A Machine-Checked Necessity Result for Accountable Delegated Authority

Fitzgerald Jones Heslop
AgentOBO Inc.
`founder@agentobo.ai`

August 28, 2026

Formalized in Lean 4. Zero `sorry` placeholders.
Verified source sealed 25 August 2026.

A Machine-Checked Necessity Result for Accountable Delegated Authority

Fitzgerald Jones Heslop
AgentOBO Inc.
founder@agentobo.ai

August 28, 2026

Abstract

Authorization logics for delegated authority have been studied since the early 1990s, but their central claims rest on pen-and-paper argument. Recent industrial formalization has targeted *implementation conformance* — proving that a policy evaluator behaves as specified — rather than the structure such a specification must have. We formalize in Lean 4 a six-conjunct predicate O on delegated authority — cryptographic binding, identity, mandate, scope, revocation dominance, and appointment — together with a single-governor cardinality constraint, and prove a necessity result: any system achieving accountable delegated authority in an area of consequence never permits an execution failing O. Each conjunct is shown non-redundant by a necessity theorem exhibiting its individual failure. The cardinality constraint is derived as a theorem from a narrower cryptographic assumption rather than postulated. We explicitly disclaim the converse direction. We further distinguish the structure from six prior systems — ABLP, Taos, SPKI/SDSI, Macaroons, Biscuit, and UCAN — by countermodel against each system's own documented mechanism rather than a common template, and we state the resulting claim in its narrow form: no single prior system among those examined unifies all six conjuncts with an explicit cardinality constraint. The development compiles in Lean 4 with zero errors, zero warnings, and zero `sorry` placeholders; nine irreducible axioms remain, each named and attributed, and four comparison theorems report zero axiom dependency under `#print axioms`.

1 Introduction

1.1 The problem

Autonomous software agents now initiate consequential actions — moving funds, approving claims, mutating systems of record — on behalf of principals absent at the moment of execution. The governing question at that moment is not whether the action succeeded but whether it was authorized: by whom, under what grant, within what scope, revocable by whom, and provable to a third party afterward.

Existing infrastructure answers this partially and downstream. Policy engines decide whether a request is permitted; logs record what was written. Neither produces a portable artifact binding a specific act to a specific grant under a named principal, verifiable by a party trusting neither the agent nor the system that hosted it.

1.2 Two kinds of formal work

- **Implementation conformance.** An artifact exists; prove it behaves as specified. This is the Cedar programme [7, 9].

- **Basis-level results.** No artifact is presupposed; prove properties of the structure any adequate system must instantiate — which conditions are required, and whether each is individually necessary.

Cedar and its successors occupy the first; this paper occupies the second. Neither substitutes for the other. A proved evaluator does not establish whether its policy language rests on a sufficient condition set, because that question is not asked in the first programme. No system in the conformance cohort asserts a minimal basis for authority, so none proves one necessary.

1.3 Presentation and proved structure

The framework from which this development arises is presented publicly under five primitives: Appointment, Agent, Authority, Accountability, Auditability. The formal result is stated over *six* conjuncts plus a cardinality constraint. The two are related as informal grouping to formal decomposition, and the mapping is given in Table 1. This paper's claims are the formal ones.

Conjunct	**Type**	**Reading**
C	Prin $\to$ Ag $\to$ Act $\to$ T $\to$ Prop	appointment signed by P's private key
I	Prin $\to$ Prop	P is identified and non-anonymous
M	Ag $\to$ Act $\to$ Prop	the act is specified with particularity
S	Ag $\to$ Act $\to$ Prop	bounds on the agent's authority are defined
R	Prin $\to$ Ag $\to$ Act $\to$ T $\to$ Prop	P can set this false, killing execution
Appointed	Prin $\to$ Ag $\to$ Act $\to$ T $\to$ Prop	pre-execution act binding P to A for X at t

Table 1: The six conjuncts of O, as formalized.

1.4 Contributions

1. A Lean 4 formalization of a six-conjunct predicate on delegated authority, with all definitions explicit and reviewable (§3).

2. Six necessity theorems, one per conjunct, each exhibiting that conjunct's individual failure as sufficient to defeat O (§5).

3. A single-governor uniqueness result derived as a theorem from a narrower cryptographic assumption, rather than postulated (§6).

4. A reduction theorem in the necessity direction, with the converse explicitly disclaimed inside the development (§7).

5. Mechanical countermodels against six prior systems, each constructed from that system's own documented mechanism (§8), and a narrow statement of what they establish (§9).

2 Related Work

2.1 Authorization and delegation logics

The modern treatment begins with Abadi, Burrows, Lampson, and Plotkin's calculus for access control in distributed systems [1], with a two-author precursor at CRYPTO '91, and its realization in the Taos operating system [2]. Both centre a `speaks-for` relation and compound principals; both establish their results by hand.

SPKI/SDSI [3] carries authorization in the certificate rather than in an identity binding. A certificate carries an issuer, a subject — which may be a k-of-n threshold of other subjects — a delegation bit, an authorization field naming exactly what is authorized, and not-before/not-after validity dates. Macaroons [4] are bearer credentials built on a chained-HMAC construction, with caveats attenuating when, where, by whom, and for what a macaroon is honored. Biscuit and UCAN continue the attenuable-token line with Ed25519-signed Datalog and DID-based variants respectively.

None of these systems appears to have been carried into a proof assistant.

2.2 Machine-checked authorization

Over approximately six months AWS constructed a formal model of Cedar in Lean and proved its components satisfy key safety and security properties [8]; the published development reports 1,673 lines of Lean model against 5,714 lines of Lean proof across evaluator, authorizer, and validator [7]. Cedar Analysis adds a symbolic compiler in Lean with soundness and completeness proofs [9]. No Cedar release ships unless its model, proofs, and differential tests are current [10].

Cedar formalizes a policy evaluator. It does not formalize a delegation calculus, and it asserts no minimal condition set for authority.

2.3 Agent admission control

The Agent Control Protocol [11] specifies an admission control layer between agent intent and system state mutation, in which every action passes a cryptographic check validating identity, capability scope, delegation chain, and policy compliance before execution, with Ed25519 identity under JCS canonicalization, capability tokens, deterministic risk evaluation, chained delegation, transitive revocation, and cryptographically chained auditing.

ACP's formal layer is TLA+: safety and liveness properties are model-checked and the resulting invariants instantiated as runtime test vectors. This is bounded model checking rather than kernel-checked theorem proving. ACP is also an implementation specification: it fixes a mechanism set and proves properties of its composition, without asking whether that set is minimal or each element necessary.

	Object proved about	**Method**	**Basis claim**
Cedar	policy evaluator	Lean 4 (kernel)	none asserted
ACP	protocol composition	TLA+ (model checking)	none asserted
ABLP / Taos / SPKI	delegation calculus	pen and paper	informal
This work	condition set	Lean 4 (kernel)	necessity per conjunct

Table 2: Placement relative to prior formal and informal work.

3 Formalization

The development sets `set_option autoImplicit false` repo-wide. This is a deliberate hygiene decision, not a stylistic one: an earlier revision of this file had free variables in `P3` silently captured by Lean's `autoImplicit`, decoupling the axiom's meaning from the surrounding documentation. Under this option any recurrence is a compile error rather than a silent reinterpretation.

3.1 Types and predicates

The four base types are implicit, so that call sites of O resolve without four leading arguments.

```
variable {Principal Agent Action Time : Type}

variable (C : Principal -> Agent -> Action -> Time -> Prop)
variable (I : Principal -> Prop)
variable (M : Agent -> Action -> Prop)
variable (S : Agent -> Action -> Prop)
variable (R : Principal -> Agent -> Action -> Time -> Prop)
variable (Appointed : Principal -> Agent -> Action -> Time -> Prop)
variable (Revoke : Principal -> Agent -> Action -> Time -> Prop)
variable (Executes : Agent -> Action -> Time -> Prop)
```

Two of these exist to prevent vacuity, and their presence should be read as a record of correction rather than as design elegance. `Revoke` is the principal's own act of invoking revocation; it exists so that Axiom 2 asserts something about *who controls* R rather than being a tautology. `Executes` binds accountability to the actual executing agent and time; it exists to close the `autoImplicit` capture noted above.

3.2 The predicate

Definition 1 (O).

```
def O (P : Principal) (A : Agent) (X : Action) (t : Time) : Prop :=
  C P A X t /\ I P /\ M A X /\ S A X /\ R P A X t /\ Appointed P A X t
```

Note that C through `Appointed` are section variables, so every result below is universally quantified over arbitrary predicates of these types. This is a strength — the results hold for any instantiation — and a limit, discussed in §11.

3.3 The system model

```
variable {AoC : Type}
variable (inAoC : Action -> AoC -> Prop)

structure ADASystem (Principal Agent Action Time : Type) where
  enforces : Principal -> Agent -> Action -> Time -> Prop
```

`ADASystem` carries the enforcement predicate it actually uses to gate execution. An opaque type here would leave `achievesADA` and the reduction theorem unconnected to the system being reasoned about — which is precisely the defect an adversarial review found in an earlier revision.

Definition 2 (`achievesADA`).

```
def achievesADA (sys : ADASystem Principal Agent Action Time) : Prop :=
  forall (P : Principal) (A : Agent) (X : Action) (t : Time) (domain : AoC),
    inAoC X domain -> sys.enforces P A X t -> O C I M S R Appointed P A X t
```

The system never permits an execution, in an area of consequence, that fails O. This is the necessity direction only.

4 Axioms

Nine irreducible axioms remain in the development. Two are this work's own; seven instantiate a named prior system's documented mechanism, solely to derive a comparison result in §8.

4.1 This work's two

Axiom 1 (`C_functional`).

```
axiom C_functional :
  forall (P P' : Principal) (A : Agent) (X : Action) (t : Time),
    C P A X t -> C P' A X t -> P = P'
```

A valid signature verifies against exactly one principal's key. Justified on standard unforgeability grounds. In an earlier revision, uniqueness was axiomatized directly over the whole six-conjunct predicate; a two-line countermodel — two distinct principals, every predicate trivially `True` — showed that nothing about O's structure forces cardinality, making that axiom unjustified independent content. `C_functional` is the narrower replacement from which uniqueness is now derived.

Axiom 2 (`P2`, revocation dominance).

```
axiom P2 :
  forall (P : Principal) (A : Agent) (X : Action) (t t' : Time),
    t < t' -> Appointed P A X t -> Revoke P A X t' -> not (R P A X t')
```

The binding must be revocable by the principal alone. This is stated in a section carrying `variable [LT Time]`, scoped to this axiom alone so that no Mathlib dependency propagates through the file. Its earlier form asserted the existence of a proposition `revoked` with `revoked` $\rightarrow \neg R$, which is a tautology — witness `revoked := False` discharges it with zero hypotheses for any R. The present form asserts who can trigger revocation.

4.2 The prior systems' seven

`ABLP.conj_speaks_for_left` and `_right` (ABLP's compound-principal rule); `SPKI.spki_threshold_empowers_left` and `_right` (RFC 2693's threshold subject); `Macaroons.copy_preserves_authorization` (the documented bearer property); `Biscuit.biscuit_signature_functional` (Ed25519 chain-signature unforgeability); `UCAN.ucan_revocation_dominance` (UCAN's own issuer-revocation mechanism). Each is stated in §8 at the point of use.

4.3 A statement to make carefully

The development's closing note records that the nine axioms contain none of this file's own claims *about* O, and separately names `C_functional` and `P2` as this file's own two. Both statements are accurate together and misleading apart: neither axiom asserts anything about the structure of O, but both are this work's assumptions rather than a prior system's. Any summary that drops the qualifier is falsifiable by a single `#print axioms` invocation, and this paper does not make one.

5 Necessity

Each conjunct is individually necessary. The six theorems below each take the failure of one conjunct and derive the failure of O.

Theorem 3 (N1–N6)**.** *For all* P, A, X, t: $\neg C\,P\,A\,X\,t \to \neg O$; $\neg I\,P \to \neg O$; $\neg M\,A\,X \to \neg O$; $\neg S\,A\,X \to \neg O$; $\neg R\,P\,A\,X\,t \to \neg O$; $\neg\text{Appointed}\,P\,A\,X\,t \to \neg O$.

```
theorem N5_necessity :
  forall (P : Principal) (A : Agent) (X : Action) (t : Time),
    not (R P A X t) ->
    not (O C I M S R Appointed P A X t) := by
  intro P A X t hR hO
  exact hR hO.2.2.2.2.1
```

N2, N3, and N4 were added in revision to close an asymmetry: mandate (M) and scope (S) were two of the six conjuncts but had no supporting proof at all, while the other four did.

5.1 Consequences of achievesADA

Four further theorems extract, from a system achieving ADA on an enforced execution within an area of consequence, the specific properties that must hold.

```
theorem P1 (sys : ADASystem Principal Agent Action Time)
    (hADA : achievesADA C I M S R Appointed inAoC sys)
    (P : Principal) (A : Agent) (X : Action) (t : Time) (domain : AoC)
    (hDomain : inAoC X domain) (hEnf : sys.enforces P A X t) :
    C P A X t /\ I P /\ Appointed P A X t := ...
```

P1 gives a verifiable binding before execution; P3 gives assignable accountability for the actual execution, taking `Executes A X t` as a hypothesis; P4 gives mandate particularity; P5 gives scope boundedness.

6 Uniqueness

Theorem 4 (O_unique)**.**

```
theorem O_unique :
  forall (P P' : Principal) (A : Agent) (X : Action) (t : Time),
    O C I M S R Appointed P  A X t ->
    O C I M S R Appointed P' A X t ->
    P = P' := by
  intro P P' A X t hO hO'
  exact C_functional C P P' A X t hO.1 hO'.1
```

At most one principal governs a delegated act. Proved, not assumed, from Axiom 1 alone. `#print axioms O_unique` *reports exactly that one dependency.*

7 The reduction, and its limit

Theorem 5 (U3_reduction)**.**

```
theorem U3_reduction :
  forall (sys : ADASystem Principal Agent Action Time),
    achievesADA C I M S R Appointed inAoC sys ->
    forall (P : Principal) (A : Agent) (X : Action) (t : Time) (domain : AoC),
```

```
      inAoC X domain -> sys.enforces P A X t ->
        O C I M S R Appointed P A X t :=
  fun _sys hADA P A X t domain hDomain hEnf => hADA P A X t domain hDomain hEnf
```

The proof term applies `hADA` to the witnessing execution. This matters because of what the theorem replaced. An earlier revision stated the reduction existentially — $\exists$ gate, $\forall P\, A\, X\, t$, gate $\leftrightarrow$ O — which is provable by picking gate := O itself, and `#print axioms` on that proof reported zero dependency on `P1`, `P2`, `P3`, or `O_unique`. That is mechanical proof that the statement was vacuous: `sys` and `achievesADA sys` were never used. Tying `ADASystem` to its own `enforces` predicate, and making `achievesADA` a real definition over it, is what makes the present statement non-trivial.

The limit, stated. This establishes the necessity half: an ADA-achieving system can never permit an execution failing O. It does not claim the converse — that such a system permits everything O allows. That stronger claim is not argued anywhere in the development and is not assumed. The disclaimer is carried in the source, not only in this paper.

8 Distinguishing prior systems

Six systems, each examined against its own documented mechanism rather than against the ABLP template, which does not fit them.

8.1 ABLP and Taos

`D1` and `D2` are type-level facts, not deep theorems. ABLP's `Says`/`SpeaksFor` carry no `Time` and no `Action` argument, so anything built from them alone is constant along those axes. But R must be able to flip from true to false across time — that is what dominance means — and M must be able to differ across actions — that is what particularity means. A predicate provably constant along an axis it would need to vary along cannot be that conjunct.

```
def TimeInvariant (Q : Time -> Prop) : Prop := forall t t', Q t <-> Q t'

theorem revocation_dominance_requires_time_variance
    [LT Time] (R' : Time -> Prop) (t t' : Time)
    (_ht : t < t') (hBefore : R' t) (hAfter : not (R' t')) :
    not (TimeInvariant (Time := Time) R') :=
  fun hInv => hAfter ((hInv t t').mp hBefore)
```

`D1_no_revocation_dominance_primitive` and `D2_not_act_specific` compose this with the invariance of ABLP-expressible predicates. `#print axioms` reports zero axiom dependency on both.

`D3` is checked the other way. ABLP's own compound-principal rule is instantiated as axioms — standard content of the calculus, not something this work invents — and Lean derives a concrete pair of distinct principals both legitimately governing the same compound principal.

```
axiom conj_speaks_for_left  : forall (A B : Prin), SpeaksFor A (Conj A B)
axiom conj_speaks_for_right : forall (A B : Prin), SpeaksFor B (Conj A B)

theorem D3_compound_principals_violate_single_governor
    (A B : Prin) (hAB : A <> B) :
    SpeaksFor A (Conj A B) /\ SpeaksFor B (Conj A B) /\ A <> B := ...
```

This is a genuine modeling difference from O's single-governor constraint, not a missing feature: `O_unique` forecloses exactly the joint-authority case ABLP treats natively. Taos implements the

same speaks-for and compound-principal logic at the OS layer, so the countermodel applies without separate formalization.

`D4` — that Taos's authentication logic is scoped to the monitor/OS layer rather than stated as a portable, system-independent predicate — is deliberately *not* given a Lean theorem. It is a claim about how the 1994 paper packages its result, not about the logic's expressiveness. Mechanizing it would be the category error an adversarial review warned against. It stands as a documented textual distinction.

8.2 SPKI/SDSI

The honest finding here is the dual of `D1`/`D2`, not a repeat of them. SPKI certificates natively carry validity dates and an authorization tag, so they are not structurally barred from time- or action-variance.

```
theorem spki_certs_can_vary_over_time
    (iss sub : Prin) (a : Action) (t1 t2 : Time)
    (hHolds : Cert iss sub a t1) (hFails : not (Cert iss sub a t2)) :
    not (ABLP.TimeInvariant (Time := Time) (fun t => Cert iss sub a t)) := ...
```

Both SPKI variance theorems report zero axiom dependency. What SPKI *does* share with ABLP is joint authority, in a more direct form: a 1-of-2 threshold subject empowers either member alone, which is a sharper single-governor violation than a binary conjunction, since no joint agreement is even required.

8.3 Macaroons

The documented bearer property is that possession of the bytes authorizes, with no cryptographic identity check at the base primitive.

```
axiom copy_preserves_authorization :
  forall (P Q : Prin), Authorized P -> Copy P Q -> Authorized Q

theorem bearer_tokens_admit_multiple_governors
    (A B : Prin) (hA : Authorized A) (hCopy : Copy A B) (hAB : A <> B) :
    Authorized A /\ Authorized B /\ A <> B := ...
```

This is not merely consistent with a single-governor violation. It is the same shape as this development's own opening countermodel against a bare uniqueness axiom — an authorization predicate with no `C_functional`-style constraint on how many principals can satisfy it — now with a documented real-world instance.

8.4 Biscuit

Each attenuation block is Ed25519-signed back to a fixed root public key, giving real cryptographic functionality on who issued each step of the delegation chain — stronger than a shared-secret HMAC chain, and the same shape as `C_functional`.

```
theorem biscuit_chain_integrity_matches_C_functional
    (P Q R : Prin) (h1 : SignedBlock P R) (h2 : SignedBlock Q R) : P = Q := ...
```

What the signed chain does not give is non-bearer presentation: holding a valid token suffices unless the Datalog policy adds an identity check as a fact it evaluates. Where that check is absent, the Macaroons result applies to Biscuit's presentation layer without change of argument, and is not re-derived.

8.5 UCAN

UCAN extends the JWT structure with DID-based issuer and audience identity, delegable capabilities, expiry and not-before, and — uniquely among the four token systems examined — an explicit issuer-invoked revocation act rather than a passive timeout.

```
axiom ucan_revocation_dominance :
  forall (Iss Sub : Prin), Appointed' Iss Sub -> UCANRevoke Iss Sub -> not (R' Iss
      Sub)

theorem ucan_matches_P2_shape
    (Iss Sub : Prin) (hApp : Appointed' Iss Sub) (hRev : UCANRevoke Iss Sub) :
    not (R' Iss Sub) := ...
```

This is a structural *agreement* with Axiom 2, not a countermodel. Of the six systems examined, UCAN is the one whose revocation model already matches O's R condition: issuer-controlled and act-triggered rather than passive. It is reported as agreement.

9 The claim, stated narrowly

These are real distinctions, but they occupy a narrower seat than a claim of wholly novel primitives would. Fragments of C-like, I-like, and `Appointed`-like content already exist across ABLP, Taos, SPKI/SDSI, and Macaroons. The defensible claim is the unification:

> No single prior system among those examined unifies all six conjuncts with an explicit cardinality constraint.

Nothing stronger is asserted.

10 Scope boundary

Six prior systems are examined: ABLP, Taos, SPKI/SDSI, Macaroons, Biscuit, UCAN. Kerberos, OAuth/OIDC, X.509 path validation, Zanzibar-style relationship-based access control, and any delegation system not named are not examined. Naming a boundary this way is a moving target by nature; it is not a claim of exhaustive coverage of everything written on access control, and should not be read as one.

11 What Lean checks, and what it does not

What is checked. Zero `sorry`. Zero errors, zero warnings. Every claim other than the nine named axioms is a proof rather than a postulate, and `#print axioms` on each theorem confirms which of the nine it actually uses. Four theorems — `ABLP.D1_no_revocation_dominance_primitive`, `ABLP.D2_not_act_specific`, `SPKI.spki_certs_can_vary_over_time`, and `SPKI.spki_certs_can_vary_over_action` — report zero axiom dependency: pure type-level facts.

Statement fidelity. Lean certifies the theorem as formalized, not that the formalization captures the intended claim. This gap is intrinsic to the method and does not close with additional proof effort. The reviewable surface is §3–§7, not the proof bodies, which the kernel already guarantees.

Abstract predicates. C, I, M, S, R, and `Appointed` are section variables of the stated types. Nothing in the development constrains what they mean beyond their arity. The necessity results therefore hold for every instantiation, which is their strength; equally, a reader who believes a particular deployment's notion of scope is not captured by *some* $S : \text{Agent} \to \text{Action} \to \text{Prop}$ has an objection this development does not answer.

Faithfulness of the prior-art fragments. Each comparison encodes a minimal fragment of the target system chosen for the specific point being checked, not a full formalization. `D1`/`D2` depend on the claim that ABLP's core relations carry no `Time` or `Action` argument, which is read off the published signature rather than proved. A reader may dispute the fragment's faithfulness; the fragments are given in full so that this dispute can be conducted over specific text.

Cryptographic trust boundary. No verified implementation of SHA-256, Ed25519, or ML-DSA-87 is assumed or provided. Cryptographic properties enter only through Axiom 1 and `biscuit_signature_functional`, both declared with stated unforgeability assumptions.

12 Artifact availability

The complete verified source, `A50.lean`, is reproduced as an appendix to the signed report of 25 August 2026 and compiles in Lean 4 with zero errors, zero warnings, and zero `sorry` placeholders. Independent re-checking requires the source, a Lean 4 toolchain, and `lean A50.lean` followed by `#print axioms` on the theorems of §5 and §6.

References

[1] M. Abadi, M. Burrows, B. Lampson, and G. Plotkin. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems*, 15(4):706–734, 1993.

[2] E. Wobber, M. Abadi, M. Burrows, and B. Lampson. Authentication in the Taos operating system. *ACM Transactions on Computer Systems*, 12(1):3–32, February 1994.

[3] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. SPKI Certificate Theory. RFC 2693, IETF, September 1999.

[4] A. Birgisson, J. G. Politz, Ú. Erlingsson, A. Taly, M. Vrable, and M. Lentczner. Macaroons: Cookies with contextual caveats for decentralized authorization in the cloud. *NDSS*, 2014.

[5] Eclipse Biscuit project. Biscuit specification. https://github.com/eclipse-biscuit/biscuit

[6] B. Zelenka et al. UCAN: User Controlled Authorization Networks. https://ucan.xyz

[7] J. W. Cutler et al. How we built Cedar: A verification-guided approach. arXiv:2407.01688, 2024.

[8] AWS Open Source Blog. Lean into verified software development. April 2024. https://aws.amazon.com/blogs/opensource/lean-into-verified-software-development/

[9] AWS Open Source Blog. Introducing Cedar Analysis. June 2025. https://aws.amazon.com/blogs/opensource/introducing-cedar-analysis-open-source-tools-for-verifying-authorization-policies/

[10] Lean FRO. Lean powers secure software at AWS. https://lean-lang.org/use-cases/cedar/

[11] M. Fernandez. Agent Control Protocol: Admission control for agent actions. arXiv:2603.18829, March 2026.

[12] L. de Moura and S. Ullrich. The Lean 4 theorem prover and programming language. *CADE*, 2021.