# Event Universes: Specification and Analysis Using Coq Proof Assistant

Grygoriy Zholtkevych[0000−0002−7515−2143]

School of Mathematics and Computer Science
V.N. Karazin Kharkiv national University,
4, Svobody Sqr., Kharkiv, 61022, Ukraine
g.zholtkevych@karazin.ua

**Abstract.** In the paper, the formal specification of event universes theory developed with using Coq Proof Assistant is presented. The main attention is paid on the discussion of the definition and obtained facts. In the same time, a proof technique is not the subject of this discussion. The reader can get acquainted with the details of the proof technique, referring to the source text of Coq-scripts hosted on the GitHub, using the links provided in the text of the paper.

**Keywords:** causality relationship· Calculus of Inductive Constructions· Coq Proof Assistant· decidability· class type· Category Theory



## 1 Introduction

Inception of distributed computation technology has posed a problem of orchestration of different computational device operating. One of the key problem in this context is to ensure adequate responses of a computational device involved into the computation to requests of other computational devices involved into the computation too in order to ensure all these computational devices behave consistently and purposefully. Thus, it is needed to give system developers tools for specification and analysis of timing constraints for ensuring the consistent behaviours of hardware and software constituted the system being under design. Hence, we may claim that carefully vetted specification guaranteeing the system behaviour consistency in the time is the important part of a good design for a distributed system.

The specificity of the design process for distributed systems is the impossibility to apply the methods of dynamic analysis of program code to ensure its correctness. The reason for the validity of this claim is that any additional observations of the system are not possible without inclusion into the system special components, which collect the needed information but at the same time these components change the system behaviour. In some sense, we have behaviour like quantum behaviour: any observation changes essentially the system behaviour. In the situation when dynamical analysis of the system correctness is not useful, the static analysis remains the unique method for assessment of system behaviour correctness. Thus, the creation of a rigorous theory, which

would ensure computer aided in developing special software for verification and analysis of causality specifications for distributed systems is a very important problem for nowadays.

To solve the problem we propose to use Coq Proof Assistant [1] and for constructing the corresponding formal theory. Informally, we use as the theoretical framework for our construction the following papers, which had initiated and developed a number of logical time models. First of all, this is L. Lamport's paper [2]. Further, we need to mention G. Winskel's papers [3]. We use also as informal background papers of C. André and F. Mallet [4,5,6]. Own results presented in [7,8] are also used.

## 2 Preliminaries: Binary Relations

Theory of binary relations form a mathematical basis for studying causality relationship. Therefore, we present the Coq-specification[1] used below for the some needed for us fragment of the binary relation theory in this section.

We stress that we assume the some variant of the extensionality formalised as the axiom called `extensionality`.

```
Axiom extensionality :
  ∀  (A B : Type) (f g : A → B), (∀  x : A, f x = g x) → f = g.
```

The next code section defines bunary relations and their properties.

```
Section EventPreliminaries.
Variable U : Type.

  Definition BiRel : Type := U → U → Prop.

  Definition Reflexive (R : BiRel) : Prop := ∀ x : U, R x x.
  Definition Irreflexive (R : BiRel) : Prop := ∀ x : U, ¬ R x x.
  Definition Symmetric (R : BiRel) : Prop := ∀ x y : U, R x y → R y x.
  Definition Transitive (R : BiRel) : Prop :=
    ∀  x y z : U, R x y → R y z → R x z.

  Inductive Preorder (R : BiRel) : Prop :=
    PreorderDef : Reflexive R → Transitive R → Preorder R.
  Inductive Equivalence (R : BiRel) : Prop :=
    EquivalenceDef :
      Reflexive R → Transitive R → Symmetric R → Equivalence R.
  Inductive StrictOrder (R : BiRel) : Prop :=
    StrictOrderDef : Irreflexive R → Transitive R → StrictOrder R.

  Definition Decidable (R : BiRel) : Prop :=
    ∀  x y : U, R x y ∨  ¬ R x y.

End EventPreliminaries.
```

[1] This specification is contained in https://github.com/gzholtkevych/Causality/blob/master/Coq/EventPreliminaries.v.

Further, we give definitions for the polymorphic identity function and composition of functions.

```
Definition id {A : Type} := fun x : A ⇒ x.
Definition compose {A B C : Type} (f : A → B) (g : B → C) : A → C :=
  fun x : A ⇒ g (f x).

Notation "g * f" := (compose f g)
  (at level 40, left associativity) : event_scope.
```

Now we prove the following propositions.

**Proposition 1 (`id_is_leftId`).**
*For any sets $A$ and $B$ and function $f : A \to B$, the equation $id_B\, f = f$ holds.*

**Proposition 2 (`id_is_rightId`).**
*For any sets $A$ and $B$ and function $f : A \to B$, the equation $f\, id_A = f$ holds.*

**Proposition 3 (`compose_is_assoc`).**
*For any sets $A$, $B$, $C$, and $D$ and functions $f : A \to B$, $g : B \to C$, and $h : C \to D$, the equation $(h\, g)\, f = h\, (g\, f)$ holds.*

We need also the following concept

```
  Definition Decidable (R : BiRel) : Prop := ∀ x y : U, R x y ∨ ¬ R x y.

End EventPreliminaries.
```

## 3 Event Universes

This section introduces in the logical time model that focuses on the causality relationships between event occurrences called instants below.

### 3.1 Informal Meaning and Mathematical Definitions

The principal causality relationship between events $A$ and $B$ is labelled by the sentence "the event $A$ causes the event $B$". We consider informally that the semantic meaning this sentence is the statement "if the event $A$ has not occurred then the event $B$ cannot occur" rather than the statement "if the event $A$ has occurred then the event $B$ should also occur". It is evident that this semantic meaning leads to the following properties of the causality relationship

1. each event causes itself;
2. if an event $A$ causes an event $B$ and the event $B$ causes an event $C$ then the event $A$ causes the event $C$.

Thus, we can accept the following definition of an event universe.

**Definition 1.** *A set $U$ equipped with a preorder (quasi-order) relation "$\ll$" is below called an event universe.*
*The statement "$x \ll y$" where $x, y \in U$ is pronounced as "$x$ causes $y$".*

It is well known that each preorder generates an equivalence and a strict order, which are below called synchronisation and precedence and denoted by "$\div$" and "$<$" respectively. The definitions of these relations are the following

**Definition 2.** *Let a set $U$ equipped with a preorder "$\ll$" be an event universe then*

1. *the relation $x \div y$ between any $x, y \in U$ (pronounced $x$ and $y$ are synchronous) is defined by the condition $x \ll y$ and $y \ll x$ and called the synchronisation relation;*
2. *the relation $x < y$ between any $x, y \in U$ (pronounced $x$ precedes $y$) is defined by the condition $x \ll y$ and $\neg y \ll x$ and called the precedence relation.*

Moreover, the next two relations called mutual exclusion and independence are useful too.

**Definition 3.** *Let a set $U$ equipped with a preorder "$\ll$" be an event universe then*

1. *the relation $x \# y$ between any $x, y \in U$ (pronounced $x$ and $y$ are mutually exclusive) is defined by the condition $x < y$ or $y < x$ and called the mutual exclusive relation;*
2. *the relation $x \parallel y$ between any $x, y \in U$ (pronounced $x$ and $y$ are independent) is defined by the condition $\neg x \ll y$ and $\neg y \ll x$ and called the independence relation.*

### 3.2 Formal Model of an Event Universe and Used Notation

Now we are ready to specify the formal model of the concept of an event universe using Coq Proof Assistant. We consider the type classes [9,10] as the appropriate construction of the Gallina specification language to describe the required model. More details, one can find in the following fragment of Coq-script[2].

The concept of an event universe introduces as follows

```
Class Event {A : Type} :=
  {  universe  := A ;
     causality  : BiRel universe ;
  (* Constraints *)
     causality_constraint (* causality is a preorder *) :
        Preorder universe causality ;
  (* Derived relations *)
     synchronisation : BiRel universe :=
        fun x y ⇒ causality x y ∧ causality y x ;
```

[2] The complete code is contained in https://github.com/gzholtkevych/Causality/blob/master/Coq/EventDefinitions.v

```
    precedence : BiRel universe :=
        fun x y ⇒ causality x y ∧ ¬ causality y x ;
    exclusion : BiRel universe :=
        fun x y ⇒ precedence x y ∨ precedence y x ;
    independence : BiRel universe :=
        fun x y ⇒ ¬ causality x y ∧ ¬ causality y x
}.
```

This script specifies an event universe as some type called `universe` equipped with a binary relations `causality`. The script imposes only one constraint, namely, the relation `causality` should be a preorder. The class definition determines also the derived relations `syncronisation`, `precedence`, `exclusion`, and `independence` in accordance with Def. 2 and 3.

In order for scripts specifying the formal theory being developed and giving proofs of the facts are more compact and readable, the following notation is introduced.

```
Notation "x ≪ y" (* x causes y *) := (causality x y)
  (at level 70) : event_scope.
Notation "x ÷ y" (* x and y are synchronous *) := (synchronisation x y)
  (at level 70) : event_scope.
Notation "x < y" (* x precedes y *) :=
  (precedence x y) (at level 70) : event_scope.
Notation "x  # y" (* x and y are mutually exclusive *) := (exclusion x y)
  (at level 70) : event_scope.
Notation "x  ‖ y" (* x and y are independent *) := (independence x y)
  (at level 70) : event_scope.
```

## 4 Properties of an Event Universe

In this section, we establish several properties of the relations mentioned in the specification of the type class `Event`. In this section, we present several simple properties of the relations mentioned in the specification of type class `Event`. Our presentations are the formulations of the properties as mathematical statements. Each such a statement is equipped with the reference to the corresponding CIC-term [11], which one can find in https://github.com/gzholtkevych/Causality/blob/master/Coq/EventRelationFacts.v.

Everywhere in this section, we assume that $U$ is an event universe equipped with the causality relation " $\ll$ " and the synchronisation relation " $\div$ ", precedence relation " $<$ ", mutual exclusion relation " $\#$ ", and independence realtion " $\|$ " are defined by Def. 2 and Def. 3.

### 4.1 Simple Properties of Relations

Firstly, we establish the properties of the synchronisation relation.

**Proposition 4** (`synchronisation_implies_causality`)**.**
*For any events $x$ and $y$, $x \div y$ implies $x \ll y$.*

**Lemma 1** (`synchronisation_is_reflexive`).
*The synchronisation relation is reflexive.*

**Lemma 2** (`synchronisation_is_transitive`).
*The synchronisation relation is transitive.*

**Lemma 3** (`synchronisation_is_symmetric`).
*The synchronisation relation is symmetric.*

These lemmas ensure immediately the following proposition.

**Proposition 5** (`synchronisation_is_equivalence`).
*The synchronisation relation is an equivalence.*

Further, we establish properties of the precedence relation.

**Proposition 6** (`precedence_implies_causality`).
*For any events $x$ and $y$, $x < y$ implies $x \ll y$.*

**Lemma 4** (`precedence_is_irreflexive`).
*The precedence relation is irreflexive.*

**Lemma 5** (`precedence_is_transitive`).
*The precedence relation is transitive.*

These lemmas ensure immediately the following proposition.

**Proposition 7** (`precedence_is_strictOrder`).
*The precedence relation is a strict order.*

Now we establish properties of the mutual exclusion relation.

**Proposition 8** (`exclusion_is_irreflexive`).
*The mutual exclusion relation is irreflexive.*

**Proposition 9** (`exclusion_is_symmetric`).
*The mutual exclusion relation is symmetric.*

The independence relation has the same properties.

**Proposition 10** (`independence_is_irreflexive`).
*The independence relation is irreflexive.*

**Proposition 11** (`independence_is_symmetric`).
*The independence relation is symmetric.*

### 4.2 Relations Incompatibility

The next group of facts concerns the incompatibility of the relations being studied.

**Proposition 12** (`incompatibility_of_synchronisation_and_exclusion`)**.** *For any events $x$ and $y$, at most one of the statements $x \doteqdot y$ and $x \# y$ is fulfilled.*

**Proposition 13** (`incompatibility_of_synchronisation_and_independence`)**.** *For any events $x$ and $y$, at most one of the statements $x \doteqdot y$ and $x \parallel y$ is fulfilled.*

**Proposition 14** (`incompatibility_of_exclusion_and_independence`)**.** *For any events $x$ and $y$, at most one of the statements $x \# y$ and $x \parallel y$ is fulfilled.*

One can note that the mutual exclusion relation between events $x$ and $y$ ensures either $x < y$ or $y < x$ but does not determine what of these precedence statements is fulfilled. To determine what precedence statement is valid one can use the following fact.

**Proposition 15** (`causality_distinguishes_exclusion`)**.** *For any events $x$ and $y$ such that $x \# y$, $x \ll y$ implies $x < y$.*

Further reasoning is aimed at identifying conditions that ensure the fulfillment of the following statement called `ixsDecomposition` that means the following

$$\text{for any events } x \text{ and } y,\ x \parallel y \vee x \# y \vee x \doteqdot y.$$

The next theorem formulates the obtained result.

**Theorem 1** (`decidable_causality_is_equivalent_to_ixsDecomposition`)**.** `ixsDecomposition` *is fulfilled if and only if the causality relation is decidable.*

### 4.3 Congruence Properties

In this subsection, we establish interrelations between the synchronisation relation and other relations being studied. The corresponding scripts proving these facts one can find at https://github.com/gzholtkevych/Causality/blob/master/Coq/EventSynchronisationCongruenceProperties.v.

**Proposition 16** (`congruence_causality`)**.** *For any events $x$, $x'$ and $y$, $y'$ such that $x \doteqdot x'$ and $y \doteqdot y'$, $x \ll y$ implies $x' \ll y'$.*

**Proposition 17** (`congruence_precedence`)**.** *For any events $x$, $x'$ and $y$, $y'$ such that $x \doteqdot x'$ and $y \doteqdot y'$, $x < y$ implies $x' < y'$.*

**Proposition 18** (`congruence_exclusion`)**.** *For any events $x$, $x'$ and $y$, $y'$ such that $x \doteqdot x'$ and $y \doteqdot y'$, $x \# y$ implies $x' \# y'$.*

**Proposition 19** (`congruence_independence`)**.** *For any events $x$, $x'$ and $y$, $y'$ such that $x \doteqdot x'$ and $y \doteqdot y'$, $x \parallel y$ implies $x' \parallel y'$.*

Summing up the above we can state that the synchronisation relation is a congruence for all other considered relations.

## 5 Morphisms of Event Universes

In this section, we beat a path to using Category Theory[3] for studying causality in distributed systems. The concept of morphism is the key to such consideration. The specification of type class presented below `EventMorphism` can be found at https://github.com/gzholtkevych/Causality/blob/master/Coq/EventDefinitions.v.

```
Class EventMorphism {A B : Type} `{Event A} `{Event B} (f : A → B) :=
  {  arrow := f ;
  (* Constraints *)
    preserving_sync : ∀ x y :  A,  x ≒  y → arrow x ≒ arrow y ;
    preserving_precedence : ∀ x y : A,  x <  y → arrow x < arrow y
  }.
```

In other words, an event morphism is a mapping of corresponding event sets that preserves the synchronisation and precedence relations.

The proving scenarios for the facts presented above one can find at https://github.com/gzholtkevych/Causality/blob/master/Coq/EventMorphisms.v

First of all, we establish that morphism preserves all relations, except perhaps independence relations.

**Proposition 20** (`preserving_sync`)**.**
*For any event universes $A$ and $B$, morphism $f : A \to B$, and events $x$ and $y$ of $A$ such that $x \doteqdot y$, the statement $fx \doteqdot fy$ is fulfilled.*

**Proposition 21** (`preserving_prec`)**.**
*For any event universes $A$ and $B$, morphism $f : A \to B$, and events $x$ and $y$ of $A$ such that $x < y$, the statement $fx < fy$ is fulfilled.*

Now we check that identity mapping is a morphism.

**Proposition 22** (`id_is_morphism`)**.**
*For any event universe $A$, $id_A$ is a morphism.*

Let us make sure that the composition of morphisms is a morphism.

**Proposition 23** (`composition_of_morphisms_is_morphism`)**.**
*For any event universes $A$, $B$, and $C$ and morphisms $f : A \to B$ and $g : B \to C$, $g\,f$ is a morphism.*

Combining Prop. 22 and 23 with Prop. 1–3 one can obtain the following theorem.

**Theorem 2.** *The class of event universes equipped with morphisms is a category.*

Category-theoretic studying causality relationships was initiated in the papers [7,8].

[3] See, for example, [12]

## 6 Conclusion

This paper presents the formal approach to study causality relationships in distributed systems.

In the paper, the basic formal theory developed with Coq Proof Assistant has been described.

Further research and development may be focused on

- the formalisation with Coq Proof Assistant of finiteness conditions to emphasise the subclass of discrete event universes;
- the identification of states related to events to try to construct a natural coalgebra for specifying the behaviour of systems being studied;
- the identification of logical clocks for obtaining some complete and usable causality constraint specification language.

## References


1. Bertot, Y., Castéran, P.: Interactive Theorem Proving and Program Development. Springer-Verlag Berlin Heidelberg (2004)
2. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Communications of the ACM 21(7), 558–565 (1978)
3. Winskel, G.: Event structures. In: Brauer, W., Reisig, W., Rozenberg, G. (eds.) Petri Nets: Applications and Relationships to Other Models of Concurrency. LNCS, vol. 255, pp. 325–392 (1986)
4. André, C., Mallet, F.: UML/MARTE CCSL, Signal and Petri nets. Research Report RR-6545, INRIA (2008)
5. André, C., Mallet, F., De Simone, R.: Logical time: specification vs. implementation. ACM SIGSOFT Software Engineering Notes 36(1), 1–8 (2011)
6. Mallet, F.: MARTE/CCSL for modeling cyber-physical systems. In: Drechsler, R., Kühne, U. (eds.) Formal Modeling and Verification of Cyber-Physical Systems. Springer Vieweg, Wiesbaden (2015)
7. Zholtkevych, G., El Zein, H.: Two Approaches to Modelling Logical Time in Cyber-Physical Systems. In: Basliades, N., et al. (eds.) ICT in Education, Research, and Industrial Applications, CCIS, vol. 826, pp. 21–40. Spinger International Publishing AG (2018)
8. Zholtkevych, G., El Zein, H., Polyakova, L.: Category Methods for Modelling Logical Time Based on the Concept of Clocks. In: Ermolayev, V., et al. (eds.) ICT in Education, Research, and Industrial Applications, CCIS, vol. 1007, pp. 89–101. Spinger International Publishing AG (2019)
9. The Coq Development Team: Type Classes. In: The Coq Reference Manual. INRIA, 8.9.0 edn. (2019), https://coq.inria.fr/distrib/current/refman/index.html, (accessed 2.02.2019)
10. Lampropoulos, L., Pierce, P.C.: QuickChick: Property-Based Testing in Coq, Software Foundations, vol. 4 (2018), https://softwarefoundations.cis.upenn.edu/qc-current/index.html, (accessed 2.02.2019)
11. The Coq Development Team: The Gallina specification language. In: The Coq Reference Manual. INRIA, 8.9.0 edn. (2019), https://coq.inria.fr/distrib/current/refman/index.html, (accessed 2.02.2019)
12. Awodey, S.: Category Theory, Oxford Logic Guides, vol. 52. Oxford University Press, 2nd edn. (2010)