \_)\right)^{i \in 1..n}$ *with* $dom(\ell_1)$ *containing the assets* $\overline{h}$ *of* C. *Let also* $\varphi = \{\ \mathtt{Q}_i\ \mathtt{A}_i.\mathtt{f}_i\ \mathtt{Q}_{i+1}\ \}^{i \in 1..n}$ *have liquidity type* $\mathtt{L}_\varphi = \Xi \to \Xi'$.

Then there is a substitution σ *such that* $\mathbb{E}(\ell_1)|_{\overline{h}} \leq \sigma(\Xi)$ *and* $\mathbb{E}(\ell_{n+1})|_{\overline{h}} \leq \sigma(\Xi')$.

Proof. Let $\varphi = \{\mathtt{Q}_i\, A_i.\mathtt{f}_i\, \mathtt{Q}_{i+1}\}^{i\in 1..n}$ and, for every i, $\mathtt{Q}_i\, A_i.\mathtt{f}_i\, \mathtt{Q}_{i+1} : \Xi_i \to \Xi'_i \in \mathcal{L}$. Let also $\mathtt{L}_\varphi = \Xi_1^{(b)}|_{\overline{\mathtt{h}}} \to \Xi_n^{(e)}|_{\overline{\mathtt{h}}}$ such that

$$\Xi_1^{(b)} = \Xi_1 \qquad \Xi_{i+1}^{(b)} = \Xi_{i+1}\{^{\Xi_i^{(e)}(\overline{\mathtt{h}})}/_{\overline{\xi}}\} \qquad \Xi_i^{(e)} = \Xi'_i\{^{\Xi_i^{(b)}(\overline{\mathtt{h}})}/_{\overline{\xi}}\}\,.$$

By definition we have

$$\Xi_i|_{\overline{\mathtt{h}}} = [\overline{\mathtt{h}} \mapsto \overline{\xi}] \qquad \text{and} \qquad \Xi_{i+1}^{(e)} = \Xi'_{i+1}\{^{\Xi_{i+1}^{(b)}(\overline{\mathtt{h}})}/_{\overline{\xi}}\} = \Xi'_{i+1}\{^{\Xi_i^{(e)}(\overline{\mathtt{h}})}/_{\overline{\xi}}\}$$

because $\Xi_{i+1}^{(b)}(\overline{\mathtt{h}}) = \Xi_i^{(e)}(\overline{\mathtt{h}})$. Next, by $\mathtt{Q}_i\, A_i.\mathtt{f}_i\, \mathtt{Q}_{i+1} : \Xi_i \to \Xi'_i \in \mathcal{L}$ and [L-FUNCTION], we derive

$$\Xi_i[\overline{\mathtt{k}_i} \mapsto \overline{\xi_i}] \vdash_{\mathtt{X}\cup\overline{\gamma_i}} S_i : \Xi''_i \qquad \text{where} \quad \Xi'_i = \Xi''_i\{^{\overline{\mathbb{1}}}/_{\overline{\xi_i}}\}$$

for every $i \in 1..n$. Therefore, Lemma 1 (Substitution) ($\Xi_0^{(e)}(\overline{\mathtt{h}}) = \overline{\xi}$):

$$\Xi_i[\overline{\mathtt{k}_i} \mapsto \overline{\xi_i}]\{^{\Xi_{i-1}^{(e)}(\overline{\mathtt{h}})}/_{\overline{\xi}}\} \vdash_{\mathtt{X}\cup\overline{\gamma_i}} S_i : \Xi''_i\{^{\Xi_{i-1}^{(e)}(\overline{\mathtt{h}})}/_{\overline{\xi}}\}$$

and applying the substitution $\{^{\overline{\mathbb{1}}}/_{\overline{\xi_i}}\}$ we also have

$$\Xi_i[\overline{\mathtt{k}_i} \mapsto \overline{\xi_i}]\{^{\Xi_{i-1}^{(e)}(\overline{\mathtt{h}}),\overline{\mathbb{1}}}/_{\overline{\xi},\overline{\xi_i}}\} \vdash_{\mathtt{X}\cup\overline{\gamma_i}} S_i : \Xi''_i\{^{\Xi_{i-1}^{(e)}(\overline{\mathtt{h}}),\overline{\mathbb{1}}}/_{\overline{\xi},\overline{\xi_i}}\} \tag{B.8}$$

Notice that (B.8) is exactly $\Xi_i^{(b)} \vdash_{\mathtt{X}\cup\overline{\gamma_i}} S_i : \Xi_i^{(e)}$; therefore, letting $\overline{\mathtt{h}'_i} = dom(\Xi_i^{(b)})$, if there is a ground substitution σ_i such that $\mathbb{E}(\ell'_i)|_{\overline{\mathtt{h}'_i}} \le \sigma_i(\Xi_i^{(b)})$ then, by Theorem 3.2, $\mathbb{E}(\ell_{i+1})|_{\overline{\mathtt{h}'_i}} \le \sigma_i(\Xi_i^{(e)})$. In the following we demonstrate that the σ_i do exist and are all equal for every i.

Let $\overline{\mathtt{h}_1} = dom(\Xi_1^{(b)})$. By definition of $\Xi_1^{(b)}$, there is σ such that

$$\mathbb{E}(\ell_1)|_{\overline{\mathtt{h}_1}} \le \sigma(\Xi_1^{(b)}) \tag{B.9}$$

From this we derive:

$$\mathbb{E}(\ell_2)|_{\overline{\mathtt{h}_1}} \le \sigma(\Xi_1^{(e)}) \qquad \text{(by Theorem 3.2)} \tag{B.10}$$

$$\text{hence} \quad \mathbb{E}(\ell_1)|_{\overline{\mathtt{h}}} \le \sigma(\Xi_1^{(b)})|_{\overline{\mathtt{h}}} \qquad \text{(by (B.9) and } \overline{\mathtt{h}} \subseteq \overline{\mathtt{h}_1}) \tag{B.11}$$

$$\text{hence} \quad \mathbb{E}(\ell_2)|_{\overline{\mathtt{h}}} \le \sigma(\Xi_1^{(e)})|_{\overline{\mathtt{h}}} \qquad \text{(by (B.10) and } \overline{\mathtt{h}} \subseteq \overline{\mathtt{h}_1}) \tag{B.12}$$

By definition, $\ell'_2 = \ell_2[\overline{\gamma_2} \mapsto \overline{u_2}, \overline{\mathtt{k}_2} \mapsto \overline{\gamma_2}]$. Let $dom(\Xi_2^{(b)}) = \overline{\mathtt{h}_2} = \overline{\mathtt{h}} \cup \overline{\mathtt{k}_2}$. Then $\mathbb{E}(\ell'_2)|_{\overline{\mathtt{h}_2}} \le \mathbb{E}(\ell_2)|_{\overline{\mathtt{h}}}[\overline{\mathtt{k}_2} \mapsto \overline{\mathbb{1}}] \le \sigma(\Xi_1^{(e)})|_{\overline{\mathtt{h}}}[\overline{\mathtt{k}_2} \mapsto \overline{\mathbb{1}}]$ (because of (B.12)). We conclude by observing that $\Xi_2^{(b)} = \Xi_1^{(e)}|_{\overline{\mathtt{h}}}[\overline{\mathtt{k}_2} \mapsto \overline{\mathbb{1}}]$ and that $\sigma(\Xi_2^{(b)}) = \sigma(\Xi_1^{(e)}|_{\overline{\mathtt{h}}}[\overline{\mathtt{k}_2} \mapsto \overline{\mathbb{1}}]) = \sigma(\Xi_1^{(e)})|_{\overline{\mathtt{h}}}[\overline{\mathtt{k}_2} \mapsto \overline{\mathbb{1}}]$.

The theorem follows by repeating the arguments on every function in the abstract computation.

Corollary 1. *Let* $\vdash C : \mathcal{L}$ *and* $\big(C(\mathtt{Q}_i\,,\, \ell_i\,,\, \_) \overset{A_i:\mathtt{f}_i(\overline{u_i})[\overline{v_i}]}{\Longrightarrow} C(\mathtt{Q}_{i+1},\, \ell_{i+1}\,,\, \_)\big)^{i\in 1..n}$ *with* $dom(\ell_1)$ *containing the assets* $\overline{\mathtt{h}}$ *of* C. *Let also* $\varphi = \{\, \mathtt{Q}_i\, A_i.\mathtt{f}_i\, \mathtt{Q}_{i+1}\,\}^{i\in 1..n}$ *have liquidity type* $\mathtt{L}_\varphi = \Xi \to \Xi'$.

If $\mathtt{k} \in \overline{\mathtt{h}}$ *and* $[\![\Xi(\mathtt{k})]\!] = [\![\Xi'(\mathtt{k})]\!]$ *then* $\mathbb{E}(\ell_{n+1}) \le \mathbb{E}(\ell_1)$. *In particular, if* $\mathbb{E}(\ell_1) = \mathbb{0}$ *then* $\mathbb{E}(\ell_{n+1}) = \mathbb{0}$.

Proposition 2. *Let* $\Xi = [\overline{\mathtt{h}} \mapsto \overline{\xi}]$, *where* $\overline{\xi}$ *is a tuple of pairwise different symbolic names,* $\Xi \vdash_{\mathtt{X}} S : \Xi'$ *and* $[\![\Xi(\mathtt{h})]\!] \neq [\![\Xi'(\mathtt{h})]\!]$. *Then* S *either contains* $\mathtt{h} \multimap \mathtt{h}'$ *or* $\mathtt{h}' \multimap \mathtt{h}$ *or* $c \times \mathtt{h}' \multimap \mathtt{h}$ *or* $\mathtt{h} \multimap \mathtt{A}$ *(we say that* $\mathtt{h}$ *has been updated in* S*).*

Similarly, if $\vdash C : \mathcal{L}$ *and* $\mathtt{L}_\varphi = \Xi \to \Xi'$ *for an abstract computation* φ *and* $[\![\Xi(\mathtt{h})]\!] \neq [\![\Xi'(\mathtt{h})]\!]$ *then* $\mathtt{h}$ *has been updated by (the body of) at least one of the functions in* φ.

Proof. We first notice that $[\![\Xi(\mathtt{h})]\!] \neq [\![\Xi'(\mathtt{h})]\!]$ implies $\Xi(\mathtt{h}) \neq \Xi'(\mathtt{h})$ because $[\![\cdot]\!]$ is applying a number of axioms of $\sqcup$ and $\sqcap$. Therefore we demonstrate the proposition when $\Xi(\mathtt{h}) \neq \Xi'(\mathtt{h})$ and we restrict to the case of statements; the proof for abstract computations is similar. By induction on the structure of S. The basic case $S = \_$ is vacuous. The inductive case is $S = P_1\,, \cdots,\, P_n\,,\, P_{n+1}$ where the P_i (with an abuse of notation) are either prefixes or conditionals. By rules [L-SEQ] and [L-COND], there is Ξ'' such that

$$\Xi \vdash_{\mathtt{X}} P_1\,, \cdots,\, P_n : \Xi'' \qquad \text{and} \qquad \Xi'' \vdash_{\mathtt{X}} P_{n+1} : \Xi'$$

and, by inductive hypotheses, if $\Xi(\mathtt{h}) \neq \Xi''(\mathtt{h})$ then $\mathtt{h}$ has been updated by $P_1\,, \cdots,\, P_n$. Therefore, if $\Xi(\mathtt{h}) \neq \Xi'(\mathtt{h})$ then S updates $\mathtt{h}$. The interesting case is when $\Xi(\mathtt{h}) = \Xi''(\mathtt{h})$ and $\Xi''(\mathtt{h}) \neq \Xi'(\mathtt{h})$. By cases on P_{n+1}. If $P_{n+1} = \mathtt{h}' \multimap \mathtt{h}''$ then, since $[\![\Xi''(\mathtt{h})]\!] \neq [\![\Xi'(\mathtt{h})]\!]$, either $\mathtt{h}'$ or $\mathtt{h}''$ must be $\mathtt{h}$ because, by [L-AUPDATE], $\Xi'' = \Xi'[\mathtt{h}' \mapsto \mathbb{0}, \mathtt{h}'' \mapsto \Xi(\mathtt{h}') \sqcup \Xi(\mathtt{h}'')]$. Hence S updates $\mathtt{h}$. The cases $P_{n+1} = c \times \mathtt{h}' \multimap \mathtt{h}''$ and $P_{n+1} = \mathtt{h}' \multimap \mathtt{A}$ are similar. The case when $P_{n+1} = \mathtt{if}\,(E)\,\{\,S'\,\}\,\mathtt{else}\,\{\,S''\,\}$ follows by inductive hypotheses on S' and S''.

Appendix C. Correctness of the algorithms

The following Proposition is about the correctness of k-separate liquidity+ and of liquidity+.

Proposition 4. *If the algorithm of Table 4 returns that a* Stipula *contract is k-separate liquid+ (respectively, liquid+) then it is k-separate liquid+ (respectively, liquid+). Additionally, the algorithms always terminate.*

Proof. The set $\mathbb{Q}_{\mathtt{Q}}$ is a closure that contains functions $\mathtt{Q}'\,\mathtt{A.f}\,\mathtt{Q}''$ of states $\mathtt{Q}'$ reachable from $\mathtt{Q}$. This set is finite because the functions are finitely many. Next, take a function with liquidity type $\Xi \to \Xi'$ and $[\![\Xi'(\mathtt{k})]\!] \neq \mathbb{0}$ and $[\![\Xi'(\mathtt{k})]\!] \neq [\![\Xi(\mathtt{k})]\!]$ (hypothesis (*a*) of step 2) then, by Proposition 2, the function updates the asset k in a way that may be problematic for κ-separate liquidity (because $[\![\Xi'(\mathtt{k})]\!] \neq \mathbb{0}$). In this case, the algorithm must find a function in the continuation that empties k.

If there is a function in $\mathbb{Q}_{\mathtt{Q}''}$, namely $\mathtt{Q1\,B.g\,Q2}$ with a liquidity type $\Xi_1 \to \Xi_2$ such that $[\![\Xi_2(\mathtt{k})]\!] = \mathbb{0}$ (step 3) then, by Theorem 3, $\mathtt{Q1\,B.g\,Q2}$ empties k and, by Theorem 1 it is possible to synthesize a (concrete) computation starting at $\mathtt{Q}'$, with initial transition the invocation of A.f and B.g the last function invocation (therefore ending at Q2). By Theorem 4, this computation empties k. If there is no function then the algorithm terminates without granting κ-separate liquidity.

We reiterate the process on every possible function satisfying step 2(a) and every time we store our result in $\mathcal{Z}$, in oder to avoid repetitions. This guarantees termination because $\mathcal{Z}$ may contain a finite number of pairs $(\mathtt{Q}, \mathtt{k})$ and every $\mathbb{Q}_{\mathtt{Q}}$ is finite.

The proof for liquidity+ is similar.

The correctness of the costly algorithm for liquidity follows.

Proposition 5. *Let $\vdash C : \mathcal{L}$. If the algorithm of Table 5 returns that C is liquid then it is liquid. Additionally, the algorithm always terminates.*

Proof. The set $\mathbb{T}^{\kappa}_{\mathtt{Q}'}$ contains all the κ-canonical abstract computations starting at $\mathtt{Q}'$. This set is finite because the functions are finitely many (we observe that there are infinite concrete computations corresponding to an abstract computation). Additionally, the number of such sets is finite because the states $\mathtt{Q}'$ (reachable from $\mathtt{Q}$) are finite. So we take $\mathtt{Q}' \overset{\varphi}{\curvearrowright} \mathtt{Q}'' : \Xi \to \Xi' \in \mathbb{T}^{\kappa}_{\mathtt{Q}'}$ such that there is a $\mathtt{k}'$ with (*a*) $[\![\Xi'(\mathtt{k}')]\!] \neq \mathbb{0}$ and $[\![\Xi'(\mathtt{k}')]\!] \neq [\![\Xi(\mathtt{k}')]\!]$ and (*b*) $(\mathtt{Q}'', \mathtt{k}') \notin \mathcal{Z}$ (for simplicity we are assuming that there is a unique $\mathtt{k}'$ satisfying (*a*) and (*b*)).

Then, by Proposition 2, the computation updates the asset $\mathtt{k}'$ in a way that may be problematic for liquidity (because $[\![\Xi'(\mathtt{k}')]\!] \neq \mathbb{0}$). In this case, the algorithm must find a continuation that is suitable for liquidity.

Notice that all the assets in $\mathtt{k} \in \mathtt{h} \setminus \mathtt{k}'$ are such that either $[\![\Xi'(\mathtt{k})]\!] = \mathbb{0}$ or $[\![\Xi'(\mathtt{k})]\!] = [\![\Xi(\mathtt{k})]\!]$.

Then, according to step 2.2, the algorithm looks for a continuation $\mathtt{Q}'' \overset{\varphi'}{\curvearrowright} \mathtt{Q}''' : \Xi'' \to \Xi''' \in \mathbb{T}^{\kappa}_{\mathtt{Q}''}$ such that $[\![\Xi'''(\mathtt{k}')]\!] = \mathbb{0}$ and, for every $\mathtt{k} \in \overline{\mathtt{h}} \setminus \overline{\mathtt{k}'}$, either $[\![\Xi'''(\mathtt{k})]\!] = \mathbb{0}$ or $[\![\Xi'''(\mathtt{k})]\!] = [\![\Xi''(\mathtt{k})]\!]$. Now consider the composition $\varphi \cdot \varphi'$; by definition its liquidity type is $\Xi \to \Xi^{\sharp}$ such that $\Xi^{\sharp}(\mathtt{h}) = [\![\Xi'''(\mathtt{h})]\!]\{^{\Xi'(\overline{\mathtt{h}})}/_{\overline{\xi}}\}$. Additionally, according to the hypotheses on φ and φ', for every h, either $\Xi^{\sharp}(\mathtt{h}) = \mathbb{0}$ or $\Xi^{\sharp}(\mathtt{h}) = \Xi(\mathtt{h})$. Now, by Theorem 1, it is possible to synthesize a computation corresponding to $\varphi \cdot \varphi'$. Let ℓ and ℓ' be the memories of the initial and final configurations. Since in the initial configuration all the assets are 0 then, by Theorem 4, the assets are also 0 in ℓ'. This guarantees liquidity for the computation φ.

The algorithm accumulates in $\mathcal{Z}$ all the proofs that have been done in order to avoid repetitions. It also guarantees termination because $\mathcal{Z}$ may contain a finite number of pairs $(\mathtt{Q}, \overline{\mathtt{k}})$.

References

[1] Elvira Albert, Frank S. de Boer, Reiner Hähnle, Einar Broch Johnsen, Cosimo Laneve, Engineering virtualized services, in: NordiCloud '13, in: ACM International Conference Proceeding Series, vol. 826, ACM, 2013, pp. 59–63.

[2] Massimo Bartoletti, Stefano Lande, Maurizio Murgia, Roberto Zunino, Verifying liquidity of recursive bitcoin contracts, Log. Methods Comput. Sci. 18 (1) (2022).

[3] Massimo Bartoletti, Roberto Zunino, Verifying liquidity of bitcoin contracts, in: Principles of Security and Trust, Springer International Publishing, 2019, pp. 222–247.

[4] Sam Blackshear, David L. Dill, Shaz Qadeer, Clark W. Barrett, John C. Mitchell, Oded Padon, Yoni Zohar, Resources: a safe language abstraction for money, CoRR, arXiv:2004.05106, 2020.

[5] Harris Brakmić, Bitcoin Script, Apress, Berkeley, CA, 2019, pp. 201–224.

[6] Michael J. Coblenz, Jonathan Aldrich, Brad A. Myers, Joshua Sunshine, Can advanced type systems be usable? An empirical study of ownership, assets, and typestate in Obsidian, Proc. ACM Program. Lang. 132 (2020) 1.

[7] Silvia Crafa, Cosimo Laneve, Liquidity analysis in resource aware programming, in: Proceeding of 18th Int. Conf. on Formal Aspects of Component Software FACS 2022, 2022.

[8] Silvia Crafa, 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, in: Lecture Notes in Computer Science, vol. 13360, Springer, 2022, pp. 129–146.

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

[10] Chris Dannen, Introducing Ethereum and Solidity: Foundations of Cryptocurrency and Blockchain Programming for Beginners, Apress, Berkely, USA, 2017.
[11] A. Das, S. Balzer, J. Hoffmann, F. Pfenning, I. Santurkar, Resource-aware session types for digital contracts, in: IEEE 34th CSF, IEEE Computer Society, 2021, pp. 111–126.
[12] Mudabbir Kaleem, Anastasia Mavridou, Aron Laszka Vyper, A security comparison with solidity based on common vulnerabilities, in: BRAINS 2020, IEEE, 2020, pp. 107–111.
[13] Steve Klabnik, Carol Nichols, The RUST Programming Language, No Starch Press, 2019.
[14] Cosimo Laneve, Alessandro Parenti, Giovanni Sartor, Legal contract amending with Stipula, in: Proc. of Coordination 2023, in: Lecture Notes in Computer Science., Springer, 2023.
[15] Rocco De Nicola, Matthew Hennessy, CCS without tau's, in: Proc. of TAPSOFT'87 – Theory and Practice of Software Development, in: Lecture Notes in Computer Science, vol. 249, Springer, 1987, pp. 138–152.
[16] Pablo Lamela Seijas, Alexander Nemish, David Smith, Simon J. Thompson, Marlowe: implementing and analysing financial contracts on blockchain, in: Financial Cryptography and Data Security, in: LNCS, vol. 12063, Springer, 2020, pp. 496–511.
[17] Ilya Sergey, Vaivaswatha Nagaraj, Jacob Johannsen, Amrit Kumar, Anton Trunov, Ken Chan Guan Hao, Safer smart contract programming with scilla, Proc. ACM Program. Lang. 3 (OOPSLA) (2019) 185:1–185:30.
[18] Petar Tsankov, Andrei Marian Dan, Dana Drachsler-Cohen, Arthur Gervais, Florian Bünzli, Martin T. Vechev, Securify: practical security analysis of smart contracts, in: Proc. ACM SIGSAC Conference on Computer and Communications Security, ACM, 2018, pp. 67–82.