

KLAIM, certified: Mechanising flow logic for tuple-space coordination

Marino Miculan

Department of Mathematics, Computer Science and Physics, University of Udine, Italy

ARTICLE INFO

Keywords:

KLAIM
Coordination
Flow logic
Control-flow analysis
Mechanised metatheory
Locally nameless
Rocq
Language-based security

ABSTRACT

Control-flow analysis in the style of Flow Logic has long been a central tool in the Pisa school of language-based security: it provides an over-approximating static semantics whose soundness ensures that statically established security properties are preserved at run time. In this paper we revisit a representative instance of this programme, the static analysis of tuple trajectories in a language of the KLAIM family, and certify it in the Rocq proof assistant.

We work with the located-network fragment on which the original trajectory analysis is defined: first-order tuple-space coordination over a finite, statically known set of localities, with no code mobility. For this fragment we mechanise the syntax, operational semantics and Flow Logic specification of the analysis using a locally nameless representation with multi-binders of statically varying arity. We then machine-check the main metatheoretic properties of the analysis: closure of acceptable estimates under meets, yielding a least acceptable estimate when one exists; subject reduction; and a trajectory-confinement theorem stating that sensitive data never reach forbidden sites whenever the static estimate excludes them.

The mechanisation also exposes a methodological point that is easy to leave implicit on paper. The 0-CFA distinguishes binders by their canonical labels, and is therefore equivariant, rather than invariant, with respect to α -renaming: renaming binders preserves acceptability only when the abstract environment is renamed accordingly. Making this naming discipline explicit reconciles locally nameless binding with finitely supported abstract environments. Finally, acceptability is proved decidable, and the decision procedure is extracted to OCaml, yielding a certified checker for the analysed fragment.

1. Introduction

A recurring thread in Gian-Luigi Ferrari's work, from the inception of KLAIM [1] to its most recent revisitations [2], is the conviction that the coordination of distributed components and the static guarantees one can give about it are two sides of the same enterprise. The line of research he pursued together with Bodei, Degano and Galletta [3–5] makes this concrete: a *flow logic* [6] that statically over-approximates where tuples can travel in a network of tuple spaces, and from that approximation reads off security-relevant properties such as the confinement of sensitive data to trusted nodes.

This paper is a small homage to that programme. We take the static analysis of KLAIM tuple *trajectories* [3] (itself first presented in a Festschrift¹) and *certify* it inside a proof assistant [7].² Gian-Luigi has long taken an interest in the formalisation of concurrency theory, and we hope he will enjoy seeing the trajectory analysis reconstructed on that footing. The analysis is unchanged in spirit; what changes is the strength of the guarantee behind it.

E-mail address: marino.miculan@uniud.it

¹ Fittingly, the analysis we build upon appeared in the volume dedicated to Rocco De Nicola, KLAIM's co-author [1].

² The full Rocq development is publicly available at <https://github.com/miculan/FlowKlaiminRocq> and archived on Zenodo [8].

$v ::= b \mid l$	values
$\ell ::= \text{self} \mid l \mid x$	locality targets
$t ::= \langle v_1, \dots, v_n \rangle$	tuples (ground)
$F ::= v \mid x$	template fields (v actual, x formal)
$T ::= \langle F_1, \dots, F_n \rangle$	templates
$P ::= \text{nil} \mid \text{out}(t)@ \ell.P \mid \text{in}(T)@ \ell.P$ $\quad \mid \text{read}(T)@ \ell.P \mid P_1 \mid P_2 \mid A(\tilde{v})$	processes
$\text{node} ::= s ::_{\rho} P \mid s :: t$	located nodes (process $\mid$ datum)
$N ::= \text{multiset of node}$	networks

Fig. 1. Syntax of the fragment under examination. A fixed environment $\Delta : A \mapsto (\tilde{x}, P_A)$ provides process definitions $A(\tilde{x}) \triangleq P_A$; $\rho : \mathcal{V} \rightarrow \mathbb{S}$ is a *static* allocation environment.

Contributions. We give a complete, machine-checked reconstruction of the trajectory analysis [3], for the first-order, located-network fragment of KLAIM on which it is defined (Section 2): first-order tuple-space coordination over a finite, statically known set of localities, with no code mobility. That the locality space is fixed—never grown at run time—keeps the universe of sites finite, the property the least-estimate result exploits, while the analytically interesting phenomenon, the dynamic communication topology induced by tuples carrying localities, is fully present. The mechanisation is *locally nameless* (Section 4); as the only binders are the formal fields of a template, bound all at once, it lives in a *multi-binder* setting of statically varying arity. On this basis we machine-check the metatheory of Section 5: a Moore-family theorem (Theorem 5.1), subject reduction (Theorem 5.2) resting on a single substitution lemma (Lemma 5.1), and a trajectory-confinement theorem (Theorem 5.3) that is the security reading of the analysis.

Along the way the mechanisation forces a methodological point into the open (Section 4.2): a finite 0-CFA flow logic is *equivariant* rather than *invariant* under binder renaming. Its abstract environment $\hat{\rho}$ uses binder labels as cache addresses, so renaming the program alone, at a fixed estimate, may change the judgement, while renaming program and estimate together does not. Fixing a canonical naming of binders is therefore not a matter of convenience but part of the analysis—a discipline the pen-and-paper presentation relies on silently. Section 4.2 shows why the cofinite convention of locally nameless metatheory cannot be used here, and how naming the binders restores a finitely supported analysis.

Finally, Section 6 places the work in context, and Section 7 concludes.

2. A first-order located-network fragment of KLAIM

We work with the first-order, located-network fragment of the KLAIM family on which the trajectory analysis of Bodei, Degano, Ferrari and Galletta is defined [3]: the located-network presentation of the μ KLAIM core [1,9], with the out/in/read tuple-space primitives, parallel composition, and recursive process definitions.

Localities are not created at run time: the set $\mathbb{S}$ of physical sites and the set $\mathcal{L}$ of locality constants are *finite* and fixed (exactly those occurring in the program text) so there are no fresh names, no scope extrusion and no name abstraction. The communication topology is still dynamic (tuples carry localities) but the universe of sites is fixed, and it is this finiteness on which the least-estimate result of Theorem 5.1 ultimately rests.

There is no code mobility: the trajectory property studied here concerns *data*, not code, and first-order mobility of data already captures the phenomenon targeted by the original analysis.

Fig. 1 gives the syntax. We let b range over a set $Base$ of base values equipped with a decidable equality and a sensitivity predicate $\text{sens}(\cdot)$; l over locality constants; x over variables.

Following [3], a template field is either an *actual*, i.e., a value v , which matches identically, or a *formal*, which is a variable x that binds the matched value. The actions $\text{in}(T)@ \ell.P$ and $\text{read}(T)@ \ell.P$ bind *all* the formal variables of T in P simultaneously; their number is the arity $\text{ar}(T)$; out binds nothing.

We depart from [3] in two ways that are inessential for the trajectory analysis and for the metatheory below. First, we let process definitions carry *parameters*, $A(\tilde{x}) \triangleq P_A$, with the invocation $A(\tilde{v})$ binding $\tilde{x}$ to $\tilde{v}$ in P_A ; this slightly *generalises* the parameterless definitions $A \triangleq P_A$ of [3], and the analysis and its metatheory go through unchanged. Second, we take tuple fields and values to be first-order leaves, omitting the unspecified expressions $f(E_1, \dots, E_n)$ of [3]: these would add only an opaque evaluation step before matching and change nothing essential about trajectories.

Representing a network as a *multiset* of located nodes lets the *network*-level structural congruence (commutativity and associativity of $\parallel$, and the empty network as unit) be absorbed by the associativity and commutativity of the multiset, so it need not be mechanised explicitly. The *process*-level parallel $P \mid Q$ is an explicit constructor, dissolved into two located nodes by the reduction rule (PAR), as in the located-network presentation of [3].

Operational semantics. The reduction is the standard μ KLAIM tuple-space semantics [1,9], specialised to the located-network, first-order fragment underlying the trajectory analysis [3] (Fig. 2). Let $\llbracket \ell \rrbracket_{\rho} \in \mathbb{S}$ resolve a locality target at the node located at s : $\llbracket \text{self} \rrbracket_{\rho} = s$,

(OUT)	$s ::_{\rho} \text{out}(t) @ \ell . P \rightarrow s ::_{\rho} P, \llbracket \ell \rrbracket \rho :: t$
(IN)	$\frac{\text{match}(T, t) = \sigma}{s ::_{\rho} \text{in}(T) @ \ell . P, \llbracket \ell \rrbracket \rho :: t \rightarrow s ::_{\rho} P\sigma}$
(READ)	$\frac{\text{match}(T, t) = \sigma}{s ::_{\rho} \text{read}(T) @ \ell . P, \llbracket \ell \rrbracket \rho :: t \rightarrow s ::_{\rho} P\sigma, \llbracket \ell \rrbracket \rho :: t}$
(CALL)	$s ::_{\rho} A(\tilde{v}) \rightarrow s ::_{\rho} P_A[\tilde{v}/\tilde{x}] \quad \text{if } \Delta(A) = (\tilde{x}, P_A)$
(PAR)	$s ::_{\rho} P Q \rightarrow s ::_{\rho} P, s ::_{\rho} Q$
(GC)	$s ::_{\rho} \text{nil} \rightarrow \varepsilon$
(CTX)	$\frac{N \rightarrow N'}{N, M \rightarrow N', M}$

Fig. 2. Reduction over multisets of located nodes (N, M range over nets, the comma being multiset union). Both (IN) and (READ) instantiate the continuation by applying the matching substitution σ ; (IN) additionally consumes the datum. (PAR) splits a parallel thread into two located nodes, and (GC) discards a terminated one (ε the empty net), orienting the structural law $\ell :: P \equiv \ell :: (P | \text{nil})$ of [3]. (CTX) extends a reduction to any ambient net, so every rule fires anywhere in the multiset; in the mechanisation this is the rules' explicit tail plus closure under permutation (s_{perm}).

$\llbracket I \rrbracket \rho = \rho(I)$, and variables in target position are already substituted to constants when used. Following [3], pattern matching is the Linda-style associative matching of KLAIM, written $\text{match}(T, t) = \sigma$: a template T matches a tuple t when they have equal length and, position by position, every actual field v of T equals the corresponding value of t ; the resulting σ is the substitution sending each formal variable x of T to the value matched at its position. If no match exists, $\text{match}(T, t) = \perp$. We write $P\sigma$ for the (capture-avoiding) application of σ to P .

Remark 2.1. (A may-analysis cannot see destruction) In the over-approximating analysis of Section 3, *in* and *read* have the *same* effect on the abstract tuple component $\hat{\kappa}$: a may-analysis does not observe the destructive nature of *in*. This is sound for the safety property we target, but the mechanisation makes it impossible to forget, and we record it here.

3. A flow logic for tuple trajectories

The analysis is a 0-CFA presented, in the Flow Logic style of [6,10], as an *acceptability* relation: a predicate on a proposed estimate that holds exactly when the estimate is a sound over-approximation. We follow the trajectory analysis of [3], whose abstract tuple component $\hat{\kappa}$ maps each site to the tuples that may reach it. The abstract domains form a complete product lattice—the structural reason behind the Moore-family theorem:

$$\hat{\rho} : \mathcal{V} \rightarrow \wp(\mathcal{V}al), \quad \hat{\kappa} : \mathbb{S} \rightarrow \wp(\text{Tuple}).$$

A pair $(\hat{\rho}, \hat{\kappa})$ in this lattice is an *estimate*; it is *acceptable* for a network N when $(\hat{\rho}, \hat{\kappa}) \models N$, the relation defined next (Definition 3.1). The metatheory of Section 5 establishes that acceptable estimates form a Moore family, hence have a least element whenever acceptability is satisfiable, and that acceptability is preserved along reductions.

Abstract evaluation of fields and tuples, and abstract resolution of targets at a host site s (yielding sites, $\llbracket \ell \rrbracket_s^{\#} \hat{\rho} \subseteq \mathbb{S}$), are:

$$\begin{aligned} \llbracket v \rrbracket \hat{\rho} &= \{v\} \\ \llbracket x \rrbracket \hat{\rho} &= \hat{\rho}(x) \\ \llbracket \langle F_1, \dots, F_n \rangle \rrbracket \hat{\rho} &= \{\langle v_1, \dots, v_n \rangle \mid v_i \in \llbracket F_i \rrbracket \hat{\rho}\} \\ \llbracket \text{self} \rrbracket_s^{\#} \hat{\rho} &= \{s\} \\ \llbracket I \rrbracket_s^{\#} \hat{\rho} &= \{\rho(I)\} \\ \llbracket x \rrbracket_s^{\#} \hat{\rho} &= \{\rho(I) \mid I \in \hat{\rho}(x) \cap \mathcal{L}\} \end{aligned}$$

Definition 3.1. (Acceptability) The relation $(\hat{\rho}, \hat{\kappa}) \models N$ (adapted from [3]) holds node by node (for threads, by induction on the process):

$$\begin{aligned} (\hat{\rho}, \hat{\kappa}) \models s :: t & \quad \text{iff } t \in \hat{\kappa}(s) \\ (\hat{\rho}, \hat{\kappa}) \models s ::_{\rho} \text{nil} & \quad \text{always} \\ (\hat{\rho}, \hat{\kappa}) \models s ::_{\rho} \text{out}(t) @ \ell . P & \quad \text{iff } (\forall s' \in \llbracket \ell \rrbracket_s^{\#} \hat{\rho}. \llbracket t \rrbracket \hat{\rho} \subseteq \hat{\kappa}(s')) \wedge (\hat{\rho}, \hat{\kappa}) \models s ::_{\rho} P \\ (\hat{\rho}, \hat{\kappa}) \models s ::_{\rho} \text{in}(T) @ \ell . P & \quad \text{iff } \forall s' \in \llbracket \ell \rrbracket_s^{\#} \hat{\rho}. \forall t = \langle v_1, v_n \rangle \in \hat{\kappa}(s') \text{ compatible with } T. \\ & \quad (\forall i. F_i \text{ a formal } x_i \Rightarrow v_i \in \hat{\rho}(x_i)) \wedge (\hat{\rho}, \hat{\kappa}) \models s ::_{\rho} P \\ (\hat{\rho}, \hat{\kappa}) \models s ::_{\rho} A(\tilde{v}) & \quad \text{iff } \Delta(A) = (\tilde{x}, P_A) \wedge \llbracket \tilde{v} \rrbracket \hat{\rho} \subseteq \hat{\rho}(\tilde{x}) \\ (\hat{\rho}, \hat{\kappa}) \models s ::_{\rho} P | Q & \quad \text{iff } (\hat{\rho}, \hat{\kappa}) \models s ::_{\rho} P \wedge (\hat{\rho}, \hat{\kappa}) \models s ::_{\rho} Q \end{aligned}$$

The clause for *read* is identical to that for *in* (cf. Remark 2.1). Following the input rule of [3], the continuation $(\hat{\rho}, \hat{\kappa}) \models s ::_{\rho} P$ sits *under* the quantification over covered compatible tuples: it is required exactly when some covered tuple matches T . Equivalently, since

the acceptability of the continuation does not depend on the particular tuple matched, the input clause can be decomposed into two conditions: all covered compatible tuples must contribute their matched values to the caches of the corresponding formals, and the continuation must be acceptable whenever at least one covered compatible tuple exists. This is the form used in the mechanisation: the existence of such a tuple is represented as a first-order side condition, while the recursive occurrence of acceptability for the continuation remains a positive premise.

The parallel clause makes acceptability of a thread agree with acceptability of the two nodes its (PAR) reduction produces.

Remark 3.1. (Definitions are analysed once, globally) The call clause deliberately does *not* unfold the body P_A : it only demands that the actuals flow into the formals' caches, $\llbracket \bar{v} \rrbracket \hat{\rho} \subseteq \hat{\rho}(\bar{x})$. Inlining the body makes the acceptability relation ill-founded on recursive definitions: no finite derivation can witness A if A calls itself. Hence, the inductively defined judgement would be empty on every cyclic Δ . Instead the bodies are checked *once and for all*, by a separate global condition

$$(\hat{\rho}, \hat{\kappa}) \models \Delta \quad \text{iff} \quad \forall A. \Delta(A) = (\bar{x}, P_A) \Rightarrow \forall s. (\hat{\rho}, \hat{\kappa}) \models s ::_{\rho} P_A,$$

with the formal variables $\bar{x}$ free in the body P_A (in the mechanisation, opened at the formals, see Section 4).

In the subject-reduction proof, the two parts meet again: the global condition on Δ gives acceptability of each body opened at its canonical formals, and the call clause gives exactly the cache inclusions needed to instantiate those formals with the actual values by the substitution lemma. Indeed, *acceptability of a network* is the conjunction

$$(\hat{\rho}, \hat{\kappa}) \models N \quad \triangleq \quad (\forall \text{node} \in N. (\hat{\rho}, \hat{\kappa}) \models \text{node}) \wedge (\hat{\rho}, \hat{\kappa}) \models \Delta,$$

written *Accept* in the development $(\text{SatNet} \wedge \text{SatDelta})$. This is the judgement meant in Theorems 5.1, 5.2 and 5.3 below.

Every clause is a conjunction of monotone inclusions $\subseteq$: this Horn/co-additive shape is exactly what yields the least estimate, in line with the Moore-family characterisation of acceptable analyses in Flow Logic [6].

Example 3.1. (A two-hop trajectory) Take three sites s_1, s_2, s_3 , an allocation environment with $\rho(l_2) = s_2$ and $\rho(l_3) = s_3$, and the network

$$\begin{aligned} s_1 &::_{\rho} \text{out}(\langle d \rangle) @ l_2. \mathbf{nil}, \\ s_2 &::_{\rho} \text{in}(\langle x \rangle) @ \text{self}. \text{out}(\langle x \rangle) @ l_3. \mathbf{nil}, \\ s_3 &::_{\rho} \text{in}(\langle y \rangle) @ \text{self}. \mathbf{nil}, \end{aligned}$$

which relays the datum d from s_1 to s_2 and on to s_3 . Operationally, s_1 deposits $\langle d \rangle$ at s_2 by (OUT); s_2 reads it, binding $x \mapsto d$ by (IN), and forwards $\langle d \rangle$ to s_3 ; s_3 reads it, binding $y \mapsto d$. The least acceptable estimate records exactly this trajectory:

$$\hat{\kappa}(s_2) = \hat{\kappa}(s_3) = \{\langle d \rangle\}, \quad \hat{\rho}(x) = \hat{\rho}(y) = \{d\}, \quad \hat{\kappa}(s_1) = \emptyset,$$

so d may reach $\{s_2, s_3\}$ and nowhere else. Read against a policy *Forbidden* (Theorem 5.3): if d is sensitive and *Forbidden* $\cap \{s_2, s_3\} = \emptyset$, the estimate *certifies* that d never lands on a forbidden site in every run, by soundness. Enlarging the net never shrinks the estimate, and closure under $\sqcap$ (Theorem 5.1) is what makes “the least such estimate” well defined.

4. Locally nameless, mechanised KLAIM

In the fragment we are considering, the formal variables x are bound by *in/read* in their continuation. We thus need a theory of substitution. We adopt a *locally nameless* representation [11,12], with one twist specific to KLAIM: an $\text{in}(T)$ binds all the formals of T *at once*, so we are in a *multi-binder* setting of statically varying arity. Binders admit other mechanised representations—notably (*weak*) *higher-order abstract syntax*, which encodes object-level binding by meta-level functions over a distinguished type of names, as done e.g. for the π -calculus [13]. We prefer the locally nameless style here for its first-order, fully explicit account of substitution, which the value-for-name lemma (Lemma 5.1) places at the heart of subject reduction.

4.1. Datatypes

Listing 1 shows the core datatypes. The bound-variable constructor carries a pair (k, j) : the de Bruijn depth k and the position j within that binder—precisely what a multi-binder needs.

Two operations make the nameless representation usable. *Opening* instantiates the outermost binder, replacing each $\text{fbvar } 0 j$ by the j th entry of a list us of field expressions (of length the binder's arity); descending under a binder increments the depth k . Since the substituends are *leaves* (i.e., closed values) opening neither shifts indices nor captures names: this is the simplification bought by first-order data over a fixed locality space.

Locally nameless syntax is raw: a term may carry bound indices that point past every enclosing binder. *Local closure* is the predicate that singles out the genuine terms—those in which every fbvar is resolved by a binder. Usually, it is defined *cofinitely*, quantifying a body over *all* sufficiently fresh names. Here the binder instead *carries* the names $\bar{x}$ of its formals, so local closure opens the body with those very names (required distinct, of the right arity, and fresh for the body), rather than over fresh vectors.

This names-on-the-binder discipline is exactly what the analysis needs (Section 4.2); the cofinite formulation, quantifying over all fresh name vectors, is untenable there. It also gives a clean account of the surface-level substitution. The reduct $P\sigma$ of Fig. 2, with σ sending the formals $\bar{x}$ to the matched values $\bar{v}$, is—since the formals sit in the body as $\text{fbvar } 0 j$ —just the opening $\text{open } P(\text{fval } \bar{v})$; the analysis opens the *same* binder with the canonical names, $\text{open } P(\text{fvar } \bar{x})$. The two uses of one open (fval for reduction, fvar for the analysis) are bridged by the substitution lemma of Section 5.

Listing 1. Locally nameless datatypes (Rocq).

```

Inductive fexp :=
| f_val  : val  -> fexp
| f_fvar : name -> fexp      (* free: name = syntactic label
                               of the binder *)
| f_bvar : nat  -> nat -> fexp. (* bound: (k = depth, j =
                               position in binder) *)
Inductive loc := l_self | l_exp : fexp -> loc.
Definition tuple := list fexp.      (* ground when all f_val *)
Inductive tfield :=                (* template field: actual / formal slot *)
  tf_act : fexp -> tfield | tf_form : tfield.
Definition templ := list tfield.    (* template *)

Inductive proc :=
| p_nil  : proc
| p_out  : tuple -> loc -> proc -> proc
| p_in   : templ -> loc -> list name -> proc -> proc (* the
  formals' canonical names *)
| p_read : templ -> loc -> list name -> proc -> proc
| p_call : defname -> list fexp -> proc
| p_par  : proc -> proc -> proc.

```

Listing 2. Vector-shaped opening.

```

Fixpoint open_fexp (k:nat) (us:list fexp) (e:fexp) : fexp :=
  match e with
  | f_val _    => e
  | f_fvar _    => e
  | f_bvar i j => if Nat.eqb i k then nth j us e else f_bvar i j
  end.
(* open_loc, open_tuple, open_templ: structural;
   open_proc: increments k under p_in/p_read/definition body. *)

```

Listing 3. Local closure for the named multi-binder.

```

| lc_in : forall T l xs P,
  lc_templ T -> lc_loc l ->
  NoDup xs -> length xs = arity T -> vdisjoint xs (fv_proc P) ->
  lc (open_proc 0 (map f_fvar xs) P) ->
  lc (p_in T l xs P)
(* lc_read analogous; lc_call over the definition body and
   d_params. *)

```

4.2. Canonical names for a finite 0-CFA

Here the mechanisation forces a methodological point into the open. The 0-CFA assigns one cache $\hat{\rho}(x)$ to each syntactic binder label: the abstract environment tells binders apart by the names chosen for them. Hence acceptability is not invariant under renaming the program while keeping the estimate fixed. If a binder labelled x is renamed to y , the cache used by the input clause changes from $\hat{\rho}(x)$ to $\hat{\rho}(y)$, and the judgement may change unless the estimate is renamed as well. Thus the right invariance principle is equivariance under simultaneous renaming of the program and of the abstract environment, not invariance under program renaming at a fixed estimate.

This distinction is exactly where the usual cofinite locally nameless discipline clashes with the analysis. A cofinite input clause would require the cache condition to hold for all sufficiently fresh names used to open the binder. As soon as a covered tuple can be read, the matched value would have to belong to $\hat{\rho}(x)$ for cofinitely many names x , and the resulting abstract environments have infinite support. This is not the finite object computed by the 0-CFA, whose purpose is to associate one cache with each canonical binder label occurring in the program.

The reconciliation is to make the canonical names *part of the syntax*. A binder $\text{in}(T)@_{\ell}.P$ carries the names $\tilde{x}$ of its formals—exactly as a definition $A(\tilde{x}) \triangleq P_A$ carries $\tilde{x}$ —while the body still refers to them by de Bruijn position ($\text{fbvar } 0.j$). The analysis reads $\tilde{x}$

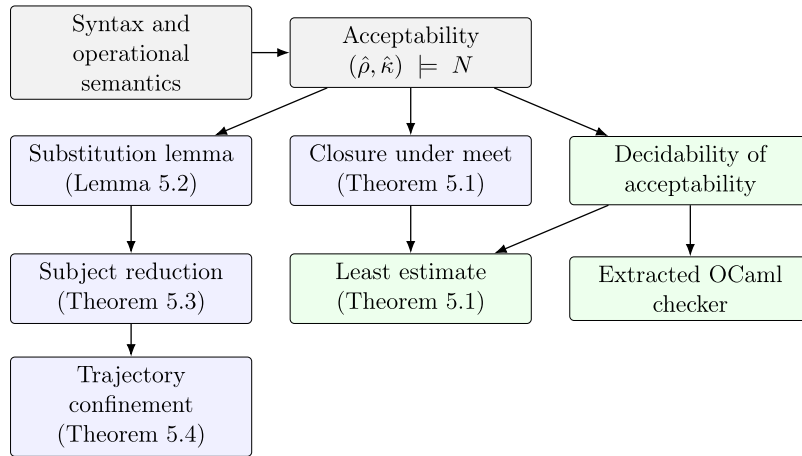


Fig 3. How the development fits together. Grey: the two definitions, the operational semantics and the acceptability relation. Blue: what is proved about acceptability—the substitution lemma carries subject reduction, which carries trajectory confinement; closure under meet is the first half of [Theorem 5.1](#). Green: the constructive half—acceptability is decidable, which yields both the least estimate (the second half of [Theorem 5.1](#)) and, by extraction, a checker that decides the very relation the soundness theorems speak about.

Table 1

Correspondence between the statements of this paper and the Rocq development. Each name links to the file that contains it in the public repository.

This paper	Development	File
Acceptability (Definition 3.1)	SatP, SatNet, SatDelta, Accept	theories/Analysis.v
Two-hop relay (Example 3.1)	relay, rh_least, kh_least	extraction/main.ml
Well formedness (Definition 5.1)	wf_params, wf_net	theories/Theorems.v
Moore family (Theorem 5.1)	T1_closed_under_meet	theories/Theorems.v
least estimate	T1_least_estimate_exists	theories/LeastEstimate.v
decidability	decide_Accept	theories/LeastEstimate.v
Substitution lemma (Lemma 5.1)	subst_preservation, key_subst_preservation	theories/Theorems.v
Subject reduction (Theorem 5.2)	T2_subject_reduction	theories/Theorems.v
Confinement (Theorem 5.3)	coverage, T3_confinement	theories/Theorems.v

off the binder and indexes $\hat{\rho}$ by those names directly; there is no quantifier over namings, and $\hat{\rho}$ is finitely supported. Put plainly, the canonical names are the *finite representatives of the binding occurrences* of the program: each occurrence owns one address in the cache, so the estimate is a finite map over those addresses rather than a condition quantified over all the ways of naming them. We keep the locally nameless representation for the *metatheory* (reduction, substitution, local closure, where α -invariance is a virtue) and the bridge between the two worlds is the substitution lemma of [Section 5](#) ([Lemma 5.1](#)). The price of named binders is that capture must be avoided explicitly: substitution past a binder requires the substituted name to differ from the binder's names, so we ask the canonical names to be globally distinct (a Barendregt convention, wf_{net} and wf_{params} of [Definition 5.1](#)).

The observation—a static analysis need not be α -invariant, so a canonical naming must be fixed, and naming the binders gives a principled one together with a clean substitution—is the kind of thing a mechanisation turns from folklore into a theorem.

5. Mechanised metatheory

We state the three results and the lemma on which subject reduction turns. Proofs are carried out in Rocq [\[7\]](#); here we give their structure and, in parentheses, the name of the corresponding development lemma. [Fig. 3](#) shows how the pieces fit together, and [Table 1](#) links every numbered statement of this paper to the item that carries it in the development; the names in that table are hyperlinks to the source files. Two of the three theorems carry a standing well-formedness assumption—the canonical-naming discipline of [Section 4.2](#)—on both the definition environment and the network, which we fix once.

Definition 5.1. (Well formedness) The definition environment Δ is *well formed* (wf_{params}) when, for every A with $\Delta(A) = (\tilde{x}, P_A)$, the formal parameters $\tilde{x}$ are pairwise distinct, of the right arity, and fresh for P_A (its free names and its nested binder names), and P_A is itself well formed.

A network N is *well formed* (wf_{net}) when every in/read binder occurring in it satisfies the same conditions on its name vector.

This is the canonical-naming hypothesis of [Section 4.2](#): globally distinct binder names are what let $subst_{intro}$ relate $open\ P\ (fval\ \tilde{v})$ to a capture-free substitution into $open\ P\ (fvar\ \tilde{x})$. Well formedness of the network is preserved by reduction ($wf_{netstep}$), hence holds along every trajectory.

Theorem 5.1. (Moore family / least estimate) *For every network N , the set $\{(\hat{\rho}, \hat{\kappa}) \mid (\hat{\rho}, \hat{\kappa}) \models N\}$ of acceptable estimates is closed under componentwise meet $\sqcap$ in the product lattice $(\mathcal{V} \rightarrow \wp(\text{Val})) \times (\mathbb{S} \rightarrow \wp(\text{Tuple}))$. Consequently, whenever N admits at least one acceptable estimate, it admits a least one.*

Proof sketch Closure under meet ($\text{TI}_{\text{closed_under_meet}}$) is by induction on the acceptability derivation, using componentwise monotonicity of abstract evaluation; it propagates uniformly through SatNet and SatDelta and does not require finiteness of Base . For the least estimate ($\text{TI}_{\text{least_estimate_exists}}$) we exploit the finiteness of the fragment constructively. Acceptability is decidable ($\text{decide}_{\text{accept}}$): each clause is a bounded conjunction over the finite sets produced by abstract evaluation, and the recursion is well founded on process size, since opening preserves it. Restricting an acceptable estimate to a finite program universe keeps it acceptable. This universe contains the binder names and values occurring in N and Δ , and, for tuples, not only the ground tuples literally occurring in the program but also all tuples of the arities appearing in out-actions that can be formed from those values. This closure is needed because an output tuple may contain variables whose run-time instantiations combine values that occur separately in the program.

The non-emptiness proviso is intentional. Acceptability is not only a system of flow constraints: it also checks that the syntactic entities referred to by the network are well defined, in particular that process identifiers are defined in Δ and that calls have the expected arity. Raw networks violating these conditions may therefore have no acceptable estimate at all. Thus meet-closure yields a least estimate precisely for those networks for which acceptability is satisfiable; this is the statement mechanised in Rocq .

The cardinal lemma is where mechanisation pays off, and the only place where the choice of binder representation is felt. $\square$

Lemma 5.1. (Preservation under value-for-name substitution) *Let $\tilde{x}$ be distinct and fresh for the binder names of P . If $(\hat{\rho}, \hat{\kappa}) \models \text{open } P(\text{fvar } \tilde{x})$ and for every j we have $v_j \in \hat{\rho}(x_j)$, then $(\hat{\rho}, \hat{\kappa}) \models [\tilde{x} \mapsto \vec{v}](\text{open } P(\text{fvar } \tilde{x}))$.*

Proof sketch By induction on the acceptability derivation of $\text{open } P(\text{fvar } \tilde{x})$, pushing the substitution through abstract evaluation with the substitution/opening lemmas of Section 4 (in Rocq : $\text{subst}_{\text{preservation}}$, iterated to the vector case as $\text{key}_{\text{subst}_{\text{preservation}}}$). The step is immediate because the substituends are leaves. The hypothesis $v_j \in \hat{\rho}(x_j)$ is what the in/read clause of Definition 3.1 guarantees when the matched tuple is covered by $\hat{\kappa}$. $\square$

Theorem 5.2. (Subject reduction) *If Δ and N are well formed (Definition 5.1), $(\hat{\rho}, \hat{\kappa}) \models \Delta$, and $(\hat{\rho}, \hat{\kappa}) \models N$ with $N \rightarrow N'$, then $(\hat{\rho}, \hat{\kappa}) \models N'$.*

Proof sketch By induction on the reduction ($\text{T2}_{\text{subject_reduction}}$). In the interesting cases, (IN)/(READ)/(CALL), the reduct is a substitution $P\sigma$ (with $\sigma = [\tilde{x} \mapsto \vec{v}]$), which the encoding performs as $\text{open } P(\text{fval } \vec{v})$; the identity $\text{open } P(\text{fval } \vec{v}) = [\tilde{x} \mapsto \vec{v}](\text{open } P(\text{fvar } \tilde{x}))$ ($\text{subst}_{\text{ntrop}_{\text{proc}}}$) rewrites it as a capture-free substitution into the analysed body $\text{open } P(\text{fvar } \tilde{x})$, reducing the case to Lemma 5.1. Here $\tilde{x}$ are the binder's own canonical names rather than a freshly chosen vector; well formedness of N (wf_{net}) supplies their distinctness and freshness, licensing the capture-free substitution. (CALL) additionally consumes $(\hat{\rho}, \hat{\kappa}) \models \Delta$ for the acceptability of the opened body, and well formedness of Δ ($\text{wf}_{\text{params}}$) at the parameters. (PAR) is immediate: acceptability of $P|Q$ is by definition acceptability of P and of Q (Sat_{par}), so splitting the parallel thread into two nodes preserves $\models$; (GC) likewise only drops a (trivially acceptable) terminated thread. (CTX)—a reduction firing anywhere in the multiset—is realised, in the list-based network, by stability of $\models$ under permutation ($\text{SatNet}_{\text{perm}}$).

Finally, the security reading. Let $\text{sens}(\cdot)$ mark sensitive tuples and let $\text{Forbidden} \subseteq \mathbb{S}$ be a policy of untrusted sites. $\square$

Theorem 5.3 (Trajectory confinement). *Let Δ , N be well formed, and such that $(\hat{\rho}, \hat{\kappa}) \models \Delta$ and $(\hat{\rho}, \hat{\kappa}) \models N$. If the estimate satisfies $\forall s \in \text{Forbidden}. \forall t \in \hat{\kappa}(s). \neg \text{sens}(t)$, then in every N' reachable from N , no sensitive datum resides at a forbidden site.*

Proof sketch Follows from Theorem 5.2 and a coverage lemma: subject reduction lifts to the reflexive-transitive closure ($\text{steps}_{\text{preserve}_{\text{satNet}}}$), with well formedness carried along as an invariant ($\text{steps}_{\text{preserve}_{\text{wf}}}$); acceptability then forces every datum $s :: t$ of every reachable state to lie in $\hat{\kappa}(s)$, and the policy excludes sensitive data at forbidden sites ($\text{T3}_{\text{confinement}}$). $\square$

Mechanisation status. The development is publicly available at <https://github.com/miculan/FlowKlaiminRocq> and archived on Zenodo [8], whose concept DOI always resolves to the latest version. It is structured in the files *Syntax*, *Semantics*, *Analysis*, *LN*, *LeastEstimate* and *Theorems*. It builds under Rocq 9.1.1 with *Metalib*; the repository pins both—an opam switch for the prover and a commit hash for *Metalib*—and a continuous-integration workflow rebuilds the development from that pinned environment, checks that no proof is admitted, and re-extracts and runs the checker. All the results of this paper are fully proved: the locally nameless infrastructure (Section 4), the cardinal lemma (Lemma 5.1), subject reduction (Theorem 5.2), confinement (Theorem 5.3), and both halves of Theorem 5.1 (closure under meet and the explicit construction of the least estimate). The least-estimate construction rests on the decidability of acceptability ($\text{decide}_{\text{accept}}$) and on the locality of acceptability under restriction to the finite program universe ($\text{U}_{\text{locality}}$, $\text{value}_{\text{locality}}$); the least estimate is then the meet of the finitely many acceptable U -bounded estimates. For the results reported in this paper, *Print Assumptions* reports only the abstract interface of the fragment: the base sorts Base , S , L and definition names, their decidable equalities, the finiteness assumptions for S , L and Δ , and the designated home site. In particular, the statements do not depend on logical axioms such as classical logic, choice, or functional extensionality.

A certified checker. Since Definition 3.1 is a conjunction of decidable inclusions over finite carriers, acceptability is decidable, and we prove it so constructively ($\text{decide}_{\text{accept}}$), that is, the very procedure that drives the least-estimate construction of Theorem 5.1. We extract it to OCaml, obtaining a *certified* acceptability checker: the analysis is then not merely sound on paper but validates estimates by running verified code. The extracted procedure is parametric in the finite interface of the fragment (the sorts and their decidable equalities, the enumerations of $\mathbb{S}$, $\mathcal{L}$ and the defined names, and the home site); realising it with concrete finite types gives a runnable

checker that decides acceptability of example nets, its only trusted part being the interface realisation, the decision logic itself being certified by extraction.

To make this concrete, the development realises that interface with three sites and two localities and runs the checker on the relay of [Example 3.1](#). The realisation uses genuinely finite sum types, so that the finiteness assumptions on $\mathbb{S}$ and $\mathcal{L}$ hold of it; these are not idle, since the enumeration of allocation environments inside the decision procedure depends on them. On that network the checker accepts the least estimate given in [Example 3.1](#) and rejects three strictly smaller ones:

Example 3.3, two-hop relay:

least estimate	expect true	got true	ok
second hop dropped	expect false	got false	ok
$\rho\text{-hat}(y)$ dropped	expect false	got false	ok
empty estimate	expect false	got false	ok

Each rejection fails a different clause: dropping $\hat{\kappa}(s_3)$ breaks the out clause of the continuation at s_2 , dropping $\hat{\rho}(y)$ breaks the formal condition of the input at s_3 , and the empty estimate already fails at the first hop. The driver exits with a non-zero status if any verdict differs from the expected one, so the example doubles as a regression test in the continuous-integration workflow.

6. Related work

KLAIM and its static guarantees. KLAIM [1] introduced localities as first-class data and tuple-space coordination with *network awareness*; μ KLAIM [9] distilled a minimal core, and the family is still being refined [2]. Static guarantees for KLAIM have classically been delivered by *type systems* governing access rights and mobility [1,9]. We take an orthogonal route for the fragment considered here: instead of a type discipline we certify a *flow logic*, an over-approximating analysis in the abstract-interpretation tradition.

Flow logic and control-flow analysis. The flow-logic style we adopt is that of Nielson and Nielson [6]; its use for security is exemplified by the control-flow analysis of LySA for cryptographic protocols [10], developed in the same Pisa-DTU collaboration. The specific analysis we mechanise is the trajectory analysis of [3], which over-approximates where tuples may travel; it has since been enriched with quantitative metrics [4], modelling support [5] and risk assessment [14]. We take the foundational analysis and add the ingredient a pen-and-paper development omits—a machine-checked soundness proof.

Symbolic protocol verification. A complementary style of static reasoning about security is *symbolic*, as implemented in tools such as ProVerif [15] and applied to deployed protocols [16]. In that setting, protocols, attacker capabilities and security goals are translated into Horn clauses and discharged by specialised resolution procedures, proving secrecy or authentication (or exhibiting attacks) with high precision, at the price of no general termination guarantee. The flow logic certified here sits at the other end of that trade-off: its constraints are monotone and Horn-shaped over a finite abstract domain, acceptability is decidable, and a least acceptable estimate always exists ([Theorem 5.1](#)). The abstraction is deliberately coarse (as observed in [Remark 2.1](#), $\hat{\kappa}$ does not record that *in consumes* a tuple) but the loss of must-information is harmless for confinement ([Theorem 5.3](#)), which needs only an over-approximation of the sites where sensitive data may reside, and it is precisely what buys totality. The two approaches are thus complementary: one gives precision when the symbolic search succeeds, the other a certified may-guarantee by construction.

Certified static analysis. Our work also belongs to the broader line of research on mechanically verified static analyses and abstract interpreters, where proof assistants are used either to certify the soundness of an analysis, to extract a verified analyser, or to connect an executable checker with a semantic correctness theorem [17]. The present work follows this tradition, but in a setting that is less common than imperative programs: a coordination calculus with tuple-space communication, binding templates, and a Flow-Logic-style acceptability judgement. The main technical issue is therefore not numerical abstraction or fixpoint iteration per se, but the interaction between tuple matching, binder names, substitution, and the subject-reduction argument for a process calculus. This is where the locally nameless, multi-binder formalisation and the canonical-naming discipline of [Section 4](#) enter the proof.

Mechanised metatheory. Our development follows the locally nameless methodology [11] and the engineering practices of [12], here generalised to a multi-binder of statically varying arity. To our knowledge, this is the first machine-checked soundness proof for a flow logic of a coordination calculus of the KLAIM family. More broadly, mechanised soundness proofs for control-flow analyses of process calculi remain uncommon, and the canonical-naming issue brought out in [Section 4.2](#) (usually left implicit on paper) is one reason such mechanisations are delicate.

7. Conclusions

We have given a machine-checked reconstruction of a flow logic for tuple trajectories in a coordination language of the KLAIM family. The analysis is faithful to the one Ferrari and his co-authors designed [3]; we add a certified soundness together with a methodological dividend, the recognition that the finite 0-CFA presentation assumes a canonical naming discipline for binders.

Decidability turns the metatheory into a tool. The decision procedure `decideaccept` extracts to executable OCaml, yielding a checker that is *correct by construction*: it decides exactly the acceptability relation the theorems speak about, so a positive verdict

is a proof of acceptability and, through subject reduction and trajectory confinement (Theorems 5.2 and 5.3), a certificate that the network keeps its sensitive data off the forbidden sites. There is no trusted gap between the analysed relation, its soundness proof, and the running checker. The development bundles the two-hop relay of Example 3.1 as an executable check: the extracted checker accepts the least estimate given there and rejects three strictly smaller estimates, each failing a different clause of acceptability.

Future work. The fragment was deliberately frozen (Section 2), and each feature set aside is a natural next step: name generation and restriction (newloc), code mobility (the higher-order eval), non-deterministic choice, and arithmetic on tuple fields. Name generation is the most demanding, as it would exercise the locally nameless infrastructure on genuinely fresh names, beyond the canonical labels that suffice here. Orthogonally, sharpening the analysis from 0-CFA to a context-sensitive k -CFA, or extending it to the quantitative and metric refinements of the same line of work [4,5], would carry the certified guarantees to a more precise analysis. The extracted checker, finally, invites a usable front end (a parser and a reporting interface) so that the certified analyser can be run on networks of realistic size. Further afield, the question of where a datum can travel and whether it can be kept away from untrusted nodes is equally pressing for the data-driven, *attribute-based* languages used to program reactive and IoT systems, modelled by calculi such as AbU [18]. There the communication topology is dynamic for a different reason: the receivers of an update are not named by a locality but selected at run time by a predicate over node attributes, so the abstract counterpart of $\llbracket \mathcal{C} \rrbracket_{\delta}^{\#}$ would over-approximate the nodes whose attributes may satisfy a guard, while the rest of the recipe certified here (closure under meet, subject reduction, decidable acceptability) should carry over.

CRedit authorship contribution statement

Marino Miculan: Writing – review & editing, Writing – original draft, Validation, Software, Resources, Methodology, Investigation, Funding acquisition, Formal analysis, Conceptualization.

Declaration of generative AI and AI-assisted technologies in the manuscript preparation process

During the preparation of this work, the author used Anthropic’s Claude, through the Claude Code interface, to draft and revise prose, and to assist with the Rocq formalisation and the OCaml extraction that accompany it. The author reviewed and edited the output as needed and takes full responsibility for the content of the published article.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

I am grateful to Chiara Bodei, Rocco De Nicola and Pierpaolo Degano for inviting me to contribute to this Festschrift. And I dedicate this work, warmly, to Gian-Luigi Ferrari, whom I have known since my doctoral years in Pisa, in the now-distant ’90s—first as a colleague, and then as a friend.

Data availability

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

References

- [1] R. De Nicola, G.-L. Ferrari, R. Pugliese, KLAIM: a kernel language for agents interaction and mobility, *IEEE Trans. Softw. Eng.* 24 (5) (1998) 315–330. <https://doi.org/10.1109/32.685256>
- [2] L. Bettini, G.-L. Ferrari, M. Loret, R. Pugliese, F. Tiezzi, E. Tuosto, Klaim in the making, in: *Leveraging Applications of Formal Methods, Verification and Validation (ISoLA 2024)*, Part I, vol. 15219 of *Lecture Notes in Computer Science*, Springer, 2024, pp. 27–49. https://doi.org/10.1007/978-3-031-73709-1_3
- [3] C. Bodei, P. Degano, G.-L. Ferrari, L. Galletta, Revealing the trajectories of KLAIM tuples, statically, in: M. Boreale, F. Corradini, M. Loret, R. Pugliese (Eds.), *Models, Languages, and Tools for Concurrent and Distributed Programming – Essays Dedicated to Rocco De Nicola on the Occasion of His 65th Birthday*, vol. 11665 of *Lecture Notes in Computer Science*, Springer, Cham, 2019, pp. 437–454. https://doi.org/10.1007/978-3-030-21485-2_24
- [4] C. Bodei, P. Degano, G.-L. Ferrari, L. Galletta, Security metrics at work on the things in IoT systems, in: A. Di Pierro, P. Malacaria, R. Nagarajan (Eds.), *From Lambda Calculus to Cybersecurity through Program Analysis – Essays Dedicated to Chris Hankin on the Occasion of His Retirement*, vol. 12065 of *Lecture Notes in Computer Science*, Springer, Cham, 2020, pp. 233–255. https://doi.org/10.1007/978-3-030-41103-9_9
- [5] C. Bodei, P. Degano, G.-L. Ferrari, L. Galletta, Modelling and analysing IoT systems, *J. Parallel Distrib. Comput.* 157 (2021) 233–242. <https://doi.org/10.1016/j.jpdc.2021.07.004>
- [6] F. Nielson, H.R. Nielson, C. Hankin, *Principles of Program Analysis*, Springer, 1999. <https://doi.org/10.1007/978-3-662-03811-6>
- [7] The Rocq Development Team, The Rocq Prover, 2025, Formerly the Coq proof assistant. <https://rocq-prover.org>.
- [8] M. Miculan, FlowKlaimInRocq: a certified flow logic for KLAIM tuple trajectories, 2026, (Zenodo, v1.0.0). <https://doi.org/10.5281/zenodo.22171485>
- [9] D. Gorla, R. Pugliese, Resource access and mobility control with dynamic privileges acquisition, in: *Automata, Languages and Programming (ICALP 2003)*, vol. 2719 of *Lecture Notes in Computer Science*, Springer, 2003, pp. 119–132. https://doi.org/10.1007/3-540-45061-0_11
- [10] C. Bodei, M. Buchholtz, P. Degano, F. Nielson, H.R. Nielson, Static validation of security protocols, *J. Comput. Secur.* 13 (3) (2005) 347–390. <https://doi.org/10.3233/JCS-2005-13302>
- [11] A. Charguéraud, The locally nameless representation, *J. Autom. Reason.* 49 (3) (2012) 363–408. <https://doi.org/10.1007/s10817-011-9225-2>

- [12] B. Aydemir, A. Charguéraud, B.C. Pierce, R. Pollack, S. Weirich, Engineering formal metatheory, in: Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2008), ACM, 2008, pp. 3–15. <https://doi.org/10.1145/1328438.1328443>
- [13] F. Honsell, M. Miculan, I. Scagnetto, π -calculus in (Co)inductive-type theory, Theor. Comput. Sci. 253 (2) (2001) 239–285. [https://doi.org/10.1016/S0304-3975\(00\)00095-5](https://doi.org/10.1016/S0304-3975(00)00095-5)
- [14] C. Bodei, G.-L. Ferrari, L. Galletta, P. Degano, Risk estimation in IoT systems, in: Challenges of Software Verification, vol. 238 of *Intelligent Systems Reference Library*, Springer, 2023, pp. 221–242. https://doi.org/10.1007/978-981-19-9601-6_12
- [15] B. Blanchet, Modeling and verifying security protocols with the applied Pi calculus and ProVerif, Found. Trends Priv. Secur. 1 (1–2) (2016) 1–135. <https://doi.org/10.1561/33000000004>
- [16] M. Miculan, N. Vitacolonna, Automated symbolic verification of telegram’s MTPROTO 2.0, in: Security and Cryptography (SECURITY 2021), SCITEPRESS, 2021, pp. 185–197. <https://doi.org/10.5220/0010549601850197>
- [17] D. Cachera, T.P. Jensen, D. Pichardie, V. Rusu, Extracting a data flow analyser in constructive logic, Theor. Comput. Sci. 342 (1) (2005) 56–78. <https://doi.org/10.1016/j.tcs.2005.06.008>
- [18] M. Miculan, M. Pasqua, A calculus for attribute-based memory updates, in: A. Cerone, P.C. Ölveczky (Eds.), Theoretical Aspects of Computing (ICTAC 2021), vol. 12819 of *Lecture Notes in Computer Science*, Springer, Cham, 2021, pp. 366–385. https://doi.org/10.1007/978-3-030-85315-0_21