Smart digital contracts: Algebraic foundations for resource accounting

Fritz Henglein

Email: henglein@diku.dk, henglein@deondigital.com

OPLSS 2019, 2019-06-26

Recall

Agents: Persons, companies, robots, devices that sign events and their evidence

Events: Significant real-world events that update the state of the (business) world

- Business events: Transmission of information and other events whose resource effect is idempotent (e.g. queries)
- Resource events: Producing (transforming) and transferring resources, which have a resource effect (who owns or possesses what)

Resources: Physical (goods, services) or digital (money, rights) resources that cannot/must not be freely copied and discarded

Contract: A classifier of event sequences into "happy" paths (correct contract executions) and "breaches" (incorrect contract executions).

Today

- Algebraic model of resources, with user-definable resource types ("multi-currency")
- Resource ownership via coproducts
- Resource transfers via kernels
- Operations and properties: vector space operations and basic linear algebra

Vector spaces

Definition

- Field: $(K, +, -, 0, \cdot, /, 1)$, commutative ring with multiplication and division
- Vector space over K: $(V, +, -, 0, \cdot)$, usual properties
- Dimension of vector space: Cardinality of smallest subset of V that spans all of V

Example

The reals $\mathbb{R}$ are a field and simultaneously a vector space of dimension 1 over itself.

Vector space constructions

Let V_x be vector spaces.

$\prod_{x \in X} V_x$ (product): Functions f from $x : X$ to V_x

$\coprod_{x \in X} V_x$ (coproduct): Functions f from $x : X$ to V_x with *finite support* $\mathrm{Supp}(f) = \{x \mid f(x) \neq 0\}$; that is, *finite maps* with default return value 0.

$V \to_1 W$ (linear map space): Functions (linear maps) f from V to W such that $f(v_1 + v_2) = f(v_1) + f(v_2)$ and $f(k \cdot v) = k \cdot f(v)$.

$U \subseteq V$ (subspace): Subset U of V that is closed under $0, +, -, \cdot$

If $V_x = V$ for all $x \in X$, write

$$\prod_X V = \prod_{x \in X} V$$

$$\coprod_X V = \coprod_{x \in X} V$$

Vector space constructions: Examples

Let X be a set.

$V_1 \oplus V_2$ (direct sum): $\coprod_{x \in \{1,2\}} V_i$ $(= V_1 \times V_2)$

$\mathrm{Free}_K(X)$ (free vector space): $\coprod_X K$

$\sum : (\coprod_X V) \to_1 V$ (sum, addition):

$$\sum(\{x_1 : v_1, \ldots, x_n : v_n\}) = v_1 + \ldots + v_n$$

$p^* : \mathrm{Free}_K(X) \to_1 K$ (valuation under price $p : X \to K$): Unique extension of p to $\mathrm{Free}_K(X)$.

$\ker f \subseteq V$ (kernel of $f : V \to W$): $\{x \in V \mid f(x) = 0\}$.

$\mathrm{im}\, f \subseteq W$ (image of $f : V \to W$): $\{f(x) \mid x \in V\}$.

Vector space constructions: Examples of examples

- $(5, 8) \in \mathbb{R} \oplus \mathbb{R} = \mathbb{R}^2$
- $5 \cdot X_1 + 8 \cdot X_2 = \{X_1 : 5, X_2 : 8\} \in \coprod_{\{X_1, X_2\}} \mathbb{R}$
- $\sum\{X_1 : 5, X_2 : 8\} = 5 + 8 = 13$
- $p^*(\{X_1 : 5, X_2 : 8\}) = 4 \cdot 5 + 3 \cdot 8 = 44$
  for $p(X_1) = 4, p(X_2) = 3$.
- $\ker p^* = \{\{X_1 : x_1, X_2 : x_2\} \mid 4 \cdot x_1 + 3 \cdot x_2 = 0\}$.
- $\operatorname{im} p^* = \mathbb{R}$.

Agents and resources

Agents A:	A set.	$A = \{\text{Alice}, \text{Bob}, \text{Charlie}, \ldots\}$.
Resource types X:	A set.	$X = \{\text{USD}, \text{iPhone}, \ldots\}$.
Resources R:	A vector space.	$R = \coprod_X \mathbb{R}$
Ownership states O:	A vector space.	$O = \coprod_A R$
Transfers T:	Subspace of O.	$T = \sum_X R = \ker(\sum : \coprod_A R \to_1 R)$

Example

- A *simple* resource: $50 \cdot \text{USD}$
- A *compound* resource: $50 \cdot \text{USD} + 2 \cdot \text{iPhone}$
- A *missing* resource is also a resource: $-50 \cdot \text{USD}$
- An ownership state: $\{\text{Alice} : 50 \cdot \text{USD}, \text{Bob} : 1 \cdot \text{iPhone} + 10 \cdot \text{USD}\}$
- A *simple* (2-party) transfer: $\{\text{Alice} : -30 \cdot \text{USD}, \text{Bob} : 30 \cdot \text{USD}\}$
- A *compound* (multi-party) transfer: $\{\text{Alice} : -30 \cdot \text{USD}, \text{Bob} : 20 \cdot \text{USD}, \text{Charlie} : 10 \cdot \text{USD}\}$

Resource manager

- Credit limit policy: Predicate (Boolean function), classifying ownership states into *valid* and *invalid* ones
  - Usually : $P_{A_0,c}(o) = o(a) \geq c(a)$ for all $a \in A_0$ where $A_0 \subseteq A$.
- Resource manager: Object (service) with
  - Internal state o: An ownership state satisfying credit limit policy P.
  - Method ApplyTransfer:
    Receive transfer t.
    If $P(o + t)$, update internal state to $o + t$ and return "success"; otherwise, return "failure".

Example

Credit limit policy:	No credit (no negative amounts of any resource type)
Initial ownership:	$o_1 = \{\mathrm{Alice} : 50 \cdot \mathrm{USD}, \mathrm{Bob} : 1 \cdot \mathrm{iPhone} + 10 \cdot \mathrm{USD}\}$
First transfer:	$t_1 = \{\mathrm{Alice} : -30 \cdot \mathrm{USD}, \mathrm{Bob} : 30 \cdot \mathrm{USD}\}$
Second transfer:	$t_2 = \{\mathrm{Alice} : 1 \cdot \mathrm{iPhone}, \mathrm{Bob} : -1 \cdot \mathrm{iPhone}\}$
Combined transfer:	$\{\mathrm{Alice} : 1 \cdot \mathrm{iPhone} - 30 \cdot \mathrm{USD}, \mathrm{Bob} : -(1 \cdot \mathrm{iPhone} - 30 \cdot \mathrm{USD})\}$
Final ownership:	$o_2 = \{\mathrm{Alice} : 1 \cdot \mathrm{iPhone} + 20 \cdot \mathrm{USD}, \mathrm{Bob} : 40 \cdot \mathrm{USD}\}$

Ownership state as balance plus transfer

Theorem

Let $f : V \to_1 W$. *Then:*

$$\begin{aligned} V &\cong \mathrm{im}\, f \oplus \ker f \\ \dim V &= \dim(\mathrm{im}\, f) + \dim(\ker f). \end{aligned}$$

Corollary

$$O = \coprod_A R \cong R \oplus \sum_A R = R \oplus T$$

Intuitively: Ownership state $\cong$ a *resource balance* owned by one particular agent $b \in A$ and some transfer; for example:

$$\begin{aligned} o &= \{\mathrm{Bank} : 60 \cdot \mathrm{USD}, \mathrm{Alice} : 30 \cdot \mathrm{USD}, \mathrm{Bob} : 40 \cdot \mathrm{USD}\} \\ &= \{\mathrm{Bank} : 130 \cdot \mathrm{USD}\} + \\ &\quad \{\mathrm{Bank} : -70 \cdot \mathrm{USD}, \mathrm{Alice} : 30 \cdot \mathrm{USD}, \mathrm{Bob} : 40 \cdot \mathrm{USD}\} \end{aligned}$$

Resource manager properties

- A multiset $M = \{t_1, \ldots, t_n\}$ of transfers can be applied by a resource manager in *any* order: any two orders that succeed result in the same ownership state. Some orders may fail, however, due to the resource manager's credit limit policy.
- If there is *some* successful order of applying M satisfying P, then applying the *single* "netted" transfer $t = \sum M = \sum_{i=1}^{n} t_i$ is valid, too. The converse is *not* true.
- The internal ownership state can be stored as a pair, a balance and a transfer.
- The balance component in a resource manager is invariant. Only the transfer component is updated by ApplyTransfer.

Zero-balance resource managers

- Balance of a resource manager can be kept in another resource manager.
- *Zero-balance resource manager:* internal state of resource manager consists of a transfer only; resource balance component is implicitly 0.

Zero-balance resource managers: Example

Two resource managers:

$$
\begin{aligned}
o_1 &= \{\mathrm{Bank}_1 : 60 \cdot \mathrm{USD}, \mathrm{Alice} : 30 \cdot \mathrm{USD}, \mathrm{Bob} : 40 \cdot \mathrm{USD}\} \\
&= \{\mathrm{Bank}_1 : 130 \cdot \mathrm{USD}\} + \\
&\quad \{\mathrm{Bank}_1 : -70 \cdot \mathrm{USD}, \mathrm{Alice} : 30 \cdot \mathrm{USD}, \mathrm{Bob} : 40 \cdot \mathrm{USD}\} \\
o_2 &= \{\mathrm{Bank}_2 : 10 \cdot \mathrm{USD}, \mathrm{Alice} : 100 \cdot \mathrm{USD}, \mathrm{Bob} : 200 \cdot \mathrm{USD}\} \\
&= \{\mathrm{Bank}_2 : 310 \cdot \mathrm{USD}\} + \\
&\quad \{\mathrm{Bank}_1 : -300 \cdot \mathrm{USD}, \mathrm{Alice} : 100 \cdot \mathrm{USD}, \mathrm{Bob} : 200 \cdot \mathrm{USD}\}
\end{aligned}
$$

Replace by three resource managers mainting transfers only:

$$
\begin{aligned}
t_1 &= \{\mathrm{Bank}_1 : -70 \cdot \mathrm{USD}, \mathrm{Alice} : 30 \cdot \mathrm{USD}, \mathrm{Bob} : 40 \cdot \mathrm{USD}\} \\
t_2 &= \{\mathrm{Bank}_1 : -300 \cdot \mathrm{USD}, \mathrm{Alice} : 100 \cdot \mathrm{USD}, \mathrm{Bob} : 200 \cdot \mathrm{USD}\} \\
t_0 &= \{\mathrm{Bank}_0 : -440 \cdot \mathrm{USD}, \mathrm{Bank}_1 : 130 \cdot \mathrm{USD}, \mathrm{Bank}_2 : 310 \cdot \mathrm{USD}\}
\end{aligned}
$$

where Bank_0 is another agent, corresponding to the *central bank* in the banking system or the *equity account* in a company's chart of accounts.
Note: $\{\mathrm{Bank}_0 : -\sum(o_1 + o_2)\} + o_1 + o_2 = t_0 + t_1 + t_2$ is a transfer.

Double-entry bookkeeping

Fundamental principle of double-entry bookkeeping:

- All (scalar) account ($\cong$ agent) balances sum to 0.
- Every transaction consists of multiple ("double") account entries that sum to 0.

"Equity" plays role of resource balance when decomposing ownership state into resource balance and transfer satisfying

$$\text{Assets} - \text{Liabilities} - \text{Equity} = 0$$

Resource accounting

Resource accounting: Double-entry bookkeeping, generalized to admit

- arbitrary *resources*, not just scalars, with
- expressive algebra (vector space) of *transfers* that *are not* composed from possibly incorrect adding/subtracting to/from account balances, but from a *base of simple transfers*; and
- arbitrary *report functions* on internal state,
  - often *linear maps* on internal ownership states or on sequences of transfers T^*, and then
  - easily incrementalized to maintain report function results online (dynamically) as new transfers arrive.

A resource manager (implemented whichever way) provides digital resource management for arbitrary (including user-defined) resource types.

- Updating by *transfers only* guarantees *resource preservation*: No managed resource is duplicated or lost.
- *Credit limit enforcement* by checking of credit limit policy.

Distributed resource managers by additive decomposi

- Idea: Implement *distributed resource manager* r by a P2P network of resource managers $r_1, \ldots, r_n$ such that $r.o = r_1.o + \ldots r_n.o$.
- The r_i may be distributed themselves. Advantages:
  - Some transfers can be performed *locally*: If r_i can validate and effect a transfer t, then no communication with other resource managers is necessary.[1]
  - In general, decompose transfer t into $t = t_1 + \ldots + t_n$ and *transactionally* execute all t_i to r_i. No communication with r_i is required if $t_i = 0$.

[1] Assume credit limit policy of r is conjunction of credit limit policies $r_1, \ldots, r_n$.

Distributed resource managers: Example

Let r consist of resource managers r_1, r_2 with current ownership states

$$\begin{aligned}
o_1 &= \{\mathrm{Bank}_1 : 60 \cdot \mathrm{USD}, \mathrm{Alice} : 30 \cdot \mathrm{USD}, \mathrm{Bob} : 40 \cdot \mathrm{USD}\} \\
&= \{\mathrm{Bank}_1 : 130 \cdot \mathrm{USD}\} + \\
&\quad \{\mathrm{Bank}_1 : -70 \cdot \mathrm{USD}, \mathrm{Alice} : 30 \cdot \mathrm{USD}, \mathrm{Bob} : 40 \cdot \mathrm{USD}\} \\
o_2 &= \{\mathrm{Bank}_2 : 10 \cdot \mathrm{USD}, \mathrm{Alice} : 100 \cdot \mathrm{USD}, \mathrm{Bob} : 200 \cdot \mathrm{USD}\} \\
&= \{\mathrm{Bank}_2 : 310 \cdot \mathrm{USD}\} + \\
&\quad \{\mathrm{Bank}_1 : -300 \cdot \mathrm{USD}, \mathrm{Alice} : 100 \cdot \mathrm{USD}, \mathrm{Bob} : 200 \cdot \mathrm{USD}\}
\end{aligned}$$

and zero-credit policy (only nonnegative balances allowed).

- Transfer $\{\mathrm{Alice} : -80 \cdot \mathrm{USD}, \mathrm{Bob} : 80 \cdot \mathrm{USD}\}$ can be performed by r_2 without communication with r_1.
- Transfer $\{\mathrm{Alice} : -120 \cdot \mathrm{USD}, \mathrm{Bob} : 120 \cdot \mathrm{USD}\}$ cannot be performed by either r_1 or r_2, but it can be decomposed into $t_1 + t_2$ where $t_1 = \{\mathrm{Alice} : -20 \cdot \mathrm{USD}, \mathrm{Bob} : 20 \cdot \mathrm{USD}\}$ and $t_2 = \{\mathrm{Alice} : -100 \cdot \mathrm{USD}, \mathrm{Bob} : 100 \cdot \mathrm{USD}\}$ and then performed by *transactionally* executing t_1 on r_1 and t_2 on r_2.

Distributed resource managers: Transactionality

Nodes in a distributed resource manager need to support atomic execution of distributed transactions, e.g. for 2-phase commit:

- *Precommit* transfer t: Like ApplyTransfer, but with guarantee that, if validated, subsequent execution of $-t$ will succeed. For simple transfers: deducts resource from sender, but does not make it available yet to receiver.
- *Commit* transfer t: Apply previously precomitted t (remove requirement that $-t$ must be applicable later on). For simple transfer: releases resource to receiver.
- *Abort* transfer t: Apply $-t$ to previously precommitted t. For simple transfer: return resource to sender.

Distributed resource managers: Discussion

- Many freely combinable “dimensions” of decomposition possible:
  - By resource type (e.g. land registry managing houses; national banking system (with individual banks as “peers”) managing USD accounts; the Bitcoin network for managing Bitcoin accounts (UTxOs), etc.
  - By agents (e.g. residents divided into countries of residence)
  - By statically or dynamically splitting off resource managers from existing resource managers for privacy and/or load balancing purposes (e.g. state channels, sharding).
- Resource managers should have API for participating in distributed transactions.
- Algebraic resource model as semantic basis for large design space for distributed resource managers.

Summary

- Algebra of transfers: infinite-dimensional vector space.
  - The power of negative: Additive inverses important.
- Separation of resource preservation (unrestricted algebra) and credit limit policies (restrictions).
- Additive decomposition of transfers: partitioning of resource managers for distributed implementation.