# Proof of the Instrumented Semantics for Orc

Matthieu Perrin, Claude Jard, Achour Mostefaoui

# Proof of the Instrumented Semantics for Orc

Matthieu Perrin
LINA, University of Nantes
*matthieu.perrin@univ-nantes.fr*

Claude Jard
LINA, University of Nantes
*claude.jard@univ-nantes.fr*

Achour Mostéfaoui
LINA, University of Nantes
*achour.mostefaoui@univ-nantes.fr*

**Abstract**

This report shows how the operational semantics of a language like ORC can be instrumented so that the execution of a program produces information on the causal dependencies between events. The concurrent semantics we obtain is based on labeled asymetric event structures. This report contains the complete demonstration of correctness of this approach.

**Key words:** Preemption, Conflict, Concurrency, Orc, Semantics, Causality, Asymmetric event structure

## 1 Introduction

The standard form for describing the semantics of a computer language is the structural operational semantics (SOS) [3]. SOS specifications take the form of a set of inference rules that define the valid transitions of a composite piece of syntax in terms of the transitions of its components. Rewriting transforms terms by executing a rule (it may be a non-deterministic transition in case of multiple alternatives). The successive transitions represent the program behavior. This may produce a sequence of values, that can be brought by the labelling of rules. In this vision, the behaviors are therefore defined sequentially, even for languages describing parallel behaviors.

This vision is consequently limited for modern languages comprising parallelism. Indeed, in these languages, the concepts of concurrency, causality and conflict are important to explicit (for example, to answer questions on root causes analysis, debugging&replay, race detection, QoS assessment, etc.). This is the role of a concurrent semantics. In such semantics, behaviors are seen as partial orders (encoding the causality between the events produced by a given program), optionally augmented by the notion of conflict identifying non-deterministic choices that have been made (e.g. event structures [6]).

The method used in this article is to extend the standard SOS semantics to rewrite extended expressions, in which additional information has been added to compute causal and weakly-causal dependencies. This information is also made visible by extending the labelling of transitions. Concurrency is just the complement of the weakly-causal relation, and conflict is defined by cycles in this relation.

Capturing causality and concurrency by instrumenting the semantics rules is a difficult job. This is mainly because these relationships are global and are difficult to locate on the syntactic forms. The solution is to keep information about the causal past in a context associated with each rule being careful to build the necessary links between different contexts during the execution of rules. The aim is that such instrumented semantics reproduces the standard behavior of the program while calculating the additional information needed to find concurrency, causality and conflict between events produced by the execution.

The proposed approach is illustrated on the Orc language [2]. Orc is a novel language, formally defined and specially adapted to program across the Web in a highly parallel and distributed context. A first attempt of concurrent semantics based on event structures has already been published [4], but using an ad-hoc connection of Petri net diagrams. A similar approach was used in [1] to compare Orc with the Join calculus. Here, the approach appears to us more simple and elegant. It is especially dynamic in the sense that it can produce online information, interesting feature in practice for many above-mentioned applications. To our knowledge, the only attempt to use an instrumentation to track

causality online in a multithreaded program is [5] for the C language, but the instrumentation is made by a program transformation, which excludes the possibility to explore further than causality.

Our contribution in this article consists in two new semantics for the Orc language. The first one is an instrumentation of the standard semantics that preserves the behavior of Orc while tracking causal and weakly-causal relations between the events. The second semantics, the concurrent semantics, is an extension of the instrumented semantics that generates all possible events for an Orc expression, so every execution represents all possible sequential executions of the program. These two semantics are complementary, as the first one is lighter and respects the standard notion of execution, while the second one provides more information.

The reminder of the article is organized as follows. We start with a presentation of the Orc language in Section 2. In Section 3, we introduce the labelled asymmetric event structures, used to represent concurrent executions. Section 4 and 5 detail our two contributions: respectively the instrumented and the concurrent semantics. The proof of there properties are are given in Section 6. Finally, section 7 discusses how to combine the two semantics and concludes the paper.

# 2 The Orc Programming Language

## 2.1 Wide-area computing

The Orc language was initially designed to orchestrate web sites. Its underlying philosophy is that modularity is essential to the construction of large modern programs. Such programs may be composed out of several components, possibly written in a variety of languages, that cannot work together without a conductor that orchestrates their executions. In Orc, these components, called *sites*, can be any kind of externally defined service as well as encapsulations of Orc expressions. Unlike functions in functional programming languages, Orc sites cannot return a result. Instead, they publish zero, one or more values. Sites can be higher-ordered, so values can be sites themselves. This allows the use of complex shared data types; each instance of which is viewed as a different site. A standard library of sites is defined to handle many aspects of everyday programming such as complex data types features, clocks and timers to define some synchrony in programs. How the sites interact is the heart of the Orc language. To this end, it provides the developers with a series of control structures. Some of them, like the conditional structure, are very classical, while others, designed to express concurrency, are more original. These are formally defined in a mathematical abstract programming language: the Orc core calculus, from which the full language is defined as syntactic sugar.

## 2.2 Core calculus

As the higher-level constructions of the language are defined from the Orc core calculus, we only consider this well-defined calculus in the remaining of this paper. The expressions of the calculus, grouped in the set $Orc_s$, is defined as follows:

$$
\begin{array}{lcl}
f, g, h \in \text{Expression} & ::= & p \| p(p) \| ?k \| f | g \| f >x> g \\
 & & \| f <x< g \| f; g \| D \# f \| \bot \\
D \in \text{Definition} & ::= & \textbf{def}\ y(x) = f \\
v \in \text{Orc Value} & ::= & V \| D \\
p \in \text{Parameter} & ::= & v \| \textbf{stop} \| x \\
w \in \text{Response} & ::= & NT(v) \| T(v) \| Neg \\
n \in \text{Hidden Label} & ::= & ?V_k(v) \| ?D \| h(\omega) \| h(!v) \\
l \in \text{Label} & ::= & !v \| n \| \omega
\end{array}
$$

The expressions of the calculus that correspond to real Orc programs, in the set $Orc$, are those that do not contain $?k$ and $\bot$ expressions. Here are the rules of the standard semantics :

$$\text{(PUBLISH)}\ \frac{}{v \xrightarrow{!v} \textbf{stop}}\ v \text{ closed}$$

$$\text{(DEFDECLARE)}\ \frac{[D/y]f \xrightarrow{l} f'}{D \# f \xrightarrow{l} f'}\ D \text{ is } \textbf{def}\ y(x) = g$$

$$\text{(STOP)}\ \frac{}{\textbf{stop} \xrightarrow{\omega} \bot}$$

$$\text{(IntCall)}\ \frac{}{D(p) \xrightarrow{?D} [D/y][p/x]g}\ D \text{ is } \mathbf{def}\ y(x) = g$$

$$\text{(StopCall)}\ \frac{}{\mathbf{stop}(p) \xrightarrow{\omega} \bot}$$

$$\text{(ExtCall)}\ \frac{}{V(v) \xrightarrow{?V_k(v)} ?k}\ k \text{ fresh}$$

$$\text{(ExtStop)}\ \frac{}{V(\mathbf{stop}) \xrightarrow{\omega} \bot}$$

$$\text{(OtherN)}\ \frac{f \xrightarrow{n} f'}{f; g \xrightarrow{n} f'; g}$$

$$\text{(OtherV)}\ \frac{f \xrightarrow{!v} f'}{f; g \xrightarrow{!v} f'}$$

$$\text{(OtherStop)}\ \frac{f \xrightarrow{\omega} \bot}{f; g \xrightarrow{h(\omega)} g}$$

$$\text{(NtRes)}\ \frac{?k \text{ receives } NT(v)}{?k \xrightarrow{!v} ?k}$$

$$\text{(TRes)}\ \frac{?k \text{ receives } T(v)}{?k \xrightarrow{!v} \mathbf{stop}}$$

$$\text{(NegRes)}\ \frac{?k \text{ receives } Neg}{?k \xrightarrow{\omega} \bot}$$

$$\text{(SeqN)}\ \frac{f \xrightarrow{n} f'}{f >x> g \xrightarrow{n} f' >x> g}$$

$$\text{(SeqV)}\ \frac{f \xrightarrow{!v} f'}{f >x> g \xrightarrow{h(!v)} (f' >x> g)|[v/x]g}$$

$$\text{(SeqStop)}\ \frac{f \xrightarrow{\omega} \bot}{f >x> g \xrightarrow{\omega} \bot}$$

$$\text{(ParLeft)}\ \frac{f \xrightarrow{l} f'}{f|g \xrightarrow{l} f'|g}\ l \neq \omega$$

$$\text{(ParRight)}\ \frac{g \xrightarrow{l} g'}{f|g \xrightarrow{l} f|g'}\ l \neq \omega$$

$$\text{(ParStop)}\ \frac{f \xrightarrow{\omega} \bot \quad g \xrightarrow{\omega} \bot}{f|g \xrightarrow{\omega} \bot}$$

$$\text{(PruneN)}\ \frac{g \xrightarrow{n} g'}{f <x< g \xrightarrow{n} f <x< g'}$$

$$\text{(PruneLeft)}\ \frac{f \xrightarrow{l} f'}{f <x< g \xrightarrow{l} f' <x< g}\ l \neq \omega$$

$$\text{(PruneV)}\ \frac{g \xrightarrow{!v} g'}{f <x< g \xrightarrow{h(!v)} [v/x]f}$$

$$\text{(PruneStop)}\ \frac{g \xrightarrow{\omega} \bot}{f <x< g \xrightarrow{h(\omega)} [\mathbf{stop}/x]f}$$

The only values considered in the Orc core calculus are sites. They can be external, which is denoted $V$ in the syntax, or internally defined ($D$). For the sake of clarity, we consider in this work that the sites are curryfied. An internal site can be defined as $\mathbf{def}\ y(x) = f\#g$ where $f$ is the body of the

site and $g$ is the remaining of the program in which $y$ can be used as any site. Site definitions are recursive, which allows the same expressivity as any functional language. Both kinds of sites can be called with their argument using the classical functional notation. However, their behaviors are different, as we can see on the operational semantics. The main difference is that calls to external sites are strict, i.e. their arguments have to be bounded before the site can be called, while an internal site can be called immediately, and its arguments are evaluated lazily. When an external site is called, it sends its responses to a placeholder $?k$. A response can be either a non-terminating value $NT(v)$ if further responses are expected, or a terminating value $T(v)$ if this is the last publication of the site or $Neg$ if the site terminates without publishing any value.

Besides sites, four combinators are provided by core calculus. The parallel composition expresses pure concurrency. In $f|g$, $f$ and $g$ are run in parallel, their events are interleaved and the expression stops when both $f$ and $g$ have terminated. As suggested by its name, the sequential operator expresses sequentiality. In the expression $f >x> g$, the variable $x$ can be used in $g$. Here, $f$ is started first, and then a new instance of $g[v/x]$, where $x$ is bounded to $v$, is launched as a consequence of each publication of $v$. The third operator, called pruning, expresses preemption. In $f <x< g$, the variable $x$ can be used in $f$. Both $f$ and $g$ are started at once, but $f$ is paused when it needs to evaluate $x$. When $g$ publishes a value, it is bounded to $x$ in $f$ and $g$ is stopped. The other events that could have been produced by $g$ are preempted by the publication. For example, if $g$ is supposed to publish two values $a$ and $b$, only one will be selected and published in each execution. We say that these two events are in conflict. The last operator is called otherwise. In $f; g$, $f$ is first started alone and $g$ is started if and only if $f$ stops without publishing any value.

Finally, the **stop** symbol can be used by the programmer exactly like a site or a variable to denote a terminated program. **stop** still produces an event $\omega$ to notify its parent expression that it has terminated. It then evolves into $\perp$, the inert final expression. Just like $?k$, $\perp$ cannot be used directly.

## 2.3 Illustration

We will now illustrate the Orc core calculus on an example. Let us consider a program $f_0$ defined as

$$f_0 = y + z <y< ((2|3) >x> x) <z< 1.$$

The left hand side causally depends on $y$ and $z$, its execution is thus delayed. Two conflictual values are possible for $y$. In this example, both 2 and 3 can be published concurrently in the left hand side of the sequential operator, but when a value is published for the second time, it is bounded to $y$ and the remainder of the execution is preempted. Finally, the value of $z$ is defined by the publication of a constant value, the execution of which is concurrent to the computation of $y$. The following is a possible execution.

$$\begin{array}{rcl} f_0 & \xrightarrow{h(2)} & y + z <y< ((\mathbf{stop}|3) >x> x|2) <z< 1 \\ & \xrightarrow{h(1)} & y + 1 <y< ((\mathbf{stop}|3) >x> x|2) \\ & \xrightarrow{h(3)} & y + 1 <y< ((\mathbf{stop}|\mathbf{stop}) >x> x|3|2) \\ & \xrightarrow{h(3)} & 3 + 1 \xrightarrow{?+(3,1)} ?k \xrightarrow{!4} \mathbf{stop} \xrightarrow{\omega} \perp \end{array}$$

Many executions are possible for the standard semantics. A few of them are given below.

$$\left\{ \begin{array}{c} h(!1).h(!3).h(!3).? + (3,1).!4.\omega, \\ h(!3).h(!1).h(!3).? + (3,1).!4.\omega, \\ h(!3).h(!3).h(!1).? + (3,1).!4.\omega, \\ h(!2).h(!1).h(!3).h(!3).? + (3,1).!4.\omega, \\ h(!1).h(!2).h(!3).h(!2).? + (2,1).!3.\omega, \\ h(!2).h(!1).h(!3).h(\omega).h(!3).? + (3,1).!4.\omega, \\ h(!2).h(!1).h(!3).h(\omega), \\ \dots \end{array} \right\} \subset [\![f_0]\!]$$

It is possible to do some observation on these executions. For example, all the executions that contain the event $!4$ also contain one event $h(!1)$ and two instances of $h(!3)$, that appear before the publication $!4$. This hides a relation of causality between these events. Moreover, no execution contains both the events $? + (2,1)$ and $? + (3,1)$, which denotes conflict.

However, it is very difficult to use these observations in practice for two reasons. First, the number of executions is too large, and even infinite very often, to test even simple properties. Secondly, we lost all the information of how to identify events from different executions. We cannot say that the two instances of $h(!3)$ are in the same order in the two first executions, so we can say nothing about there

causal dependencies. This is why it is necessary to aggregate the information of many executions into a unique object.

# 3 Labelled Asymmetric Event Structure

The explosion of the number of possible executions is due to their representation as sequences of events: parallel executions are interleaved, creating an exponential number of possibilities. A natural objects to represent concurrent events in a compact way are labelled asymmetric event structures (LAES).

**Definition 1** (Labelled asymmetric event structure)**.** *A* labelled asymmetric event structure *(LAES) is a tuple* $(E, L, \leq, \nearrow, \Lambda)$.

- $E$ *is a set of* events,
- $L$ *is a set of* labels,
- $\leq$, causality *is a partial order on* $E$,
- $\nearrow$, weak causality *is a binary relation on* $E$,
- $\Lambda : E \mapsto L$ *is the* labelling function,
- *each* $e \in E$ *has a finite* causal history $[e] = \{e' \in E | e' \leq e\}$,
- *for all events* $e < e' \in E, e \nearrow e'$, *where* $<$ *is the irreflexive restriction of* $\leq$,
- *for all* $e \in E$, $\nearrow \cap [e]^2$ *is acyclic.*

A LAES can encode many relations between events in concurrent systems. Let $(E, L, \leq, \nearrow, \Lambda)$ be an asymmetric event structure and $e, e' \in E$ two events. We say that:

- $e$ is a *cause* of $e'$, if $e$ happens before $e'$ in all executions;
- $e$ is a *weak cause* of $e'$, if there is no execution in which $e$ happens after $e'$;
- $e$ and $e'$ are *concurrent*, denoted $e||e'$, if they can occur in either order. Formally, $e||e'$ if neither $e \nearrow e'$ nor $e' \nearrow e$.
- $e$ is *preempted* by $e'$, denoted $e \rightsquigarrow e'$, if $e'$ can occur independently from $e$, but after that, $e$ cannot occur anymore. Formally, $e \rightsquigarrow e'$ if $e \nearrow e'$ and $e \not\leqslant e'$,

We also define an induced *conflict* relation $\#_a$ as the smallest set of finite parts of $E$ such that: for $E' \subset E$ and $e_0, e_1, ..., e_n \in E$,

- if $e_0 \nearrow e_1 \nearrow \cdots \nearrow e_n \nearrow e_0$ then $\{e_0, e_1, \cdots, e_n\} \in \#_a$
- if $E' \cup \{e_0\} \in \#_a$ and $e_0 \leq e_1$ then $E' \cup \{e_1\} \in \#_a$.

Informally, two events are conflictual if they cannot occur together in the same execution.

A LAES can be seen as a structure that encodes concisely several sequential executions; each of them being a linearization of the LAES.

**Definition 2** (Linearization)**.** *Let* $\mathcal{E} = (E, L, \leq, \nearrow, \Lambda)$ *be a LAES. A* finite linearization *of* $\mathcal{E}$ *is a word* $w = \Lambda(e_0) \dots \Lambda(e_n)$ *where the different* $e_i \in E$ *are distinct and such that:*

- *it is left-closed for causality:*

$$\forall e \in E, \forall e' \in \{e_0, \dots, e_n\}, e \leq e' \Rightarrow e \in \{e_0, \dots, e_n\},$$

- *the weak causality is respected:*

$$\forall e_i, e_j \in \{e_0, \dots, e_n\}, e_i \nearrow e_j \Rightarrow i < j$$

*We denote* $Lin(\mathcal{E})$ *as the set of all finite linearizations of* $\mathcal{E}$.

The last definition allows to compare two LAESs. There is an injection between two LAESs if one contains at least the information of the other one. In that case, there is an inclusion between the set of linearizations. Two LAESs are isomorphic if they encode the same information.

**Definition 3** (Injection)**.** *Let* $\mathcal{E}_1 = (E_1, L_1, \leq_1, \nearrow_1, \Lambda_1)$ *and* $\mathcal{E}_2 = (E_2, L_2, \leq_2, \nearrow_2, \Lambda_2)$ *be two LAESs. An injection from* $\mathcal{E}_1$ *to* $\mathcal{E}_2$ *is an injective function* $f : E_1 \mapsto E_2$ *such that:*

- $\forall e \in \mathcal{E}_1, \Lambda_1(e) = \Lambda_2(f(e))$,
- $\forall e \leq_1 e' \in \mathcal{E}_1, f(e) \leq_2 f(e')$,
- $\forall e \nearrow_1 e' \in \mathcal{E}_1, f(e) \nearrow_2 f(e')$.
- $\forall e_2 \leq_2 f(e) \in \mathcal{E}_2, \exists e_1 \in \mathcal{E}_1, e_2 = f(e_1)$.

*We write* $\mathcal{E}_1 < \mathcal{E}_2$ *if there is an injection from* $\mathcal{E}_1$ *to* $\mathcal{E}_2$ *and we say that* $\mathcal{E}_1$ *and* $\mathcal{E}_2$ *are isomorphic, denoted* $\mathcal{E}_1 \equiv \mathcal{E}_2$, *if* $\mathcal{E}_1 < \mathcal{E}_2$ *and* $\mathcal{E}_2 < \mathcal{E}_1$.

# 4 Instrumented Semantics

## 4.1 Method

The principle of an operational semantics is to produce the set of events that may occur during an execution and to publish them as transition labels. Total ordering of events is not relevant for concurrent programs. Our approach does not affect the successive enforcement of the rules, but adds additional information in the labels that allows to track the partial order.

Actually, a label in the instrumented semantics is a tuple $e = (e_k, e_l, e_c, e_a)$, where $e_k$ is an identifier taken in a countable set $K$, that is unique for the execution, $e_l$ is a label similar to those of the standard semantics and $e_c$ and $e_a$ contain the finite sets of the identifiers of the causes and the week causes of the event, respectively.

In order to record the information concerning the past of an expression, we enrich the language with a new syntactic construction: $\langle f, c, a\rangle_L$ means that $c$ and $a$ are the causes of the Orc instrumented expression $f$. Thus, if $f$ has $c$ and $a$ as causes and if it can evolve into $f'$, this transition should also have $c$ and $a$ as causes. The index $L$ expresses the kind of events that can activate the rule: $!v$ matches any publication, $l$ stands for any label and $\omega$ means that $c$ and $a$ are only the causes of the termination of the program. This new construction is formally defined by two new rules in the semantics: CAUSEYES and CAUSENO.

We also consider that the external sites track themselves causality, as an internally-defined function would do. It makes sense as some sites (e.g. $+$) handle their calls independently, while others (e.g. shared registers, management library) induce more complex causality patterns between the calls. Hence, the responses we get include this additional information. The verification of these responses is not the subject here, and we suppose them correct by hypothesis.

The instrumented semantics is defined on the set $Orc_i$ of expressions of this extended syntax:

$$
\begin{array}{lll}
f, g, h \in \text{Expression} & ::= & p \| p(p) \| ?k \| f|g \\
 & & \| f >x> g \| f <x< g \| f; g \\
 & & \| D\#f \| \bot \| \langle f, K, K\rangle_L \\
D \in \text{Definition} & ::= & \textbf{def}\ y(x) = f \\
v \in \text{Orc Value} & ::= & V \| D \\
p \in \text{Parameter} & ::= & v \| \textbf{stop} \| x \| \langle p, K, K\rangle_L \\
w \in \text{Response} & ::= & NT(v, K, K) \| T(v, K, K) \\
 & & \| Neg(K, K) \\
n \in \text{Hidden Label} & ::= & ?V_k(v) \| ?D \| h(\omega) \| h(!v) \\
l \in \text{Label} & ::= & !v \| n \| \omega
\end{array}
$$

## 4.2 Rules

The rules of the instrumented semantics are given Thereafter.

$$\text{(PUBLISH)}\ \frac{}{v \xrightarrow{k,!v,\emptyset,\emptyset}_i \langle \textbf{stop}, \{k\}, \emptyset\rangle_l}\quad \begin{array}{l} v \text{ closed} \\ k \text{ fresh}\end{array}$$

$$\text{(STOP)}\ \frac{}{\textbf{stop} \xrightarrow{k,\omega,\emptyset,\emptyset}_i \bot}\quad k \text{ fresh}$$

$$\text{(DEFDECLARE)}\ \frac{[D/y]f \xrightarrow{k,l,c,a}_i f'}{D\#f \xrightarrow{k,l,c,a}_i f'}\quad D \text{ is } \textbf{def}\ y(x) = g$$

$$\text{(STOPCALL)}\ \frac{P \xrightarrow{k,\omega,c,a}_i \bot}{P(p) \xrightarrow{k,\omega,c,a}_i \bot}$$

$$\text{(INTCALL)}\ \frac{P \xrightarrow{k,!D,c,a}_i P'}{P(p) \xrightarrow{k,?D,c,a}_i \langle [D/y][p/x]g, c \cup \{k\}, a\rangle_l}\quad D \text{ is } \textbf{def}\ y(x) = g$$

$$\text{(EXTCALL)}\ \frac{P \xrightarrow{k,!V,c,a}_i P' \qquad p \xrightarrow{k',!v,c',a'}_i p'}{P(p) \xrightarrow{k,?V_k(v),c\cup c',a\cup a'}_i \langle ?k, c \cup c' \cup \{k\}, a \cup a'\rangle_l}$$

$$\text{(EXTSTOP)}\ \frac{P \xrightarrow{k,!V,c,a}_i P' \qquad p \xrightarrow{k',\omega,c',a'}_i p'}{P(p) \xrightarrow{k,\omega,c\cup c',a\cup a'}_i \bot}$$

$$\text{(OtherN)}\ \frac{f \xrightarrow{k,n,c,a}_i f'}{f;g \xrightarrow{k,n,c,a}_i f';g}$$

$$\text{(ResNt)}\ \frac{?k \text{ receives } NT(v,c,a)}{?k \xrightarrow{j,!v,c,a\cup c}_i ?k}\ j \text{ fresh}$$

$$\text{(OtherV)}\ \frac{f \xrightarrow{k,!v,c,a}_i f'}{f;g \xrightarrow{k,!v,c,a}_i f'}$$

$$\text{(ResT)}\ \frac{?k \text{ receives } T(v,c,a)}{?k \xrightarrow{j,!v,c,a\cup c}_i \langle \mathbf{stop}, c\cup\{j\}, a\rangle_\omega}\ j \text{ fresh}$$

$$\text{(OtherStop)}\ \frac{f \xrightarrow{k,\omega,c,a}_i \bot}{f;g \xrightarrow{k,h(\omega),c,a}_i \langle g, c\cup\{k\}, a\rangle_l}$$

$$\text{(ResNeg)}\ \frac{?k \text{ receives } Neg(c,a)}{?k \xrightarrow{j,\omega,c,a\cup c}_i \bot}\ j \text{ fresh}$$

$$\text{(SeqN)}\ \frac{f \xrightarrow{k,n,c,a}_i f'}{f >x> g \xrightarrow{k,n,c,a}_i f' >x> g}$$

$$\text{(ParLeft)}\ \frac{f \xrightarrow{k,l,c,a}_i f'}{f|g \xrightarrow{k,l,c,a}_i f'|g}\ l \neq \omega$$

$$\text{(SeqV)}\ \frac{f \xrightarrow{k,!v,c,a}_i f'}{f >x> g \xrightarrow{k,h(!v),c,a}_i (f' >x> g)|\langle [v/x]g, c\cup\{k\}, a\rangle_l}$$

$$\text{(ParRight)}\ \frac{g \xrightarrow{k,l,c,a}_i g'}{f|g \xrightarrow{k,l,c,a}_i f|g'}\ l \neq \omega$$

$$\text{(SeqStop)}\ \frac{f \xrightarrow{k,\omega,c,a}_i \bot}{f >x> g \xrightarrow{k,\omega,c,a}_i \bot}$$

$$\text{(ParStop)}\ \frac{f \xrightarrow{k,\omega,c,a}_i f' \qquad g \xrightarrow{k',\omega,c',a'}_i g'}{f|g \xrightarrow{k,\omega,c\cup c',a\cup a'}_i \bot}$$

$$\text{(PruneN)}\ \frac{g \xrightarrow{k,n,c,a}_i g'}{f <x< g \xrightarrow{k,n,c,a}_i f <x< \langle g', \emptyset, \{k\}\rangle_{!v}}$$

$$\text{(PruneLeft)}\ \frac{f \xrightarrow{k,l,c,a}_i f'}{f <x< g \xrightarrow{k,l,c,a}_i f' <x< g}\ l \neq \omega$$

$$\text{(PruneV)}\ \frac{g \xrightarrow{k,!v,c,a}_i g'}{f <x< g \xrightarrow{k,h(!v),c,a}_i \langle [\langle v, c\cup\{k\}, a\rangle_l/x]f, c\cup\{k\}, a\rangle_\omega}$$

$$\text{(PruneStop)}\ \frac{g \xrightarrow{k,\omega,c,a}_i \bot}{f <x< g \xrightarrow{k,h(\omega),c,a}_i \langle [\langle \mathbf{stop}, c\cup\{k\}, a\rangle_l/x]f, c\cup\{k\}, a\rangle_\omega}$$

$$\text{(CauseYes)}\ \frac{f \xrightarrow{k,l,c,a}_i f'}{\langle f, c', a'\rangle_L \xrightarrow{k,l,c\cup c',a\cup a'\cup c'}_i \langle f', c', a'\rangle_L}\ l \in L$$

$$\text{(CauseNo)}\ \frac{f \xrightarrow{k,l,c,a}_i f'}{\langle f, c', a'\rangle_L \xrightarrow{k,l,c,a}_i \langle f', c', a'\rangle_L}\ l \notin L$$

Let us comment the more relevant ones.

Let us consider the rule SeqV. When a value is published, a new instance of the right hand side expression is created. All the events produced by this new expression need the former publication to *have occured before* them, i.e. they are consequences of this publication. This is why the new expression is instrumented.

Even if PRUNEV is syntactically very similar to SEQV, the fact that both hand sides of the pruning operator are run in parallel makes them very different in terms of causality. In the expression $(1|x) < x < 2$, the second publication of 2 is a consequence of the first one, but not the publication of 1. This is why the instrumentation covers the occurrences of the newly bounded variable. However, this is not sufficient. Let us consider the program $(\mathbf{stop} < x < 2); 3$. The publication of 3 must wait the end of the left hand side (i.e the publication of 2). However, this publication is useless, in the sense that no variable $x$ can be bounded to its value. To handle this case, we add an instrumentation to the whole expression that is only triggered when the expression stops. Note that this instrumentation would not have been needed if another rule allowed to terminate the expression just by knowing that **stop** would never produce a value. This means that the instrumentation of a rule does not only depend on this rule, but also on the other rules of the semantics.

Finally, the rule PRUNEN is also interesting as it generates weak causality. Indeed, in the program $x < s < (1 + 1|3)$, the left hand side can call site $+$ and then publish 3, or publish 3 directly, but can never publish 3 and then call site $+$, because a publication preempts any other event. Of course, it could also wait for the answer of the site and then publish 2, which would preempt the publication of 3. This preemption relation is operated by an instrumentation that contains $k$ as weak causes and that is triggered only in case of publication.

## 4.3 Concurrent executions

Like all operational semantics, this set of rules defines a transition system $\rightarrow_i$ and a sequential semantics $[\![\cdot]\!]_i$. In order to capture concurrency, we want to express an execution as a LAES, hence the following definition of *concurrent executions*.

**Definition 4** (Concurrent execution)**.** *Let $\sigma = \sigma^1 \dots \sigma^n \in [\![f_0]\!]_i$. We define the* concurrent execution *of $\sigma$ as the tuple*

$$\overline{\overline{\sigma}} = (\{\sigma^1_k, \dots, \sigma^n_k\}, \{\sigma^1_l, \dots, \sigma^n_l\}, \leq, \nearrow, \Lambda)$$

*where for all $i, j \in \{1, \dots, n\}$:*

- $\sigma^i_k \leq \sigma^j_k$ *if* $\sigma^i_k \in \sigma^j_c$ *or* $i = j$,
- $\sigma^i_k \nearrow \sigma^j_k$ *if* $\sigma^i_k \in \sigma^j_a$,
- $\Lambda(\sigma^i_k) = \sigma^i_l$.

## 4.4 Correctness

In this section, we prove the main result: the behavior of a program is preserved by the instrumented semantics. It is established through two properties. The first one justifies the name of the instrumented semantics and the second one proves that the instrumentation is correct, i.e. that it does not define incorrect behaviors.

**Property 1** (Instrumentation)**.** *Let*

$$([\![f]\!]_i)\,|_l = \{\sigma|_l = \sigma[1]_l ... \sigma[n]_l | \sigma \in [\![f]\!]_i\}$$

*be the set of projections of the executions by the instrumented semantics on their labels. We have:*

$$\forall f \in \mathcal{O}, ([\![f]\!]_i)\,|_l = [\![f]\!].$$

In other words, it is always possible to instrument a standard execution to get a concurrent execution, and conversely we can get a standard execution from an instrumented execution by a simple projection.

**Theorem 1** (Correctness)**.** *The behaviors that can be observed from an execution in the instrumented semantics are correct with respect to the standard semantics:*

$$\forall f \in Orc, \forall \sigma_i \in [\![f]\!]_i, Lin(\overline{\overline{\sigma_i}}) \subset [\![f]\!].$$

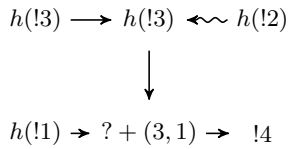
## 4.5 Illustration

To illustrate the behavior of the instrumented semantics, we reuse the example $f_0 = y + z < y < ((2|3) > x > x) < z < 1$. The instrumented version of the execution presented in section 2.3 is labelled

by the sequence:

$$(k_1, h(2), \emptyset, \emptyset).\ (k_2, h(1), \emptyset, \emptyset).\ (k_3, h(3), \emptyset, \emptyset).$$
$$(k_4, h(3), \{k_3\}, \{k_1, k_3\}).$$
$$(k_5, ? + (3,1), \{k_2, k_3, k_4\}, \{k_2, k_3, k_4\}).$$
$$(k_6, !4, \{k_2, k_3, k_4, k_5\}, \{k_2, k_3, k_4, k_5\}).$$
$$(k_7, \omega, \{k_2, k_3, k_4, k_5, k_6\}, \{k_2, k_3, k_4, k_5, k_6\}).$$

We represent the LAES of this execution thereafter. The events of the execution are depicted by their labels, that are not unique (two events are denoted $h(!3)$). The solid arrows ($\rightarrow$) represent direct causality, i.e. the smallest relation that has $\leq$ as transitive and reflexive closure. The twisty arrows ($\rightsquigarrow$) denote passive direct preemption. By definition, $a$ is preempted by $b$ if they resp. have $a'$ and $b'$ as causes and those are related by a twisty arrow.




We observe that the events labelled $h(!3)$ and $h(!2)$, that correspond to the computation of $y$, are concurrent with the event labelled $h(!1)$, and the first publication of 3 is concurrent with the publication of 2, which corresponds to our analysis. Moreover, the second event $h(!3)$ preempts the occurrence of $h(!2)$, which explains why $h(!3).h(!1).h(!3).h(!2).? + (3,1).!4.\omega$ is invalid.

There is no conflict in this execution. This is due to the fact that the instrumented semantics is based on the same set of executions as the standard semantics, in which there cannot be conflictual events.

# 5 Concurrent Semantics

## 5.1 Method

By introducing concurrency and preemption between events that were arbitrarily ordered by the standard semantics, the instrumented semantics heavily reduces the number of different executions. However, there may remain as many others (because of preemption) which are difficult to link in a reasoning. For this reason, the instrumented semantics is not well suited for studying race conditions for example.

Our strategy to add information about conflicts is to keep firing transitions even if they are preempted by a previous event. Thereby, it becomes possible to observe conflictual events in the same execution. The difficulty is to adapt the rest of the program so that it can also generate recursively the consequences of these conflictual events. Let us illustrate this problem on the program $(1 >x> x + y) <y< 2|3$. In the concurrent semantics, the publication of 2 and 3 will both occur even though they are conflictual. The variable $y$ will then be bounded to 2 values. Note that it is not possible to duplicate the whole expression at the left hand side of the pruning operator because the publication of 1 is a single event concurrent to the computation of $y$.

To circumvent this difficulty, we propose a new way to bound variables. Instead of replacing the free occurrences of a $x$ by $p$ in $f$, $[p//x]f$ adds $p$ in a bag maintained by the free occurrences of $x$ in $f$. Specifically, $[p//x]x = x_{\{p\}}$ and $[p//x]x_P = x_{P\cup\{p\}}$. If $p$ can evolve into $p'$, $x_{P\cup\{p\}}$ can evolve into $x_{P\cup\{p'\}}$.

In the standard semantics, an expression terminates when it is preempted in the pruning operator or when it produces an $\omega$ event. In the concurrent semantics, preempted expressions are not stopped and an expression may produce several conflictual $\omega$ events. It is then difficult — and not necessary — to know when an expression terminates and inactive sub-expressions cannot be removed.

It is yet still necessary to generate $\omega$ events, for example to know when the right hand side of the otherwise operator should be started. Let us consider again the program $(\mathbf{stop} <x< 1|2|3)|(\mathbf{stop} <y< 4|5|6)$. In the standard semantics, the program terminates by a unique $\omega$ event when two publications are caught and ignored. In the concurrent semantics, each side of the main parallel operator produces three conflictual $\omega$ events. The global expression must produce nine conflictual $\omega$ events due to the distributivity of their causes. We introduce a new distributivity structure, that is a pair $(K_L, K_R)$ of sets of pairs $(e, L)$ where $e$ is an event and $L$ the set of the events that must be distributed on $e$. We introduce the notation $e \triangleright (K_L, K_R)$ meaning that the event $e$ happened on the left hand side. It is distributed on all of the events that have already happened on the right hand side and it gets an entry

on the left hand side to the future events. A similar notation exists for the symmetric case. More formally:

$$\begin{array}{rcl} e \otimes \emptyset & = & \emptyset \\ e \otimes (\{(e', L)\} \cup K') & = & \{(e', L \cup \{e\})\} \cup (e \otimes K') \\ e \triangleright (K_L, K_R) & = & (\{(e, \emptyset)\} \cup K_L, e \otimes K_R) \\ (K_L, K_R) \triangleleft e & = & (e \otimes K_L, \{(e, \emptyset)\} \cup K_R) \end{array}$$

A distributivity structure is used with the parallel operator as well as the pruning operator to manage the stopping of expressions. This is done in three steps. Both sides generate $\omega$ events when they are ready; these are stored into the distributivity structure before the whole expression can generate an $\omega$ event. The two first events have a new empty label $\varepsilon$ that is just ignored as an event.

There is no possible conflict in $x <x< 1$, so we want to produce the same execution as in the instrumented semantics. In particular, the right hand side of the pruning operator never produces an $\omega$ event. More generally, the consequences of a publication in the left hand side of a pruning operator can be ignored. We introduce the syntax $f \setminus k$ to say that the consequences of $k$ must be ignored in the execution of $f$.

Until now, we have only presented how to generate the events of all the possible executions produced by the standard semantics in a single execution. These events are instrumented in a way similar to the instrumented semantics. The only new difficulty is that the weak causes of an event $e$ can now happen after $e$. We added a field $e_b$ in the transition labels and in the $\langle f, c, a, b\rangle_L$ structure to represent weak consequences.

The concurrent semantics is defined on the set $Orc_{ic}$ of expressions of this extended syntax:

$$\begin{array}{lcl} f, g, h \in \text{Expression} & ::= & p \| p(p) \| ?k \| f | g \| f|_{(K_L, K_R)} g \\ & & \| f >x> g \| f <x< g \\ & & \| f <x, (K_L, K_R)< g \| f; g \| f;_K g \\ & & \| D\#f \| \bot \| \langle f, K, K, K\rangle_L \| f \setminus k \\ D \in \text{Definition} & ::= & \textbf{def } y(x) = f \\ v \in \text{Orc Value} & ::= & V \| D \\ p \in \text{Parameter} & ::= & v \| \textbf{stop} \| x \| x_P \| \langle p, K, K, K\rangle_L \\ w \in \text{Response} & ::= & NT(v, K, K, K) \| T(v, K, K, K) \\ & & \| Neg(K, K, K) \\ n \in \text{Hidden Label} & ::= & ?V_k(v) \| ?D \| h(\omega) \| h(!v) \\ l \in \text{Label} & ::= & !v \| n \| \omega \| \varepsilon \end{array}$$

## 5.2 Rules

The rules of the instrumented semantics are presented thereafter.

$$\text{(PUBLISH)}\ \frac{}{v \xrightarrow{k,!v,\emptyset,\emptyset,\emptyset}_{ic} \langle \textbf{stop}, \{k\}, \emptyset, \emptyset\rangle_l}\ \begin{array}{l} v \text{ closed} \\ k \text{ fresh} \end{array}$$

$$\text{(STOP)}\ \frac{}{\textbf{stop} \xrightarrow{k,\omega,\emptyset,\emptyset,\emptyset}_{ic} \bot}\ k \text{ fresh}$$

$$\text{(DEFDECLARE)}\ \frac{[D/y]f \xrightarrow{k,l,c,a,b}_{ic} f'}{D\#f \xrightarrow{k,l,c,a,b}_{ic} f'}\ D \text{ is } \textbf{def } y(x) = g$$

$$\text{(VAR)}\ \frac{p \xrightarrow{k,l,c,a,b}_{ic} p'}{x_{P \cup \{p\}} \xrightarrow{k,l,c,a,b}_{ic} x_{P \cup \{p'\}}}$$

$$\text{(CALLSTART)}\ \frac{P \xrightarrow{k,!v,c,a,b}_{ic} P'}{P(p) \xrightarrow{k,\varepsilon,\emptyset,\emptyset,\emptyset}_{ic} (P'(p) \setminus k)|\langle v(p), c, a, b\rangle_l}\ P \neq v$$

$$\text{(CALLN)}\ \frac{P \xrightarrow{k,l,c,a,b}_{ic} P'}{P(p) \xrightarrow{k,l,c,a,b}_{ic} P'(p)}\ l \text{ is } n, \omega \text{ or } \varepsilon$$

$$\text{(INTCALL)}\ \frac{}{D(p) \xrightarrow{k,?D,\emptyset,\emptyset,\emptyset}_{ic} \langle [D/y][p/x]g, \{k\}, \emptyset, \emptyset\rangle_l}\ D \text{ is } \textbf{def } y(x) = g$$

$$\text{(EXTE)}\ \frac{p \xrightarrow{k,\varepsilon,c,a,b}_{ic} p'}{V(p) \xrightarrow{k,\varepsilon,c,a,b}_{ic} V(p')}$$

$$\text{(ExtCall)}\ \frac{p \xrightarrow{k,!v,c,a,b}_{ic} p'}{V(p) \xrightarrow{k,?V_k(v),c,a,b}_{ic} (V(p') \setminus k)|\langle ?k, c \cup \{k\}, a, b\rangle_l}$$

$$\text{(ExtStop)}\ \frac{p \xrightarrow{k,\omega,c,a,b}_{ic} p'}{V(p) \xrightarrow{k,\omega,c,a,b}_{ic} V(p') \setminus k}$$

$$\text{(NtRes)}\ \frac{?k \text{ receives } NT(v,c,a,b)}{?k \xrightarrow{j,!v,c,a,b}_{ic} ?k}\ j \text{ fresh}$$

$$\text{(OtherStart)}\ \frac{f;_\emptyset\, g \xrightarrow{k,l,c,a,b}_{ic} f'}{f; g \xrightarrow{k,l,c,a,b}_{ic} f'}$$

$$\text{(TRes)}\ \frac{?k \text{ receives } T(v,c,a,b)}{?k \xrightarrow{j,!v,c,a,b}_{ic} ?k|\langle \mathbf{stop}, c \cup \{j\}, a, b\rangle_l}\ j \text{ fresh}$$

$$\text{(OtherN)}\ \frac{f \xrightarrow{k,n,c,a,b}_{ic} f'}{f;_K g \xrightarrow{k,n,c,a,b}_{ic} f';_K g}$$

$$\text{(NegRes)}\ \frac{?k \text{ receives } Neg(c,a,b)}{?k \xrightarrow{j,\omega,c,a,b}_{ic} ?k}\ j \text{ fresh}$$

$$\text{(OtherV)}\ \frac{f \xrightarrow{k,!v,c,a,b}_{ic} f'}{f;_K g \xrightarrow{k,!v,c,a,b}_{ic} f';_{K\cup\{k\}} g}$$

$$\text{(OtherStop)}\ \frac{f \xrightarrow{k,\omega,c,a,b}_{ic} f'}{f;_K g \xrightarrow{k,h(\omega),c,a,b}_{ic} f';_K g|\langle g, c \cup \{k\}, a, b\rangle_l}\ c \cap K = \emptyset$$

$$\text{(OtherHalt)}\ \frac{f \xrightarrow{k,\omega,c,a,b}_{ic} f'}{f;_K g \xrightarrow{k,h(\omega),c,a,b}_{ic} f';_K g}\ c \cap K \neq \emptyset$$

$$\text{(SeqN)}\ \frac{f \xrightarrow{k,l,c,a,b}_{ic} f'}{f >x> g \xrightarrow{k,l,c,a,b}_{ic} f' >x> g}\ l \text{ is } n \text{ or } \omega$$

$$\text{(ParStart)}\ \frac{f|_{(\emptyset,\emptyset)} g \xrightarrow{k,l,c,a,b}_{ic} f'}{f|g \xrightarrow{k,l,c,a,b}_{ic} f'}$$

$$\text{(SeqV)}\ \frac{f \xrightarrow{k,!v,c,a,b}_{ic} f'}{f >x> g \xrightarrow{k,h(!v),c,a,b}_{ic} (f' >x> g)|\langle [v/x]g, c \cup \{k\}, a, b\rangle_l}$$

$$\text{(ParLeftStop)}\ \frac{f \xrightarrow{k,\omega,c,a,b}_{ic} f'}{f|_K g \xrightarrow{k,\varepsilon,\emptyset,\emptyset,\emptyset}_{ic} f'|_{(c,a,b)\triangleright K} g}$$

$$\text{(PruneStart)}\ \frac{f <x, (\emptyset,\emptyset)< g \xrightarrow{k,l,c,a,b}_{ic} f'}{f <x< g \xrightarrow{k,l,c,a,b}_{ic} f'}$$

$$\text{(ParRightStop)}\ \frac{g \xrightarrow{k,\omega,c,a,b}_{ic} g'}{f|_K g \xrightarrow{k,\varepsilon,\emptyset,\emptyset,\emptyset}_{ic} f'|_{K\triangleleft(c,a,b)} g}$$

$$\text{(PruneLeft)}\ \frac{f \xrightarrow{k,l,c,a,b}_{ic} f'}{f <x, K< g \xrightarrow{k,l,c,a,b}_{ic} f' <x, K< g}\ l \neq \omega$$

$$\text{(ParLeft)}\ \frac{f \xrightarrow{k,l,c,a,b}_{ic} f'}{f|_K g \xrightarrow{k,l,c,a,b}_{ic} f'|_K g}\ l \neq \omega$$

$$\text{(PruneLeftStop)}\ \frac{f \xrightarrow{k,\omega,c,a,b}_{ic} f'}{f <x, K< g \xrightarrow{k,\varepsilon,\emptyset,\emptyset,\emptyset}_{ic} f' <x, (c,a,b) \triangleright K< g}$$

$$\text{(ParMid)}\ \frac{K \xrightarrow{k,\omega,c,a,b}_{ic} K'}{f|_K g \xrightarrow{k,\omega,c,a,b}_{ic} f|_{K'} g}$$

$$\text{(PRUNEMID)}\ \frac{K \xrightarrow{k,l,c,a,b}_{ic} K'}{f <x, K< g \xrightarrow{k,l,c,a,b}_{ic} f <x, K'< g}$$

$$\text{(PARRIGHT)}\ \frac{g \xrightarrow{k,l,c,a,b}_{ic} g'}{f|_K g \xrightarrow{k,l,c,a,b}_{ic} f|_K g'}\ l \neq \omega$$

$$\text{(PRUNESTOP)}\ \frac{g \xrightarrow{k,\omega,c,a,b}_{ic} g'}{f <x, K< g \xrightarrow{k,h(\omega),c,a,b}_{ic} [\langle \mathbf{stop}, c \cup \{k\}, a, b\rangle_l//x] f <x, K \triangleleft (c,a,b)< g'}$$

$$\text{(PRUNEN)}\ \frac{g \xrightarrow{k,n,c,a,b}_{ic} g'}{f <x, K< g \xrightarrow{k,n,c,a,b}_{ic} f <x, K< \langle g', \emptyset, \{k\}, \emptyset\rangle_{!v}}$$

$$\text{(PRUNEE)}\ \frac{g \xrightarrow{k,\varepsilon,c,a,b}_{ic} g'}{f <x, K< g \xrightarrow{k,\varepsilon,c,a,b}_{ic} f <x, K< g'}$$

$$\text{(PRUNEV)}\ \frac{g \xrightarrow{k,!v,c,a,b}_{ic} g'}{f <x, K< g \xrightarrow{k,h(!v),c,a,b}_{ic} [\langle v, c \cup \{k\}, a, b\rangle_l//x] f <x, K \triangleleft (c,a,b)< \langle g' \setminus k, \emptyset, \{k\}, \{k\}\rangle_{!v}}$$

$$\text{(DISTLEFT)}\ \frac{}{(\{(e, \{e'\} \cup L)\} \cup K_L, K_R) \xrightarrow{k,\omega,e_c \cup e'_c, e_a \cup e'_a, e_b \cup e'_b}_{ic} (\{(e, L)\} \cup K_L, K_R)}\ \begin{array}{l} k \text{ fresh} \\ e_c \cap e'_b = \emptyset \\ e'_c \cap e_b = \emptyset \end{array}$$

$$\text{(DISTRIGHT)}\ \frac{}{(K_L, \{(e, \{e'\} \cup L)\} \cup K_R) \xrightarrow{k,\omega,e_c \cup e'_c, e_a \cup e'_a, e_b \cup e'_b}_{ic} (K_L, \{(e, L)\} \cup K_R)}\ \begin{array}{l} k \text{ fresh} \\ e_c \cap e'_b = \emptyset \\ e'_c \cap e_b = \emptyset \end{array}$$

$$\text{(CAUSEYES)}\ \frac{f \xrightarrow{k,l,c,a,b}_{ic} f'}{\langle f, c', a', b'\rangle_L \xrightarrow{k,l,c \cup c', a \cup a' \cup c', b \cup b'}_{ic} \langle f', c', a', b'\rangle_L}\ \begin{array}{l} l \in L \\ c \cap b' = \emptyset \\ c' \cap b = \emptyset \end{array}$$

$$\text{(CAUSENO)}\ \frac{f \xrightarrow{k,l,c,a,b}_{ic} f'}{\langle f, c', a', b'\rangle_L \xrightarrow{k,l,c,a,b}_{ic} \langle f', c', a', b'\rangle_L}\ l \notin L$$

$$\text{(HIDE)}\ \frac{f \xrightarrow{k,l,c,a,b}_{ic} f'}{f \setminus k' \xrightarrow{k,l,c,a,b \cup \{k\}}_{ic} f' \setminus k'}\ k' \notin c$$

The first remark is the far bigger number of rules compared to the other semantics. For example, it requires twice as much rules to define the parallel and pruning operators. For the parallel operator, for example, only two rules have their counterparts in the standard semantics. The rule PARSTART is introduced only for adding an empty distributivity structure. The rules PARLEFTSTOP, PARRIGHTSTOP and PARMID illustrate the problem explained above. The program $\mathbf{stop}|_{(\emptyset,\emptyset)} Stop$ needs three steps to generate an $\omega$ event. The first two, generated by PARLEFTSTOP and PARRIGHTSTOP are labelled by $\varepsilon$ and put in the distributivity structure. Then rule PARMID is used to generate $\omega$ with the correct causes.

The same approach is used for the pruning operator, explaining the high number of rules. Let us now consider the rule PRUNEV. The right hand side is no longer destroyed when it publishes a value. Instead, a stop event is added into the distributivity structure to allow the future publications of $\omega$, and the right hand side expression is instrumented in two ways. First, the consequences of the publication are ignored, and the future publications are considered as both weak causes and consequences of this event, to enforce a cycle in the weak causality relation, i.e. a conflict.

Surprisingly, the definition of the sequential operator is simpler in the concurrent semantics. This is due to the fusion of the rules SEQN and SEQSTOP, since nothing needs to be destroyed anymore when an $\omega$ event is observed.

## 5.3 Concurrent executions

The rules presented in Figure **??** define the transition system $\rightarrow_{ic}$ and the sequential semantics $[\![\cdot]\!]_{ic}$. To define our concurrent semantics, we also need to consider the infinite executions. The set of all the — finite and infinite — executions starting from $f_0$ is denoted $[\![f_0]\!]^\infty_{ic}$.

**Definition 5** (Concurrent execution). *Let $\sigma \in [\![f_0]\!]^\infty_{ic}$. We define the* concurrent execution *of $\sigma$ as the tuple*

$$\overline{\overline{\sigma}} = (\{\sigma[i]_k | i \leq |\sigma| \sigma[i]_l \neq \varepsilon\}, \{\sigma[i]_l | i \leq |\sigma|\} \setminus \{\varepsilon\}, \leq, \nearrow, \Lambda)$$

*where for all $i, j$:*

- $\sigma_k^i \leq \sigma_k^j$ *if* $\sigma_k^i \in \sigma_c^j$ *or* $i = j$,
- $\sigma_k^i \nearrow \sigma_k^j$ *if* $\sigma_k^i \in \sigma_a^j$ *or* $\sigma_b^i \cap (\sigma_c^j \cup \{\sigma_k^j\}) \neq \emptyset$
- $\Lambda(\sigma_k^i) = \sigma_l^i$.

In this semantics, we expect the LAES obtained from an execution to be complete. To this purpose, we need to consider the executions that contain all the possible events. However, there is no guaranty on the choices made by the system if several events can be fired concurrently. For example, if two Orc programs are run in parallel, it is possible for a scheduler to ignore one of them and to only execute the other. We remove the starvation cases by introducing a fairness property. Intuitively, the concurrent semantics is confluent, in the sense that if two transitions are possible from an expression, they can be drawn in any order, which gives two similar expressions. The fairness property means that if a transition is enabled at one time, it will be eventually fired.

**Definition 6** (Fair execution)**.** *A fair execution is a (possibly infinite) execution* $\sigma \in [\![f_0]\!]_{ic}^{\infty}$ *such that, for any prefix* $\gamma$ *of* $\sigma$, $\gamma \in [\![f_0]\!]_{ic}$, *and for any event* $e$ *such that* $\gamma.e \in [\![f_0]\!]_{ic}$, *there is an injection from* $\overline{\overline{\gamma.e}}$ *to* $\overline{\overline{\sigma}}$ *that preserves* $\gamma$.

*The set of all fair executions starting from* $f_0$ *is denoted* $[\![f_0]\!]_{ic}^{\omega}$.

## 5.4 Justification

In the remainder of this section, we limit our analysis to fair executions. Even though the technique is classic to avoid degenerate cases such as those mentioned above, we have to justify that it is not too restrictive. Property 2 claims that all finite executions can be completed into a fair execution. In other words, it is still possible to chose the event one wants to fire at any time as the other events will still be enabled afterwards. As a direct consequence, still the empty execution belongs to $[\![f_0]\!]_{ic}$, every Orc program has at least one fair execution. Another consequence is that maximal finite executions, if they do exist, are fair.

**Property 2** (Fair completion)**.** *For all* $\gamma \in [\![f_0]\!]_{ic}$, *there is a (possibly infinite) word of events* $\sigma$ *such that* $\gamma.\sigma \in [\![f_0]\!]_{ic}^{\omega}$.

The goal of a concurrent semantics is to define a unique object that contains all the information on the execution of a distributed program. As we have seen above, it is possible to generate a lot of fair executions by completing different finite executions. Property 3 claims that all these fair executions actually define the same object: *the* concurrent semantics of an Orc program.

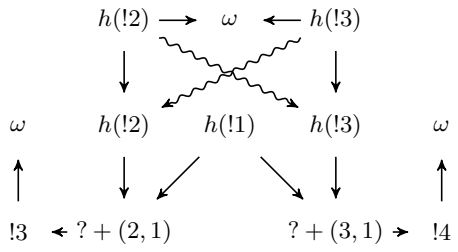
**Property 3** (fairness equivalence)**.** *Two fair executions define equivalent LAESs:* $\forall f \in Orc, \forall \sigma, \tau \in [\![f]\!]_{ic}^{\omega}, \overline{\overline{\sigma}} \equiv \overline{\overline{\tau}}$.

The proof of the correctness of this concurrent semantics consists of two points (1) the semantics is sound, as it does not introduce new behaviors like the instrumented semantics, and (2) the linearizations represent sequential executions of the standard semantics. It is also complete; all the behaviors are encoded in the linearizations of the concurrent semantics.

**Theorem 2** (Correctness)**.** *Fair executions are sound and complete:* $\forall f \in Orc, \forall \sigma \in [\![f]\!]_{ic}^{\omega}, Lin(\overline{\overline{\sigma}}) = [\![f]\!]$.

## 5.5 Illustration

Any fair execution of the concurrent semantics of the program $f_0 = y + z <y< ((2|3) >x> x) <z< 1$ now gives the same LAES:




The first observation is that we can obtain the LAES of the instrumented semantics (Figure 4.5) by simply removing the second $h(!2)$ and all its consequences as well as the first $\omega$. This is an example of LAES injection.

Let us now observe the first instance of $h(!2)$. It is a cause of the second one and is preempted by the second instance of $h(!3)$, so both are in relation of weak causality. By symmetry, the second instance of $h(!3)$ is also a weak cause of those of $h(!2)$. It follows a cycle in the weak causality relation, hence a conflict.

# 6 Proof

In this section, we prove the theorems stated above. Theorem 1 appears to be a consequence of Theorem 2, in which we use the properties concerning the instrumented semantics. This section is organized as follows: first, we define few lemmas about the links that the two semantics have with each other and how the events can be reordered in different executions. Then, the asserted properties and theorems are recalled and proved.

**Lemma 1.** $\forall f \in Orc, \forall \sigma \in [\![f]\!], \exists \sigma_i \in [\![f]\!]_i, \sigma_i|_l = \sigma$.

*Proof.* This lemma is shown by induction on the length of the execution. What we actually prove is a bit more general, as the intermediate expressions contain more information in the instrumented than in the standard semantics.

Let $f_i \in Orc_i$ be an instrumented program. We define its projection $\pi^i_s(f_i)$ on $Orc_s$ as as the same expression in which the angle brackets are removed, i.e.

$$\begin{array}{rcl}
\pi^i_s(\langle f, c, a\rangle_L) & = & \pi^i_s(f) \\
\pi^i_s(\mathbf{def}\ y(\overline{x}) = f) & = & \mathbf{def}\ y(\overline{x}) = \pi^i_s(f) \\
\pi^i_s(p) & = & p \\
\pi^i_s(p(\overline{q})) & = & \pi^i_s(p)(\overline{\pi^i_s(q)}) \\
\pi^i_s(?k) & = & ?k \\
\pi^i_s(f|g) & = & \pi^i_s(f)|\pi^i_s(g) \\
\pi^i_s(f >x> g) & = & \pi^i_s(f) >x> \pi^i_s(g) \\
\pi^i_s(f <x< g) & = & \pi^i_s(f) <x< \pi^i_s(g) \\
\pi^i_s(f; g) & = & \pi^i_s(f); \pi^i_s(g) \\
\pi^i_s(D\#f) & = & \pi^i_s(D)\#\pi^i_s(f) \\
\pi^i_s(\bot) & = & \bot.
\end{array}$$

Let $f_0 \in Orc$. For all $n \in \mathbb{N}$, $P(n)$ designates the following property: For all execution $f_0 \xrightarrow{l^1} f_1 ... \xrightarrow{l^n} f_n$ of the standard semantics, there is an execution $F_0 \xrightarrow{\sigma[1]}_i F_1 ... \xrightarrow{\sigma[n]}_i F_n$ such that for all $i$, $\sigma[i]_l = l_i$ and $\pi^i_s(F_i) = f_i$.

$P(0)$ is true because $\pi^i_s(f_0) = f_0$. Suppose $P(n)$ and we prove $P(n+1)$. The induction hypothesis can be applied on the first $n$ steps, and we have $F_n$ such that $\pi^i_s(F_n) = f_n \xrightarrow{l_{n+1}} f_{n+1}$. To exhibit the last step, we also have to be more general, by proving that for all step $f \xrightarrow{l} f'$ and all $F$ with $\pi^i_s(F) = f$, there is a step $F \xrightarrow{k,l,c,a}_i F'$ with $\pi^i_s(F') = f'$. This is shown by induction on the derivation tree of the transition.

There is a finite $m \in \mathbb{N}$ such that $F = \langle\langle ... \langle G, c^m_l, c^m_\omega\rangle_{L^m} ...\rangle_{L^2}, c^1_l, c^1_\omega\rangle_{L^1}$ where $G$ is not of the form $\langle G', c_l, c_\omega\rangle_L$. Moreover, $\pi^i_s(G) = \pi^i_s(F) = f \xrightarrow{l} f'$. We now considere the step $\pi^i_s(g) \xrightarrow{l} f'$. The rule in the root of the tree can be anything different from CAUSALYES and CAUSALNO. It can have zero, one or two premises of the form $h \xrightarrow{l'} h'$. Moreover, there is a rule with the same name in the instrumented semantics, with the same number of premises, of the form $H \xrightarrow{k,l',c,a} H'$, with $\pi^i_s(H) = h$. By induction, the premise can be satisfied with $\pi^i_s(H') = h'$, and we can apply the rule, which gives $G \xrightarrow{k,l,c',a'} G'$ with $[\![G']\!] = f'$. Finally, we can apply $m$ times the rule CAUSALYES or CAUSALNO depending on $l$, and get $F \xrightarrow{k,l,c'',a''} F' = \langle\langle ... \langle G, c^m_l, c^m_\omega\rangle_{L^m} ...\rangle_{L^2}, c^1_l, c^1_\omega\rangle_{L^1}$, with $[\![F']\!] = [\![G']\!] = f'$.

This proves $P(n+1)$, so $P(n)$ for all $n$. □

**Lemma 2.** $\forall f \in Orc, \forall \sigma_i \in [\![f]\!]_i, \exists \sigma_{ic} \in [\![f]\!]_{ic}, \overline{\overline{\sigma_i} < \overline{\sigma_{ic}}}$. *Moreover, for all* $i$, $\sigma_{ic}[i]_b = \emptyset$ *and* $\sigma_{ic}|_l = \sigma_i|_l$

*Proof.* The proof for this lemma is very similar to the previous one. It consist of an induction on the length of the execution where each intermediate expression has some property, namely the projection of the expressions of the concurrent semantics are those of the instrumented semantics ($\pi^{ic}_i(F) =$

$f$) and the expressions of the concurrent semantics are non-conflictual, which means that they were obtained from an execution where no conflict was observed.

Here is the full definition of $\pi_i^{ic}$:

$$
\begin{array}{rcl}
\pi_i^{ic}(\mathbf{stop}) & = & \mathbf{stop} \\
\pi_i^{ic}(V) & = & V \\
\pi_i^{ic}(\mathbf{def}\ y(x) = f) & = & \mathbf{def}\ y(x) = \pi_i^{ic}(f) \\
\pi_i^{ic}(\langle p, c, a, b\rangle_L) & = & \langle \pi_i^{ic}(p), c, a\rangle_L \\
\pi_i^{ic}(x) & = & x \\
\pi_i^{ic}(x_p) & = & \pi_i^{ic}(p) \\
\pi_i^{ic}(x_{p_1,p_2,\ldots}) & = & \bot \\
\pi_i^{ic}(\langle f, c, a, b\rangle_L) & = & \langle \pi_i^{ic}(f), c, a\rangle_L \\
\pi_i^{ic}(f \setminus k) & = & \bot \\
\pi_i^{ic}(p(p')) & = & \begin{cases} \pi_i^{ic}(p)(\pi_i^{ic}(p')) & \text{if } \pi_i^{ic}(p) \neq \bot \neq \pi_i^{ic}(p') \\ \bot & \text{otherwise} \end{cases} \\
\pi_i^{ic}(?k) & = & ?k \\
\pi_i^{ic}(\bot) & = & \bot \\
\pi_i^{ic}(D\#f) & = & \pi_i^{ic}(D)\#\pi_i^{ic}(f) \\
\pi_i^{ic}(f|g) & = & \begin{cases} \pi_i^{ic}(f)|\pi_i^{ic}(g) & \text{if } \pi_i^{ic}(p) \neq \bot \neq \pi_i^{ic}(p') \\ \bot & \text{otherwise} \end{cases} \\
\pi_i^{ic}(f|_{K_0} g) & = & \pi_i^{ic}(f|g) \\
\pi_i^{ic}(f|_{e \triangleright K_0} g) & = & \langle \mathbf{stop}, e_c, e_a\rangle_\omega | \pi_i^{ic}(g) \text{ if } \pi_i^{ic}(g) \neq \bot \\
\pi_i^{ic}(f|_{K_0 \triangleleft e} g) & = & \pi_i^{ic}(f) | \langle \mathbf{stop}, e_c, e_a\rangle_\omega \text{ if } \pi_i^{ic}(f) \neq \bot \\
\pi_i^{ic}(f|_{e \triangleright K_0 \triangleleft e'} g) & = & \langle \mathbf{stop}, e_c \cup e'_c, e_a \cup e'_a\rangle_\omega \\
\pi_i^{ic}(f|_K g) & = & \bot \text{ otherwise} \\
\pi_i^{ic}(f >x> g) & = & \begin{cases} \pi_i^{ic}(f) >x> g & \text{if } \pi_i^{ic}(f) \neq \bot \\ \bot & \text{otherwise} \end{cases} \\
\pi_i^{ic}(f <x< g) & = & \begin{cases} \pi_i^{ic}(f) <x< \pi_i^{ic}(g) & \text{if } \pi_i^{ic}(p) \neq \bot \neq \pi_i^{ic}(p') \\ \bot & \text{otherwise} \end{cases} \\
\pi_i^{ic}(f <x, K_0< g) & = & \pi_i^{ic}(f <x< g) \\
\pi_i^{ic}(f <x, e \triangleright K_0< g) & = & \langle \mathbf{stop}, e_c, e_a\rangle_\omega <x< \pi_i^{ic}(g) \text{ if } \pi_i^{ic}(f) \neq \bot \\
\pi_i^{ic}(f <x, K_0 \triangleleft e< g) & = & \pi_i^{ic}(f) <x< \langle \mathbf{stop}, e_c, e_a\rangle_\omega \text{ if } \pi_i^{ic}(g) \neq \bot \\
\pi_i^{ic}(f <x, e \triangleright K_0 \triangleleft e'< g) & = & \langle \mathbf{stop}, e_c \cup e'_c, e_a \cup e'_a\rangle_\omega \\
\pi_i^{ic}(f <x, K< g) & = & \bot \text{ otherwise} \\
\pi_i^{ic}(f; g) & = & \pi_i^{ic}(f); \pi_i^{ic}(g) \\
\pi_i^{ic}(f;_\emptyset\ g) & = & \begin{cases} \pi_i^{ic}(g) & \text{if } \pi_i^{ic}(f) = \bot \\ \pi_i^{ic}(f); \pi_i^{ic}(g) & \text{otherwise} \end{cases} \\
\pi_i^{ic}(f;_K\ g) & = & \pi_i^{ic}(f)
\end{array}
$$

We now give the full definition of non-conflictuality. A parameter $p$ is non-conflictual if it is of the form

- $\mathbf{def}\ y(x) = f$, where $f$ is non-conflictual
- $\langle p, c, a, b\rangle_L$ where $p$ is non-conflictual
- $V$, $\mathbf{stop}$, $x$
- $x_{\{p\}}$ where $p$ is non-conflictual

A program $F$ is non-conflictual if it is of the form

- $?k$, $\bot$,
- $p$, $P(p)$ where $p$ and $P$ are non-conflictual
- $f \setminus k$, $\langle f, c, a, \emptyset\rangle_L$ where $f$ is non-conflictual
- $f|g$, $f|_{K_0} g$, $f >x> g$, $f <x< g$, $f <x, K_0< g$, $f; g$, $f;_K g$, $\mathbf{def}\ y(x) = f\#g$, where $f$ and $g$ are non-conflictual
- $f|_{K_3} g$, $f <x, K_3< g$ where $\pi_i^{ic}(f) = \pi_i^{ic}(g) = \bot$
- $f|_{K_1^\triangleleft} g$, where $f$ is non-conflictual and $\pi_i^{ic}(g) = \bot$

The last definition is a notation: we denote $F \xrightarrow{k,l,c,a,\emptyset}{}^{n}_{ic} F'$ if there are $F_1, ..., F_n$ and identifiers $k_1, ..., k_n$ such that $F \xrightarrow{k_1,\epsilon,\emptyset,\emptyset,\emptyset}_i F_1 ... \xrightarrow{k_n,\epsilon,\emptyset,\emptyset,\emptyset}_i F_n \xrightarrow{k,l,c,a,\emptyset}_{ic} F'$. We write $F \xrightarrow{k,l,c,a,\emptyset}{}^{\star}_{ic} F'$ if this is true for some $n$.

Let $f_0$ be an Orc program and $\sigma_i \in [\![f_0]\!]_i$. Let $f_1, ..., f_n$ such that $f_0 \xrightarrow{\sigma_i[1]}_i f_1 ... f_{n-1} \xrightarrow{\sigma_i[n]}_i f_n$.

We prove that there are $F_0, ..., F_n$ such that for all $i$, $F_i$ is non-conflictual and $\pi_i^{ic}(F_i) \equiv f_i$, and $f_0 = F_0 \xrightarrow{\sigma_i[1]}{}^{\star}_{ic} F_1 ... F_{n-1} \xrightarrow{\sigma_i[n]}{}^{\star}_{ic} F_n$ by induction on $n$.

If $n = 0$, we just have to take $F_0 = f_0$, that is trivially non-conflictual and $f_0 = \pi_i^{ic}(f_0)$. We suppose the property true for all $\sigma_i$ of length $n$, and suppose $\sigma_i$ is of length $n+1$. By induction, we know that there is $F_n$ non-conflictual such that $\pi_i^{ic}(F_n) \equiv f_n$. As they are equivalent, $\pi_i^{ic}(F_n) \xrightarrow{\sigma_i[n+1]}_i g \equiv f_{n+1}$. We must now prove that there is $F_{n+1}$ non-conflictual such that $\pi_i^{ic}(F_{n+1}) \equiv g f_{n+1}$ and $F_n \xrightarrow{\sigma_i[n+1]_k, \sigma_i[n+1]_l, \sigma_i[n+1]_c, \sigma_i[n+1]_a, \emptyset}{}^{\star}_{ic} F_{n+1}$. To this extend, we prove the property P: for all $f$, $f'$, $(k, l, c, a)$ such that $f \xrightarrow{k,l,c,a}_i f'$, for all non-conflictual $F$ such that $\pi_i^{ic}(F) = f$, there is a non-conflictual $F'$ such that $\pi_i^{ic}(F') \equiv f'$ and $F \xrightarrow{k,l,c,a,\emptyset}{}^{n}_{ic} F'$.

We prove P by induction on the derivation tree of the step, i.e. we prove this for the axioms, and supposing P for the premises of the other rules (induction hypothesis H1), we prove it for these rules. We can remark that the base cases are exactly those in which no hypothesis is made, so we do not separate this cases.

Let $f \xrightarrow{k,l,c,a}_i f'$, that was generated by rule $R$ and $F$ non-conflictual such that $\pi_i^{ic}(F) = f$. The proof is made by induction on the syntax of $F$. Similarly, we have the induction hypothesis H2: if $G$ is a non-conflictual sub-expression of $F$ and $\pi_i^{ic}(G) \xrightarrow{k',l',c',a'}_i g'$, then there is $G'$ non-conflictual with $\pi_i^{ic}(G') \equiv g'$ such that $G \xrightarrow{k',l',c',a',\emptyset}{}^{m}_{i} G'$.

Here are listed the possible cases for $F$:

**stop (idem for $v$) :** $f = \mathbf{stop}$, so R is STOP, the transition is $\mathbf{stop} \xrightarrow{k,\omega,\emptyset,\emptyset}_i \bot$, and we can apply rule STOP on $F$, so $\mathbf{stop} \xrightarrow{k,\omega,\emptyset,\emptyset,\emptyset}_{ic} \bot$ where $F' = \bot$ is non-conflictual and $\pi_i^{ic}(\bot) = \bot = f'$.

**$x_p$:** $f = \pi_i^{ic}(p)$, so we have $\pi_i^{ic}(p) \xrightarrow{k,l,c,a}_i p'$ and by H1, $p \xrightarrow{k,l,c,a,\emptyset}{}^{n}_{ic} P'$ with $\pi_i^{ic}(P') \equiv p'$. It is possible to apply rule VAR $n+1$ times, so $x_p \xrightarrow{k,l,c,a}{}^{n}_{ic} x_{P'}$.

**$P(p)$:** $f = \pi_i^{ic}(P)(\pi_i^{ic}(P))$, R can be one of these rules:

- **STOPCALL:** by H1, $P \xrightarrow{k,\omega,c,a,\emptyset}{}^{n}_{ic} P'$, so we can apply rule CALLN $n+1$ times in the concurrent semantics.
- **INTCALL (idem for EXTCALL and EXTSTOP):** $F = P(p')$. If $P = D$, we can apply rule INTCALL in the instrumented semantics. Otherwise, by H1, $P \xrightarrow{k,!D,c,a,\emptyset}_{ic} P'$, so we can apply rule CALLN $n$ times, then rule CALLSTART once and rule INTCALL as a premise of CAUSEYES, PARRIGHT or PARRIGHTSTOP and PARSTART, which gives $F \xrightarrow{k,?D,c,a,\emptyset}{}^{n+1}_{ic} F' = (P'(p) \setminus k')|_{(\emptyset,K_R)} \langle\langle [D/y][p/x]g, \{k\}, \emptyset, \emptyset\rangle_l, c, a, \emptyset\rangle_l$, with $\pi_i^{ic}(F') = \langle\langle [D/y][p/x]g, \{k\}, \emptyset\rangle_l, c, a\rangle_l \equiv \langle [D/y][p/x]g, c \cup \{k\}, a\rangle_l = f'$.

**$?k$ :** $f = ?k$, so R can only be one of NTRES, TRES or NEGRES, and we can apply the same rule in the concurrent semantics.

**$\langle G, c, a, \emptyset\rangle_L$:** $g$ is non-conflictual and $\pi_i^{ic}(F) = \langle \pi_i^{ic}(G), c, a\rangle_L$. Using H1, we have $G \xrightarrow{k,\omega,c',a',\emptyset}{}^{n}_{ic}$ and we can apply rules CAUSEYES or CAUSENO $n+1$ times depending on $l$ and $L$.

**$G|_{(\emptyset,\emptyset)}H$:** as $F$ is non-conflictual, $G$ and $H$ are different from $\bot$, and R was one of:

- **PARLEFT (idem for PARRIGHT):** $\pi_i^{ic}(G) \xrightarrow{k,l,c,a} g'$. By H1, we have $G \xrightarrow{k,l,c,a,\emptyset}{}^{n}_{ic} G'$, so we can apply rule PARLEFT $n+1$ times, which gives $G|_{(\emptyset,\emptyset)}H \xrightarrow{k,l,c,a,\emptyset}{}^{n}_{i} G'|_{(\emptyset,\emptyset)}H$.
- **PARSTOP:** $\pi_i^{ic}(G) \xrightarrow{k,\omega,c,a} \bot$ and $\pi_i^{ic}(H) \xrightarrow{k,\omega,c',a'} \bot$. By H1, we have $G \xrightarrow{k,\omega,c,a,\emptyset}{}^{m}_{ic} G'$ and $H \xrightarrow{k,\omega,c',a',\emptyset}{}^{n}_{ic} H'$. We can apply rule PARLEFT $m$ times, rule PARLEFTSTOP once, rule PARRIGHT $n$ times, rule PARRIGHTSTOP once and rule PARMID once, to get $F \xrightarrow{k,\omega,c\cup c',a\cup a',\emptyset}{}^{m+n+2}_{ic} F' = G'|_{(\{((a,c,\emptyset),\emptyset)\},\{((a',c',\emptyset),\emptyset)\})}H'$, with $\pi_i^{ic}(F') \equiv \bot$.

**$G|_{(K_L,K_R)}H$:** The other cases are either conflictual, or $\pi_i^{ic}(F) = \bot$.

**$G <x, (\emptyset, \emptyset)< H$:** as $F$ is non-conflictual, $G$ and $H$ are different from $\bot$, and R is one of:

- **PRUNELEFT:** $\pi_i^{ic}(G) \xrightarrow{k,l,c,a} g'$. By H1, $G \xrightarrow{k,l,c,a,\emptyset}{}^{n}_{ic} G'$, so we can apply rule PRUNELEFT $n+1$ times, which gives $G <x, (\emptyset, \emptyset)< H \xrightarrow{k,l,c,a,\emptyset}{}^{n}_{i} G' <x, (\emptyset, \emptyset)< H$.

**PRUNEN:** $\pi_i^{ic}(H) \xrightarrow{k,n,c,a}_i h'$. By H1, $H \xrightarrow{k,n,c,a,\emptyset}{}^{n}_{ic} H'$, so we can apply PRUNEE $n$ times and PRUNEN once, so $F \xrightarrow{k,l,c,a,\emptyset}{}^{n}_{ic} G <x,(\emptyset,\emptyset)< \langle H',\emptyset,\{k\},\emptyset\rangle$.

**PRUNEV:** $\pi_i^{ic}(H) \xrightarrow{k,!v,c,a}_i h'$. By H1, $H \xrightarrow{k,!v,c,a,\emptyset}{}^{n}_{ic} H'$, so we can apply PRUNEE $n$ times and PRUNEV once, which gives $F \xrightarrow{k,l,c,a,\emptyset}{}^{n}_{i} F' = [x|\langle v, c\cup\{k\},a,b\rangle_l/x]G < x,(\emptyset,\{((c,a,b),\emptyset)\})< \langle H',\emptyset,\{k\},\{k\}\rangle_{!v}$. $F'$ is non-conflictual and $\pi_i^{ic}(F') = \pi_i^{ic}([x|\langle v, c\cup\{k\},a,b\rangle_l/x]G) = [v/x]\pi_i^{ic}(G) \equiv f'$.

**PRUNESTOP:** $\pi_i^{ic}(H) \xrightarrow{k,\omega,c,a}_i \bot$. By H1, we have $H \xrightarrow{k,\omega,c,a,\emptyset}{}^{n}_{ic} H'$, so we can apply PRUNEE $n$ times and PRUNESTOP once, which gives $F \xrightarrow{k,h(\omega),c,a,\emptyset}{}^{n}_{i} F' = [x|\langle \mathbf{stop}, c\cup\{k\},a,b\rangle_l/x]G <x,(\emptyset,\{((c,a,b),\emptyset)\})< H'$ with $\pi_i^{ic}(H') \equiv \bot$. $F'$ is non-conflictual and $\pi_i^{ic}(F') = \pi_i^{ic}([x|\langle \mathbf{stop}, c\cup\{k\},a,b\rangle_l/x]G) = [\mathbf{stop}/x]\pi_i^{ic}(G) \equiv f'$.

$G <x,(K_L,K_R)< H$**:** the only other cases that is not conflictual, where $\pi_i^{ic}(F) \neq \bot$ is $\pi_i^{ic}(H) = \bot$, $K_R = \{(c,a,b),\emptyset\}$. We have $\pi_i^{ic}(F) = \pi_i^{ic}(G)$, so by H2, $G \xrightarrow{k,l,c,a,\emptyset}{}^{n}_{ic} G'$. We can use rule PRUNELEFT $n+1$ times, which gives $G <x,(K_L,K_R)< H \xrightarrow{k,l,c,a,\emptyset}{}^{n}_{i} G' <x,(K_L,K_R)< H$.

$G;_\emptyset H$**:** $G$ and $H$ are non-conflictual, and $R$ is one of:

**OTHERN:** We have $\pi_i^{ic}(G) \xrightarrow{k,n,c,a}_i \pi_i^{ic}(G')$ with $G'$ non-conflictual. We can apply OTHERN, so $F \xrightarrow{k,n,c,a,\emptyset}_i G';_\emptyset H = F'$.

**OTHERV:** We have $\pi_i^{ic}(G) \xrightarrow{k,!v,c,a}_i \pi_i^{ic}(G')$ with $G'$ non-conflictual. We can apply OTHERV, so $F \xrightarrow{k,!v,c,a,\emptyset}_i G';_{\{k\}} H = F'$, with $F'$ non-conflictual and $\pi_i^{ic}(F') = \pi_i^{ic}(G') \equiv f'$.

**OTHERSTOP:** We have $\pi_i^{ic}(G) \xrightarrow{k,\omega,c,a}_i \pi_i^{ic}(G') = \bot$ with $G'$ non-conflictual. We can apply OTHERSTOP, so $F \xrightarrow{k,!v,c,a,\emptyset}_i G';_\emptyset H|\langle H, c\cup\{k\},a,b\rangle_l = F'$, with $F'$ non-conflictual and $\pi_i^{ic}(F') = \langle \pi_i^{ic}(H), c\cup\{k\},a\rangle_l \equiv f'$.

$G;_K H$**:** $\pi_i^{ic}(G) = f \xrightarrow{k,l,c,a} f' = \pi_i^{ic}(G')$. We can apply one of the rules depending on $l$. If $l = \omega$, there was a publication, that is a cause of $\omega$, so $c \cap K \neq \emptyset$. We can build $F'$ as an application of OTHERN, OTHERV or OTHERHALT, and in all these cases, $\pi_i^{ic}(F') = \pi_i^{ic}(G')$ is non-conflictual.

$G|H$**, resp** $G <x< H$**, resp** $G;H$**:** the previous reasonings gives us the properties for $G|_{(\emptyset,\emptyset)}H$, $G < x,(\emptyset,\emptyset)< H$ and $G;_\emptyset H$. If we also apply rule PARSTART, resp PRUNESTART, resp OTHERSTART on the first step, we have the property for $G|H$, $G <x< H$ and $G;H$.

$G >x> H$**:** as a step is possible, $\pi_i^{ic}(G) \neq \bot$ and $f = \pi_i^{ic}(G) >x> H$. R can be the following rules:

**SEQN:** by H1, $G \xrightarrow{k,l,c,a,\emptyset}{}^{n}_{i} G'$, so we can apply SEQN $n+1$ times, which gives $G > x > H \xrightarrow{k,l,c,a,\emptyset}{}^{n}_{i} G' >x> H$.

**SEQV:** by H1, $G \xrightarrow{k,!v,c,a,\emptyset}{}^{n}_{i} G'$, so we can apply SEQN $n$ times and SEQV once, which gives $G > x > H \xrightarrow{k,!v,c,a,\emptyset}{}^{n}_{i} G' > x > H|\langle [v/x]g, c\cup\{k\},a,\emptyset\rangle_l$ and $\pi_i^{ic}(G' > x > H|\langle [v/x]g, c\cup\{k\},a,\emptyset\rangle_l) = \pi_i^{ic}(G') >x> H|\langle [v/x]g, c\cup\{k\},a\rangle_l \equiv f'$.

**SEQSTOP:** by H1, $G \xrightarrow{k,\omega,c,a,\emptyset}{}^{n}_{i} G'$, with $\pi_i^{ic}(G') = \bot$, so we can apply SEQN $n+1$ times, which gives $G >x> H \xrightarrow{k,\omega,c,a,\emptyset}{}^{n}_{ic} G' >x> H$. As $\pi_i^{ic}(G') = \bot$, $\pi_i^{ic}(G' >x> H) \equiv \bot = f'$.

**def** $y(x) = G\#H$**:** we have $f$ is **def** $y(x) = \pi_i^{ic}(G)\#\pi_i^{ic}(H)$, so R is DEFDECLARE. By H1, $[D/y]H \xrightarrow{k,l,c,a,\emptyset}{}^{n}_{ic} H'$. We can apply DEFDECLARE, so $D\#H \xrightarrow{k,l,c,a,\emptyset}{}^{n}_{ic} H'$.

Let us denotate the execution we just built by $\sigma_{ic}$. We have $\sigma_{ic} \in [\![f_0]\!]_{ic}$ and $\overline{\overline{\sigma_i}} = \overline{\overline{\sigma_{ic}}}$.

□

**Lemma 3.** *Let $f \in Orc$ and $\sigma \in [\![f]\!]_{ic}$ such that for all $i$, $\sigma[i]_b = \emptyset$. Let $\sigma|_l$ the word constituted of the labels of $\sigma$ different from $\varepsilon$. We have $\sigma|_l \in [\![f]\!]$.*

*Proof.* This proof is very similar to the previous ones, and we have similar notions of projection $\pi_s^{ic}$

and non-conflictuality. We define the projection $\pi_s^{ic}(f)$ as :

$$
\begin{array}{rcl}
\pi_s^{ic}(\mathbf{stop}) & = & \mathbf{stop} \\
\pi_s^{ic}(V) & = & V \\
\pi_s^{ic}(\mathbf{def}\ y(x) = f) & = & \mathbf{def}\ y(x) = \pi_s^{ic}(f) \\
\pi_s^{ic}(\langle p, c, a, b\rangle_L) & = & \pi_s^{ic}(p) \\
\pi_s^{ic}(x) & = & x \\
\pi_s^{ic}(x_p) & = & \pi_s^{ic}(p) \\
\pi_s^{ic}(x_{p_1,p_2,\ldots}) & = & \bot \\
\pi_s^{ic}(\langle f, c, a, b\rangle_L) & = & \pi_s^{ic}(f) \\
\pi_s^{ic}(f \setminus k) & = & \bot \\
\pi_s^{ic}(p(p')) & = & \begin{cases} \pi_s^{ic}(p)(\pi_s^{ic}(p')) & \text{if } \pi_s^{ic}(p) \neq \bot \neq \pi_s^{ic}(p') \\ \bot & \text{otherwise} \end{cases} \\
\pi_s^{ic}(\bot) & = & \bot \\
\pi_s^{ic}(D\#f) & = & \pi_s^{ic}(D)\#\pi_s^{ic}(f) \\
\pi_s^{ic}(f|g) & = & \begin{cases} \pi_s^{ic}(f)|\pi_s^{ic}(g) & \text{if } \pi_s^{ic}(p) \neq \bot \neq \pi_s^{ic}(p') \\ \bot & \text{otherwise} \end{cases} \\
\pi_s^{ic}(f|_{K_0}g) & = & \pi_s^{ic}(f|g) \\
\pi_s^{ic}(f|_{e \triangleright K_0}g) & = & \langle \mathbf{stop}, e_c, e_a\rangle_\omega | \pi_s^{ic}(g) \text{ if } \pi_s^{ic}(g) \neq \bot \\
\pi_s^{ic}(f|_{K_0 \triangleleft e}g) & = & \pi_s^{ic}(f) | \langle \mathbf{stop}, e_c, e_a\rangle_\omega \text{ if } \pi_s^{ic}(f) \neq \bot \\
\pi_s^{ic}(f|_{e \triangleright K_0 \triangleleft e'}g) & = & \langle \mathbf{stop}, e_c \cup e'_c, e_a \cup e'_a\rangle_\omega \\
\pi_s^{ic}(f|_K g) & = & \bot \text{ otherwise} \\
\pi_s^{ic}(f >x> g) & = & \begin{cases} \pi_s^{ic}(f) >x> g & \text{if } \pi_s^{ic}(f) \neq \bot \\ \bot & \text{otherwise} \end{cases} \\
\pi_s^{ic}(f <x< g) & = & \begin{cases} \pi_s^{ic}(f) <x< \pi_s^{ic}(g) & \text{if } \pi_s^{ic}(p) \neq \bot \neq \pi_s^{ic}(p') \\ \bot & \text{otherwise} \end{cases} \\
\pi_s^{ic}(f <x, K_0< g) & = & \pi_s^{ic}(f <x< g) \\
\pi_s^{ic}(f <x, e \triangleright K_0< g) & = & \langle \mathbf{stop}, e_c, e_a\rangle_\omega <x< \pi_s^{ic}(g) \text{ if } \pi_s^{ic}(f) \neq \bot \\
\pi_s^{ic}(f <x, K_0 \triangleleft e< g) & = & \pi_s^{ic}(f) <x< \langle \mathbf{stop}, e_c, e_a\rangle_\omega \text{ if } \pi_s^{ic}(g) \neq \bot \\
\pi_s^{ic}(f <x, e \triangleright K_0 \triangleleft e'< g) & = & \langle \mathbf{stop}, e_c \cup e'_c, e_a \cup e'_a\rangle_\omega \\
\pi_s^{ic}(f <x, K< g) & = & \bot \text{ otherwise} \\
\pi_s^{ic}(f; g) & = & \pi_s^{ic}(f); \pi_s^{ic}(g) \\
\pi_s^{ic}(f;_\emptyset\ g) & = & \begin{cases} \pi_s^{ic}(g) & \text{if } \pi_s^{ic}(f) = \bot \\ \pi_s^{ic}(f); \pi_s^{ic}(g) & \text{otherwise} \end{cases} \\
\pi_s^{ic}(f;_K\ g) & = & \pi_s^{ic}(f)
\end{array}
$$

$\pi_s^{ic}(?k)$ is not defined and the proof will not be done for NTRES, TRES and NEGRES, which are correct by hypothesis.

We now give the full definition of non-conflictuality. The non-conflictual parameters are of the form:

- $\mathbf{def}\ y(x) = f$, where $f$ is non-conflictual
- $\langle p, c, a, b\rangle_L$ where $p$ is non-conflictual
- $V$, $\mathbf{stop}$, $x$
- $x_{\{p\}}$ where $p$ is non-conflictual

A program $F$ is non-conflictual if it is of the form:

- $?k$, $\bot$,
- $p$, $P(p)$ where $p$ and $P$ are non-conflictual
- $f \setminus k$, $\langle f, c, a, \emptyset\rangle_L$ where $f$ is non-conflictual
- $f|g$, $f|_{K_0}g$, $f >x> g$, $f <x< g$, $f <x, K_0< g$, $f; g$, $f;_K g$, $\mathbf{def}\ y(x) = f\#g$, where $f$ and $g$ are non-conflictual
- $f|_{K_3}g$, $f <x, K_3< g$ where $\pi_i^{ic}(f) = \pi_i^{ic}(g) = \bot$
- $f|_{K_1^3}g$, where $f$ is non-conflictual and $\pi_i^{ic}(g) = \bot$

We will now show by induction on $n$ that for all execution $f = f_0 \xrightarrow{\sigma[1]}_{ic} f_1 \ldots \xrightarrow{\sigma[n]}_{ic} f_n$ such that for all $i$, $\sigma[i]_b = \emptyset$, $f_i$ is non-conflictual and $pi_s^{ic}(f_i) = pi_s^{ic}(f_{i-1})$ if $\sigma[i]_l = \varepsilon$ and $pi_s^{ic}(f_{i-1}) \xrightarrow{\sigma[i]_l} pi_s^{ic}(f_i)$ otherwise (denoted $pi_s^{ic}(f_{i-1}) \xRightarrow{\sigma[i]_l} pi_s^{ic}(f_i)$).

It is true for $n = 0$ as $f \in Orc$. Suppose the property holds for a given $n$. This is a corollary of the following property: for all $F \xrightarrow{e}_{ic} F'$ such that $e_b = \emptyset$ and $F$ is non-conflictual, $pi_s^{ic}(F) \stackrel{e_l}{\Rightarrow} pi_s^{ic}(F')$. This is done by induction on the derivation tree of the step. The rule that generated it can be:

**PUBLISH (idem for STOP, INTCALL) :** we have $v \xrightarrow{k,!v,\emptyset,\emptyset,\emptyset}_{ic} \langle \mathbf{stop}, \{k\}, \emptyset, \emptyset\rangle$ and $v \xrightarrow{!v} \mathbf{stop}$ with $\langle \mathbf{stop}, \{k\}, \emptyset, \emptyset\rangle$ non-conflictual.

**DEFDECLARE :** suppose $D\#f \xrightarrow{k,l,c,a,\emptyset}_{ic} f'$ where $D\#f$ is non-conflictual. $[D/y]f$ is non-conflictual and $[D/y]f \xrightarrow{k,l,c,a,\emptyset}_{ic} f'$. By induction, $f'$ is non-conflictual and $\pi_s^{ic}([D/y]f) = [\pi_s^{ic}(D)/y]\pi_s^{ic}(f) \xrightarrow{l} \pi_s^{ic}(f')$, so we can apply rule DEFDECLARE, and $\pi_s^{ic}(D\#f) = \pi_s^{ic}(D)\#\pi_s^{ic}(f) \xrightarrow{l} \pi_s^{ic}(f')$.

**VAR :** $F$ is not conflictual, so $F = x_{\{p\}}$ and $\pi_s^{ic}(F) = \pi_s^{ic}(p)$. By induction, $\pi_s^{ic}(p) \xrightarrow{l} \pi_s^{ic}(p')$. Moreover, $p'$ is non-conflictual so $x_{\{p'\}}$ is non-conflictual.

**CALLSTART :** we have $P \xrightarrow{k,!v,c,a,\emptyset}_{ic} P'$, and $P$ is non-conflictual, so $P = x_{\langle \ldots \langle v,c',a',\emptyset\rangle_L \ldots\rangle_{L'}}$. We have that $g = (P'(p) \setminus k)|\langle v(p), c, a, b\rangle_l$ is non-conflictual and $\pi_s^{ic}(P(p)) = \pi_s^{ic}(g) = v(\pi_s^{ic}(p))$.

**CALLSTOP:** we have $P \xrightarrow{k,\omega,c,a,\emptyset}_{ic} P'$, and $P$ is non-conflictual, so $P$ is either **stop** or $x_{\langle \ldots \langle \mathbf{stop},c',a',\emptyset\rangle_L \ldots\rangle_{L'}}$ and $\pi_s^{ic}(P(p)) = \mathbf{stop}(\pi_s^{ic}(p))$. Moreover, $P'$ is **stop** or $x_{\langle \ldots \langle \bot,c',a',\emptyset\rangle_L \ldots\rangle_{L'}}$ so $P'(p)$ is non-conflictual. We can apply axiom STOPCALL, and $\pi_s^{ic}(P(p)) \xrightarrow{\omega} \bot = \pi_s^{ic}(P'(p))$.

**CALLE:** by induction, $P'$ is not conflictual and $\pi_s^{ic}(P') = \pi_s^{ic}(P)$, so $P'(p)$ is not conflictual and $\pi_s^{ic}(P'(p)) = \pi_s^{ic}(P(p))$.

**EXTE:** by induction, $p'$ is not conflictual and $\pi_s^{ic}(p') = \pi_s^{ic}(p)$, so $V(p')$ is not conflictual and $\pi_s^{ic}(V(p')) = \pi_s^{ic}(V(p))$.

**EXTCALL (same reasoning for EXTSTOP):** we have $\pi_s^{ic}(p) = v$ and by induction, $p'$ is non-conflictual and $\pi_s^{ic}(p') = \mathbf{stop}$. $g = (V(p') \setminus k)|\langle ?k, c \cup \{k\}, a, b\rangle_l$ is non-conflictual and EXTCALL can be applied on $\pi_s^{ic}(V(p)) = V(v)$, so $\pi_s^{ic}(V(p)) \xrightarrow{?V_k(v)} ?k = \pi_s^{ic}(g)$.

**OTHERSTART:** $\pi_s^{ic}(f;_\emptyset g) = \pi_s^{ic}(f; g)$, so the result is true by induction.

**OTHERN :** $f$ is non-conflictual and $\pi_s^{ic}(f) \neq \bot$. By induction, we have $\pi_s^{ic}(f) \xrightarrow{n} \pi_s^{ic}(f')$ and $f'$ non-conflictual. If $K = \emptyset$, it is the point, otherwise, we can apply OTHERN, so $\pi_s^{ic}(f;_K g) = \pi_s^{ic}(f);\pi_s^{ic}(g) \xrightarrow{n} \pi_s^{ic}(f');\pi_s^{ic}(g) = \pi_s^{ic}(f';_K g)$.

**OTHERV :** $f$ is non-conflictual and $\pi_s^{ic}(f) \neq \bot$. By induction, we have $\pi_s^{ic}(f) \xrightarrow{!v} \pi_s^{ic}(f')$ and $f'$ non-conflictual. If $K = \emptyset$, it is the point, otherwise, we can apply OTHERV, so $\pi_s^{ic}(f;_K g) = \pi_s^{ic}(f);\pi_s^{ic}(g) \xrightarrow{!v} \pi_s^{ic}(f') = \pi_s^{ic}(f';_{K\cup\{k\}} g)$.

**OTHERSTOP :** $f$ is non-conflictual and $\pi_s^{ic}(f) \neq \bot$. By induction, $\pi_s^{ic}(f) \xrightarrow{\omega} \bot = \pi_s^{ic}(f')$ and $f'$ is non-conflictual. If $K \cap c = \emptyset$, we can apply rule OTHERSTOP, so $\pi_s^{ic}(f;_\emptyset g) = \pi_s^{ic}(f);\pi_s^{ic}(g) \xrightarrow{h(\omega)} \pi_s^{ic}(g) = \pi_s^{ic}((f';g \setminus k)|\langle g, c \cup \{k\}, a, b\rangle_l)$. Otherwise, $\pi_s^{ic}(f;_K g) = \pi_s^{ic}(f)$, so the result is true.

**OTHERHALT :** $K \neq \emptyset$, so $\pi_s^{ic}(f;_K g) = \pi_s^{ic}(f)$, and the result is true by induction.

**SEQN :** $f >x> g$ is non-conflictual, so is $f$. By induction, we have $\pi_s^{ic}(f) \xrightarrow{l} \pi_s^{ic}(f')$ and $f'$ non-conflictual. If $l = n$, we can apply rule SEQN, so $\pi_s^{ic}(f >x> g) = \pi_s^{ic}(f) >x> g \xrightarrow{l} \pi_s^{ic}(f') >x> g = \pi_s^{ic}(f' >x> g)$ and $f' >x> g$ non-conflictual. Otherwise, $l = \omega$, and $\pi_s^{ic}(f') = \bot$. We can apply rule SEQSTOP, and $\pi_s^{ic}(f >x> g) = \pi_s^{ic}(f) >x> g \xrightarrow{\omega} \bot = \pi_s^{ic}(f' >x> g)$.

**SEQV :** $f >x> g$ is non-conflictual, so is $f$. By induction, we have $\pi_s^{ic}(f) \xrightarrow{!v} \pi_s^{ic}(f')$ and $f'$ non-conflictual. We can apply rule SEQV, so $\pi_s^{ic}(f >x> g) = \pi_s^{ic}(f) >x> g \xrightarrow{v} \pi_s^{ic}(f') >x> g|[v/x]g = \pi_s^{ic}(f' >x> g|\langle [v/x]g, c \cup \{k\}, a, b\rangle_l)$.

**PARSTART:** $\pi_s^{ic}(f|_{(\emptyset,\emptyset)}g) = \pi_s^{ic}(f|g)$, so the result is true by induction.

**PARLEFTSTOP (same reasoning for PARRIGHTSTOP) :** $f$ is non-conflictual, so by induction, $\pi_s^{ic}(f) \xrightarrow{\omega} \pi_s^{ic}(f')$ and we have $\pi_s^{ic}(f) \equiv \mathbf{stop}$ and $\pi_s^{ic}(f') = \bot$. $f'|_{(c,a,b)\triangleright K}g$ is non-conflictual, and $\pi_s^{ic}(f'|_{(c,a,b)\triangleright K}g) = \mathbf{stop}|\pi_s^{ic}(g) \equiv \pi_s^{ic}(f)|\pi_s^{ic}(g) = \pi_s^{ic}(f|g)$.

**PARLEFT (same reasoning for PARRIGHT) :** $f$ is non-conflictual, so by induction, $\pi_s^{ic}(f) \xrightarrow{l} \pi_s^{ic}(f')$. Moreover, $K_L = \emptyset$. If $\pi_s^{ic}(g) = \bot$, we have directly $\pi_s^{ic}(f|_K g) = \pi_s^{ic}(f) \xrightarrow{l} \pi_s^{ic}(f') = \pi_s^{ic}(f'|_K g)$. Otherwise, we can apply rule PARLEFT, and $\pi_s^{ic}(f|_K g) =\xrightarrow{l} \pi_s^{ic}(f'|_K g)$.

**PARMID :** as $K \xrightarrow{k,\omega,c,a,b}_{ic} K'$, $\pi_s^{ic}(f|_K g) = \mathbf{stop}$, and as $f|_K g$ is non-conflictual, $K'$ is inert, so $\pi_s^{ic}(f|_{K'} g) = \bot$. We can apply rule STOP, which gives $\pi_s^{ic}(f|_K g) = \mathbf{stop} \xrightarrow{\omega} \bot = \pi_s^{ic}(f|_{K'} g)$.

**PRUNESTART :** $\pi_s^{ic}(f <x, (\emptyset,\emptyset)< g) = \pi_s^{ic}(f <x< g)$, so the result is true by induction.

**PRUNELEFT :** $f$ is non-conflictual, so by induction, $\pi_s^{ic}(f) \xrightarrow{l} \pi_s^{ic}(f')$. Moreover, $K_L = \emptyset$. If $K_R \neq \emptyset$, we have directly $\pi_s^{ic}(f <x, K< g) = \pi_s^{ic}(f) \xrightarrow{l} \pi_s^{ic}(f') = \pi_s^{ic}(f' <x, K< g)$. Otherwise, we can apply rule PRUNELEFT, and $\pi_s^{ic}(f <x, K< g) =\xrightarrow{l} \pi_s^{ic}(f' <x, K< g)$.

**PRUNELEFTSTOP :** $f$ is non-conflictual, so by induction, $\pi_s^{ic}(f) \xrightarrow{\omega} \pi_s^{ic}(f')$ and we have $\pi_s^{ic}(f) \equiv \mathbf{stop}$ and $\pi_s^{ic}(f') = \bot$. $f' <x, (c,a,b) \triangleright K< g$ is non-conflictual, and $\pi_s^{ic}(f' <x, (c,a,b) \triangleright K< g) = \mathbf{stop} <x< \pi_s^{ic}(g) \equiv \pi_s^{ic}(f) <x< \pi_s^{ic}(g) = \pi_s^{ic}(f <x, K< g)$.

**PRUNEMID :** as $K \xrightarrow{k,\omega,c,a,b}_{ic} K'$, $\pi_s^{ic}(f <x, K< g) = \mathbf{stop}$, and as $f|_K g$ is non-conflictual, $K'$ is inert, so $\pi_s^{ic}(f <x, K'< g) = \bot$. We can apply rule STOP, which gives $\pi_s^{ic}(f <x, K< g) = \mathbf{stop} \xrightarrow{\omega} \bot = \pi_s^{ic}(f <x, K'< g)$.

**PRUNESTOP :** $g$ is non-conflictual, so by induction, $\pi_s^{ic}(g) \xrightarrow{\omega} \pi_s^{ic}(g')$ and we have $\pi_s^{ic}(g) \equiv \mathbf{stop}$ and $\pi_s^{ic}(g') = \bot$. $h = [x|\langle \mathbf{stop}, c \cup \{k\}, a, b\rangle_l/x]f <x, K \triangleleft (c,a,b)< g'$ is non-conflictual, and by rule PRUNESTOP, we have $\pi_s^{ic}(f <x, K< g) \equiv \pi_s^{ic}(f) <x< \mathbf{stop} \xrightarrow{h(\omega)} [\mathbf{stop}/x]\pi_s^{ic}(f) = \pi_s^{ic}(h)$.

**PRUNEN :** $g$ is non-conflictual, so by induction, $\pi_s^{ic}(g) \xrightarrow{n} \pi_s^{ic}(g')$. Moreover, $K_R = \emptyset$. Otherwise, we can apply rule PRUNEN, and $\pi_s^{ic}(f <x, K< g) =\xrightarrow{l} \pi_s^{ic}(f <x, K< g')$.

**PRUNEV :** $g$ is non-conflictual, so by induction, $\pi_s^{ic}(g) \xrightarrow{!v} \pi_s^{ic}(g')$ with $\pi_s^{ic}(g) \neq \bot$. Let $h = [x|\langle v, c \cup \{k\}, a, b\rangle_l/x]f <x, K \triangleleft (c,a,b)< \langle g', \emptyset, \{k\}, \{k\}\rangle_{!v}$. By rule PRUNEV, we have $\pi_s^{ic}(f <x, K< g) = \pi_s^{ic}(f) <x< \pi_s^{ic}(g) \xrightarrow{h(!v)} [v/x]\pi_s^{ic}(f) = \pi_s^{ic}(h)$.

**DISTLEFT, DISTRIGHT :** the left hand side of the rule does not match a valid program

**CAUSEYES, CAUSENO :** $\langle f, c, a, b\rangle_l$ is non-conflictual, so is $f$. By induction, $\pi_s^{ic}(\langle f, c, a, b\rangle_l) = \pi_s^{ic}(f) \xrightarrow{l} \pi_s^{ic}(f') = \pi_s^{ic}(\langle f', c, a, b\rangle_l)$ and $\pi_s^{ic}(\langle f', c, a, b\rangle_l)$ is non-conflictual.

**HIDE :** $\sigma[n+1]_b \neq \emptyset$.

Finally, we were able to create an execution labelled by $\sigma|_l$ dependless of the size of $\sigma$, so $\sigma|_l \in [\![f_0]\!]$. □

**Property 1** (Instrumentation). $\forall f \in \mathcal{O}, ([\![f]\!]_i)\,|_l = [\![f]\!]$

*Proof.* Let $\sigma_i \in [\![f]\!]_i$. By lemmas 2 and 3, there is $\sigma_{ic} \in [\![f]\!]_{ic}$ such that $\sigma_i|_l = \sigma_{ic}|_l \in [\![f]\!]$. Then $([\![f]\!]_i)\,|_l \subset [\![f]\!]$

Let $\sigma \in [\![f]\!]$. By lemma 1, there is $\sigma_i \in [\![f]\!]_i$ such that $\sigma_i|_l = \sigma$. Then $[\![f]\!] \subset ([\![f]\!]_i)\,|_l$. □

**Lemma 4.** *Let $\gamma ab\delta \in [\![f_0]\!]^{\omega}_{ic}$ such that $a \not\leq b$.*

*There are $a'$, $b'$ with $a'_k = a_k$ and $b'_k = b_k$ such that the identity on $k$ defines an injection from $\overline{\overline{\gamma ab\delta}}$ to $\overline{\overline{\gamma b' a' \delta}}$.*

*Proof.* Let us consider two consecutive causally-independant steps $f \xrightarrow{k_1,l_1,c_1,a_1,b_1}_{ic} f'_1 \xrightarrow{k'_1,l'_1,c'_1,a'_1,b'_1}_{ic} f''_1$. We prove that there are programs $f''_2$ and $f'_2$ and sets of identifiers $a_2, b_2, a'_2, b'_2$ such that $f \xrightarrow{k'_1,l'_1,c'_1,a'_2,b'_2}_{ic} f'_2 \xrightarrow{k_1,l_1,c_1,a_2,b_2}_{ic} f''_2$ and $f''_2$ is equivalent to $f''_1$ in which $a_1, b_1, a'_1, b'_1$ have been replaced respectively by $a_2, b_2, a'_2, b'_2$, with:

$$a_2, b_2, a'_2, b'_2 = \begin{cases} a_1, b_1, a'_1, b'_1 & \text{if} \quad k \notin a' \cup b' \\ a_1, b_1 \cup \{k'\}, a'_1 \setminus \{k\}, b'_1 & \text{if} \quad k \in a' \setminus b' \\ a_1 \cup \{k'\}, b_1, a'_1, b'_1 \setminus \{k\} & \text{if} \quad k \in b' \setminus a' \\ a_1 \cup \{k'\}, b_1 \cup \{k'\}, a'_1 \setminus \{k\}, b'_1 \setminus \{k\} & \text{if} \quad k \in b' \cup a' \end{cases}$$

This is shown by induction on the derivation tree of the first step, and there is no other choice than the complete enumeration of the possible pairs of following steps. The items of first level are the possibilities for the rule of the first step, and those of second level are the possibilities for the second step.

**PUBLISH, STOP, INTCALL :** there is no causaly-independant possible second step.

**DEFDECLARE** $f = D\#g \xrightarrow{k_1,l_1,c_1,a_1,b_1}_{ic} f_1'$ $\xrightarrow{k_1',l_1',c_1',a_1',b_1'}_{ic} f_1''$. We have $[D/y]g \xrightarrow{k_1,l_1,c_1,a_1,b_1}_{ic} f_1' \xrightarrow{k_1',l_1',c_1',a_1',b_1'}_{ic} f_1''$. By induction, $[D/y]g \xrightarrow{k_1',l_1',c_1',a_2',b_2'}_{ic} f_2' \xrightarrow{k_1,l_1,c_1,a_2,b_2}_{ic} f_2''$, and we can apply DEFDECLARE on the first step.

**VAR** the second rule is also VAR, so we can apply VAR twice on the premises given by induction.

**CALLSTART (idem CALLSTOP)** $f = P(p)$

- **CALLSTART** by induction, $p \xrightarrow{k_1',v_1',c'^p_1,a'^p_2,b'^p_2} g_2' \xrightarrow{k_1,v_1,c_1^p,a_2^p,b_2^p} g_2''$, and we can apply CALLSTART for the first step and rules PARLEFT, HIDE and CALLSTOP for the second step which gives $f_2'' = ((P_2'(p) \setminus k_1)|\langle v_1(p), c_1^p \cup \{k_1\}, a_2^p, b_2\rangle_l \setminus k_1')|\langle v_1'(p), c'^p_1 \cup \{k_1'\}, a'^p_2, b'_2\rangle_l \equiv f_1''$ because $\langle v_1(p), c_1^p \cup \{k_1\}, a_2^p, b_2\rangle_l \setminus k_1' \equiv \langle v_1(p), c_1^p \cup \{k_1\}, a_2^p, b_2\rangle_l$.
- **CALLSTOP** by induction, $p \xrightarrow{k_1',v_1',c'^p_1,a'^p_2,b'^p_2} g_2' \xrightarrow{k_1,v_1,c_1^p,a_2^p,b_2^p} g_2''$, and we can apply CALLSTOP the first step and CALLSTART for the second step which gives $f_2'' = (P_2'(p) \setminus k_1')|\langle v_1'(p), c'^p_1 \cup \{k_1'\}, a'^p_2, b'_2\rangle_l \equiv f_1''$.

**EXTCALL, EXTE, EXTSTOP** $f = V(p)$ these case are similar to CALLSTART and CALLSTOP where EXTCALL has the same role as CALLSTART.

**NTRES, TRES, NEGRES :** we trust the external sites on this point.

**OTHERSTART** By induction, $g_\emptyset h \xrightarrow{k_1',l_1',c_1',a_2',b_2'}_{ic} f_2' \xrightarrow{k_1,l_1,c_1,a_2,b_2}_{ic} f_2''$, and we can apply OTHERSTART on the first step.

**OTHERN (idem OTHERV, OTHERHALT, OTHERSTOP)** $f = g; h \xrightarrow{k_1,n_1,c_1,a_1,b_1} f_1' = g_1'; h$

- **OTHERN (idem OTHERV, OTHERHALT)** $f_1' \xrightarrow{k_1',n_1',c_1',a_1',b_1'} f_1' = g_1''; h$. By induction, $g \xrightarrow{k_1',n_1',c_1',a_2',b_2'} g_2' \xrightarrow{k_1,n_1,c_1,a_2,b_2} g_2''$, and we can apply the same rules.
- **OTHERSTOP** $f_1' \xrightarrow{k_1',!v_1',c_1',a_1',b_1'} f_1' = g_1'' >x> h$. By induction, $g \xrightarrow{k_1',v_1',c_1',a_2',b_2'} g_2' \xrightarrow{k_1,n_1,c_1,a_2,b_2} f_1' = g_2''$, and we can apply OTHERSTOP first with $f_2' = (g_2'; h)|\langle h, c \cup \{k\}, a, b\rangle_l$ where we can apply PARLEFT or PARLEFTSTOP and the same rule as for the first step.

**SEQN (idem SEQV)** $f = g >x>, h \xrightarrow{k_1,n_1,c_1,a_1,b_1} f_1' = g_1' >x> h$

- **SEQN** $f_1' \xrightarrow{k_1',n_1',c_1',a_1',b_1'} f_1' = g_1'' >x> h$. By induction, $g \xrightarrow{k_1',n_1',c_1',a_2',b_2'} g_2' \xrightarrow{k_1,n_1,c_1,a_2,b_2} g_2''$, and we can apply SEQN twice.
- **SEQV** $f_1' \xrightarrow{k_1',!v_1',c_1',a_1',b_1'} f_1' = g_1'' >x> h$. By induction, $g \xrightarrow{k_1',v_1',c_1',a_2',b_2'} g_2' \xrightarrow{k_1,n_1,c_1,a_2,b_2} g_2''$, and we can apply SEQV first with $f_2' = (g' >x> h)|\langle [v_1'/x]h, c_1' \cup \{k_1'\}, a_1', b_1'\rangle_l$ where we can apply PARLEFT or PARLEFTSTOP and SEQN.

**PARSTART** By induction, $g|_{(\emptyset,\emptyset)}h \xrightarrow{k_1',l_1',c_1',a_2',b_2'}_{ic} f_2' \xrightarrow{k_1,l_1,c_1,a_2,b_2}_{ic} f_2''$, and we can apply PARSTART on the first step.

**PARLEFT (idem for PARRIGHT, PARLEFTSTOP, PARRIGHTSTOP, PARMID)** $f = g|_K h \xrightarrow{k,c,a,b} f_1' = g'|_K h$

- **PARRIGHT (idem PARMID, PARRIGHTSTOP)** We can apply PARRIGHT on the premise of the second step to build the first step and then PARLEFT on the premise of the first step.
- **PARLEFT (idem PARLEFTSTOP)** We can applyinduction on the left hand side and apply PARLEFT twice.

**PRUNESTART** By induction, $g <x, (\emptyset, \emptyset)< h \xrightarrow{k_1',l_1',c_1',a_2',b_2'}_{ic} f_2' \xrightarrow{k_1,l_1,c_1,a_2,b_2}_{ic} f_2''$, and we can apply PRUNESTART on the first step.

**PRUNELEFT (idem for PRUNELEFTSTOP)** $f = g <x, K<, h \xrightarrow{k,c,a,b} f_1' = g' <x, K< h$

- **PRUNELEFT (idem for PRUNELEFTSTOP)** We can apply the induction on the premises and PRUNELEFT twice.
- **PRUNEN (idem for PRUNEMID, PRUNEE)** We have $g' <x, K< h \xrightarrow{k',c',a',b'} f_1'' = g' <x, K< \langle h', \emptyset, \{k'\}, \emptyset\rangle_{!v}$. The same rules can be applied in reverse order, which gives $f = g <x, K<, h \xrightarrow{k',c',a',b'} f_2' = g <x, K< \langle h', \emptyset, \{k'\}, \emptyset\rangle_{!v} \xrightarrow{k,c,a,b} f_2'' = f_1''$.

**PRUNEV (idem for PRUNESTOP)** the premise of the second step is indepentant of the premise of the first step, so it can be applied, and we have $f_2' = [\langle !v_1', c_1' \cup \{k_1'\}, a_2', b_2'\rangle_l//x]g < x, K \triangleleft (c_1', a_2', b_2') < \langle h' \setminus k_1', \emptyset, \{k_1'\}, \{k_1'\}\rangle_{!v}$. We also know that $g \xrightarrow{k_1,c_1,a_1,b_1} g'$. All the transitions that are possible from $g$ are also possible from $[\langle !v_1', c_1' \cup \{k_1'\}, a_2', b_2'\rangle_l//x]g$, so we can apply PRUNELEFT on it for the second step.

**PRUNEMID** $f = g <x, K<, h \xrightarrow{k,c,a,b} f_1' = g <x, K'< h$

**PRUNEMID** we can apply the induction on the premises and apply PRUNEMID twice.

**anything else** the steps are totally independant.

**DISTLEFT, DISTRIGHT :** the two steps concern different elements, so they can occure in any order.

**HIDE (idem for CAUSEYES, CAUSENO) HIDE :** We can apply the induction for the premise and apply the same rules.

The full lemma is just a corollary. □

**Lemma 5.** *Let $\gamma \in [\![f]\!]_{ic}$ and two events $e_1$ and $e_2$ such that $\gamma.e_1, \gamma.e_2 \in [\![f]\!]_{ic}$.*
*There is an event $e_2'$ such that $\gamma.e_1.e_2' \in [\![f]\!]_{ic}$ and $\overline{\overline{\gamma.e_2}} < \overline{\overline{\gamma.e_1.e_2'}}$.*

*Proof.* This lemma is a kind of converse of the previous one. What we realy prove is that for any $f \in Orc_{ic}$ and any transitions $f \xrightarrow{k_1,c_1,a_1,b_1}_{ic} f_1'$ and $f \xrightarrow{k_2,c_2,a_2,b_2}_{ic} f_2'$, there is a transition $f_1' \xrightarrow{k_2,c_2,a_2',b_2'}_{ic} f'$ with $a_2' \in \{a_2, a_2 \cup \{k_1\}\}$ and $b_2' \in \{b_2, b_2 \cup \{k_1\}\}$.

We introduce a few notations:

- if $T$ is the derivation tree of a transition, $p(T)$ is the premise if it has a unique one, $p_1(T)$ and $p_2(T)$ are respectively the left hand side and right hand side premises if they exist.
- if the derivation trees of the steps $k_1, c_1, a_1, b_1$ and $k_2, c_2, a_2, b_2$ are denoted $T_1$ and $T_2$, the derivation tree of $k_2, c_2, a_2', b_2'$ is denoted $\tau(T, T')$.
- If $R$ is a rule with one premise and $T$ is a transition, $R(T)$ is a transition such that $p(R(t)) = T$ and whom root is generated by rule $R$. $\tau(T, T') = R(T'')$ means that there is a transition $\tau(T, T')$ that the root is generated by $R$, has $f_1'$ as left hand side and $T''$ as premise. Similarly, we write $R(T, T')$ if $R$ has two premises.

Just like above, we prove the result by induction on the derivation tree of $(k_1, c_1, a_1, b_1)$. We should differentiate the base cases and the induction cases. However, by sake of clarity, we give a unique list of possibilities. The first-level items are for rules that can generate $(k_1, c_1, a_1, b_1)$ and the second-level items are the remaining possibilities for the other steps. Of course, we never use the induction hypothesis on case bases. Let $T$ and $T'$ be the transitions of the steps $f \xrightarrow{k_1,c_1,a_1,b_1}_{ic} f_1'$ and $f \xrightarrow{k_2,c_2,a_2,b_2}_{ic} f_2'$. The rules at the root of $T$ and $T'$ are respectively denoted $R$ and $R'$. Here are the possibilities for $R$:

**PUBLISH, STOP, INTCALL:** these cases are impossible as there is only one possible step, so $T'$ does not exist.

**DEFDECLARE:** $f = D\#g$, so $R' =$ DEFDECLARE. $p(T)$ and $p(T')$ are possible from $[D/y]f$, so by induction, $\tau(p(T), p(T'))$ can be fired after $p(T)$. As the premise and the conclusion of DEFDECLARE have the same right hand side, $\tau(p(T), p(T'))$ can be fired after $T$, so $\tau(T, T') = \tau(p(T), p(T'))$.

**VAR:** $f = x_{P \cup p}$, so $R' =$ VAR, and the left hand side of $p(T')$ is $p'$. There are two possibilities:

- $p = p'$**:** by induction, $\tau(p(T), p(T'))$ can be fired after $p(T)$, so $\tau(p(T), p(T')) =$ VAR$(\tau(p(T), p(T')))$.
- $p \neq p'$**:** the premises are independant, and $\tau(p(T), p(T')) =$ VAR$(p(T'))$.

**CALLSTART:** $f = P(p)$, where $P \neq v$. The possibilities for $R'$ are:

- **CALLSTART, CALLE:** $\tau(T, T') =$ PARLEFT(HIDE$(R'(\tau(p(T), p(T'))))$).
- **CALLSTOP:** $\tau(T, T') =$ PARLEFTSTOP(HIDE$(R'(\tau(p(T), p(T'))))$).

**CALLSTOP, CALLE:** $f = P(p)$, and $P \neq v$. The possibilities for $R'$ are CALLSTART, CALLE and CALLSTOP. In the cases $\tau(T, T') = R'(\tau(p(T), p(T')))$ works.

**EXTERIGHT:** $f = V(p)$. The possibilities for $R'$ are EXTERIGHT, EXTCALL and EXTSTOP, and $\tau(T, T') = R'(\tau(p(T), p(T')))$ always works.

**EXTSTOP:** $f = V(p)$. The possibilities for $R'$ are EXTERIGHT, EXTCALL and EXTSTOP, and $\tau(T, T') =$ HIDE$(R'(\tau(p(T), p(T'))))$ always works, as $k_2 \notin c_1$.

**EXTCALL:** $f = V(p)$. The possibilities for $R'$:

- **EXTERIGHT, EXTCALL:** $\tau(T, T') = \text{PARLEFT}(\text{HIDE}(R'(\tau(p(T), p(T')))))$
- **EXTSTOP:** $\tau(T, T') = \text{PARLEFTSTOP}(\text{HIDE}(R'(\tau(p(T), p(T')))))$

**RESNT, REST, RESNEG:** this management is left to the external sites

**OTHERSTART:** $R' = \text{OTHERSTART}$, and $\tau(T, T') = \tau(p(T), p(T'))$

**OTHERN, OTHERV, OTHERHALT:** $f = g ;_K h$, and $\tau(T, T') = R'(\tau(p(T), p(T')))$.

**OTHERSTOP:** $f = g ;_K h$, and $\tau(T, T') = \text{PARLEFT}(R'(\tau(p(T), p(T'))))$.

**PARSTART:** $R' = \text{PARSTART}$, and $\tau(T, T') = \tau(p(T), p(T'))$

**PARLEFTSTOP (idem for PARRIGHTSTOP):** the possibilities for $R'$ are:

- **PARLEFTSTOP (idem for PARLEFT):** $\tau(T, T') = \text{PARLEFTSTOP}(\tau(p(T), p(T')))$.
- **PARRIGHTSTOP (idem for PARRIGHT):** $\tau(T, T') = \text{PARRIGHTSTOP}(p(T'))$.
- **PARMID:** $\tau(T, T') = \text{PARMID}(p(T'))$. This application is still possible as $K'$ contains more information than $K$.

**PARLEFT (idem for PARRIGHT):** the possibilities for $R'$ are:

- **PARLEFTSTOP (idem for PARLEFT):** $\tau(T, T') = \text{PARLEFTSTOP}(\tau(p(T), p(T')))$.
- **PARRIGHTSTOP (idem for PARRIGHT):** $\tau(T, T') = \text{PARRIGHTSTOP}(p(T'))$.
- **PARMID:** $\tau(T, T') = \text{PARMID}(p(T'))$.

**PARMID:** the possibilities for $R'$ are:

- **PARMID:** $\tau(T, T') = \text{PARMID}(p(T'))$. This application is still possible because $T \neq T'$.
- **Anything else:** $\tau(T, T') = R'(p(T'))$.

**SEQN:** $f = g >x> h$, and $\tau(T, T') = R'(\tau(p(T), p(T')))$.

**SEQV:** $f = g >x> h$, and $\tau(T, T') = \text{PARLEFT}(R'(\tau(p(T), p(T'))))$.

**PRUNESTART:** $R' = \text{PRUNESTART}$, and $\tau(T, T') = \tau(p(T), p(T'))$.

**PRUNELEFT, PRUNELEFTSTOP:** the possibilities for $R'$ are:

- **PRUNELEFT, PRUNELEFTSTOP:** $\tau(T, T') = R'(\tau(p(T), p(T')))$.
- **Anything else:** $\tau(T, T') = R'(p(T'))$.

**PRUNEMID:** the possibilities for $R'$ are:

- **PRUNEMID:** $\tau(T, T') = \text{PRUNEMID}(p(T'))$. This application is still possible because $T \neq T'$.
- **Anything else:** $\tau(T, T') = R'(p(T'))$.

**PRUNESTOP:** the possibilities for $R'$ are:

- **PRUNELEFT, PRUNELEFTSTOP:** $f = g <x, K< h$ and $p(T')$ is a valid step from $g$. The only difference between $g$ and $g' = [\langle \mathbf{stop}, c_1 \cup \{k_1\}, a_1, b_1 \rangle // x]g$ is the set of values associated to the variable $x$. If VAR appears in $p(T')$ with $x$ in its left hand side, it can also be applied on the completed version of $x$, so a transition $T''$ with the same structure as $p(T')$ can be build with $g'$ at the left hand side instead of $g$. $\tau(T, T') = R'(T'')$ works.
- **PRUNEMID:** $\tau(T, T') = R'(p(T'))$.
- **Anything else:** $\tau(T, T') = R'(\tau(p(T), p(T')))$.

**PRUNEE:** the possibilities for $R'$ are:

- **PRUNELEFT, PRUNELEFTSTOP, PRUNEMID:** $\tau(T, T') = R'(p(T'))$.
- **Anything else:** $\tau(T, T') = R'(\tau(p(T), p(T')))$.

**PRUNEN:** the possibilities for $R'$ are:

- **PRUNELEFT, PRUNELEFTSTOP, PRUNEMID:** $\tau(T, T') = R'(p(T'))$.
- **PRUNEV:** $\tau(T, T') = \text{PRUNEV}(\text{CAUSEYES}(\tau(p(T), p(T'))))$.
- **Anything else:** $\tau(T, T') = R'(\text{CAUSENO}(\tau(p(T), p(T'))))$.

**PRUNEV:** the possibilities for $R'$ are:

- **PRUNELEFT, PRUNELEFTSTOP:** $\tau(T, T') = R'(T'')$, where $T''$ is defined exactly like for PRUNESTOP.
- **PRUNEMID:** $\tau(T, T') = R'(p(T'))$.

**PRUNEV:** $\tau(T,T') = \text{PRUNEV}(\text{CAUSEYES}(\text{HIDE}(\tau(p(T),p(T')))))$. HIDE is possible because $k_2 \notin c_1$.

**Anything else:** $\tau(T,T') = R'(\text{CAUSENO}(\text{HIDE}(\tau(p(T),p(T')))))$.

**DISTLEFT, DISTRIGHT:** $f = K$, so $R'$ is also DISTLEFT or DISTRIGHT. These rules just remove different elements in the structure, so they can be applied in any order.

**CAUSEYES, CAUSENO:** $R'$ is CAUSEYES or CAUSENO and $\tau(T,T') = R'(\tau(p(T),p(T')))$.

**HIDE:** $f = g \setminus k''$, so $R' = \text{HIDE}$ and $\tau(T,T') = R'(\tau(p(T),p(T')))$.

□

**Property 2** (Fair completion)**.** For all $\gamma \in [\![f]\!]_{ic}$, there is a (possibly infinite) word of events $\sigma$ such that $\gamma.\sigma \in [\![f]\!]^{\omega}_{ic}$.

*Proof.* The idea of this proof is to use a first-in-first-out list of possible steps in which we push the transitions as soon as they become available. Once $\gamma$ has been executed, we fire the steps in the order where they are poped from the list. As the list remains finite, every event will eventually be fired.

We build a sequence $(\tau_n, f_n, L_n)_{n\in\mathbb{N}}$ such that, for all $n$:

- $\tau_n \in [\![f]\!]_{ic}$. Informally, it is the prefixe of size $n$ of the fair execution we are building. We expect, for all $n \geq 1$, $\tau_{n-1} = \tau_n[1...n]$ and $\tau_{|\gamma|} = \gamma$;
- $f_n \in Orc_{ic}$. It is the expression obtained after $\tau_n$, i.e. for all $n$, $f = f_0 \xrightarrow{\tau_1[1]} f_1 ... \xrightarrow{\tau_n[n]} f_n$.
- $L_n$ is an ordered list of transitions (i.e. there derivation trees) that contains exactly the set of transitions that are valid in the instrumented semantics and that have $f_n$ as the left-hand side of the conclusion.

The sequence is built by induction on $n$. We initialize as follows: $\tau_0$ is the empty execution, $f_0 = f$ and $L_0$ contains all the transitions starting from $f$. Suppose we have built $(\tau_n, f_n, L_n)$ for a given $n$. We chose a transition $T$ in $L_n$ as:

- if $n < |\gamma|$, we have $f = f_0 \xrightarrow{\gamma[1]} f_1 ... \xrightarrow{\gamma[n]} f_n$ and there is $g$ such that $f_n \xrightarrow{\gamma[n+1]} g$. As all the transitions available from $f_n$, $f_n \xrightarrow{\gamma[n+1]} g$ is contained into $L_n$, and it defines $T$.
- if $L_n$ is not empty, we take the first transition for $T$
- otherwise, $\tau_n$ is fair. We will consider this case later.

The conclusion of $T$ is $f_n \xrightarrow{e} g$. We define $f_{n+1} = g$ and $\tau_{n+1} = e$. $L_{n+1}$ is built as follows:

- $L'_n$ is $L_n$ in which $T$ was removed.
- $L''_n$ is a list of the same size as $L'_n$. For all $i$, we have $\tau_n.T \in [\![f]\!]_{ic}$ and $\tau_n.L'_n[i] \in [\![f]\!]_{ic}$. By lemma 5, there is an event $T'$ such that $\gamma.T.T' \in [\![f]\!]_{ic}$ and $\overline{\overline{\gamma.L'_n[i]}} < \overline{\overline{\gamma.T.T'}}$. We pose $L''_n[i] = T'$.
- $L'''_n$ can be any ordered list that contains all the transitions that can be fired from $f_{n+1}$ and are absent from $L''_n$.
- $L_{n+1}$ is the concatenation of $L''_n$ and $L'''_n$.

This defines the sequence by induction.

There are two cases:

- if there is $n \geq |\gamma|$ such that $L_n$ is empty, we pose $\tau = \tau_n$;
- otherwise, for all $n \in \mathbb{N}$, we pose $\tau[n] = \tau_n[n]$.

In both cases, $\gamma$ is a prefix of $\tau$, and $\tau$ is fair, since for all $n$ and all transition $T$ that can be fired after $\tau_n$, $\tau[1...n] = \tau_n \in [\![f]\!]$ and $T$ is in $L_n$ at a finite position $p$, so it will be fired after at most $|\sigma| - n + p$ steps. □

**Property 3** (Equivalence of fair executions)**.** Two fair executions define equivalent LAES:

$$\forall f \in Orc, \forall \sigma, \tau \in [\![f]\!]^{\omega}_{ic}, \overline{\overline{\sigma}} \equiv \overline{\overline{\tau}}.$$

*Proof.* Let $\sigma, \tau \in [\![f_0]\!]^{\omega}_{ic}$. We prove the following property $(P(n))$ by induction on $n$: for all $n$, there is $\gamma^n \in [\![f_0]\!]^{\omega}_{ic}$ such that $id_k : k \mapsto k$ defines an isomorphism in both directions between $\overline{\overline{\gamma^n}}$ and $\overline{\overline{\tau}}$ (i.e. $\overline{\overline{\gamma^n}} \equiv \overline{\overline{\tau}}$) and $id^n_i : \sigma[i]_k \mapsto \gamma^n[i]_k$ defines an injection from $\overline{\overline{\sigma[1...n]}}$ to $\overline{\overline{\gamma^n}}$. Moreover, if $n > 0$, $\gamma^{n-1}[1...n-1] = \gamma^n[1...n-1]$.

$P(0)$ is true with $\gamma^0 = \tau$. We suppose now $P(n)$ for a given $n$, and we prove $P(n+1)$.

The only restriction on the identifiers is that they must be unique in the execution. In particular, there is no link between the choice of the rules and of the identifier in a transition. Let $\delta^n$ be the same execution as $\gamma^n$, but where the identifiers appear in the same order as in $\sigma$. Let us compare $\delta^n[1...n]$ and $\sigma[1...n]$. Let $i \in \{1, ..., n\}$. By construction of $\gamma^n$, we have $\delta^n[i]_k = \sigma[i]_k$. Moreover, since $id_i^n$ is an injection from $\sigma[1...n]$ to $\gamma^n$, we have $\delta^n[i]_l = \sigma[i]_l$ and as the fields $c, a$ and $b$ are contained into $\{\delta^n[1]_k, ..., \delta^n[i-1]_k\}$, $(\delta^n[i]_c, \delta^n[i]_a, \delta^n[i]_b) = (\sigma[i]_c, \sigma[i]_a, \sigma[i]_b)$. Thus, $\delta^n[1...n] = \sigma[1...n]$.

Moreover, $\delta^n$ is fair, so $\overline{\overline{\sigma[1...n+1]}} < \overline{\overline{\delta^n}}$, where $f$ is the injection. Let $p$ such that $\delta[p]_k = f(\sigma[n+1]_k)$. As $\sigma[1...n] \in [\![f_0]\!]_{ic}$, for all $n < i < p$, $\delta^n[i]_k \not\leq \delta^n[p]_k$, so $\gamma^n[i]_k \not\leq \gamma^n[p]_k$. We can apply lemma 4 $p-n-1$ times on $\gamma^n$ at positions $p-1, p-2$, ..., $n+1$ to move $\gamma^n[p]_k$ at position $n+1$ in $\gamma^{n+1}$, $id_k$ is an isomorphism between $\overline{\overline{\gamma^{n+1}}}$ and $\tau$ and $id_i^{n+1}$ is an injection from $\overline{\overline{\sigma[1...n+1]}}$ to $\overline{\overline{\gamma^{n+1}}}$. Moreover, $\gamma^n[1...n] = \gamma^{n+1}[1...n]$, which is $P(n+1)$.

This proves $P(n)$ for all $n$.

For all $n$, we define $f(\sigma[n]_k) = \gamma^n[n]_k$. Let $n < p$. By unicity of the identifier in a sequence, we have $f(\sigma[n]_k) = \gamma^n[n]_k = \gamma^p[n]_k \neq \gamma^p[p]_k = f(\sigma[p]_k)$, so $f$ is injective. Moreover, for all $n$, we have $\Lambda_\sigma(\sigma[n]_k) = \Lambda_{\gamma^n}(\gamma^n[n]_k) = \Lambda_\tau(\tau[n]_k)$; for all $i$ and $j$, if $\sigma[i]_k \leq_\sigma \sigma[j]_k$, $\gamma^{\max(i,j)}[i]_k \leq_{\gamma^{\max(i,j)}} \gamma^{\max(i,j)}[j]_k$, so $\tau[i]_k \leq_\tau \tau[j]_k$; similarly, if $\sigma[i]_k \nearrow_\sigma \sigma[j]_k$, $\gamma^{\max(i,j)}[i]_k \nearrow_{\gamma^{\max(i,j)}} \gamma^{\max(i,j)}[j]_k$, so $\tau[i]_k \nearrow_\tau \tau[j]_k$.

Finally, $f$ defines an injection from $\overline{\overline{\sigma}}$ to $\overline{\overline{\tau}}$ and $\overline{\overline{\sigma}} < \overline{\overline{\tau}}$. By symmetry, $\overline{\overline{\tau}} < \overline{\overline{\sigma}}$, and $\overline{\overline{\tau}} \equiv \overline{\overline{\sigma}}$. □

**Theorem 2** (Correctness)**.** The fair executions are sound and complete:

$$\forall f \in Orc, \forall \sigma \in [\![f]\!]_{ic}^{\omega}, Lin(\overline{\overline{\sigma}}) = [\![f]\!].$$

*Proof.* Let $f \in Orc$ and $\sigma \in [\![f]\!]_{ic}^{\omega}$.

Let $\sigma_o \in [\![f_0]\!]_o$. By lemma 1, there is $\sigma_i \in [\![f_0]\!]_i$ such that $\sigma_o \in Lin(\overline{\overline{\sigma_i}})$. By lemma 2 and property 2, there is $\sigma_{ic} \in [\![f_0]\!]_{ic}^{\omega}$ such that $\overline{\overline{\sigma_i}} < \overline{\overline{\sigma_{ic}}}$. We have $\sigma_o \in Lin(\overline{\overline{\sigma_{ic}}})$. Moreover by property 3, $\overline{\overline{\sigma}} \equiv \overline{\overline{\sigma_{ic}}}$ so $\sigma_o \in Lin(\overline{\overline{\sigma_{ic}}})$. This proves that $[\![f]\!]_o \subset Lin(\overline{\overline{\sigma}})$.

We now prove the converse. Let $w = \Lambda(k^0)...\Lambda(k^n) \in Lin(\overline{\overline{\sigma}})$. We define $L$ as the set of finite executions according to the concurrent semantics that contain $w$ in there linearizations:

$$L = \{\gamma \in [\![f]\!]_{ic} | \overline{\overline{\gamma}} < \overline{\overline{\sigma}} \wedge w \in Lin(\overline{\overline{\gamma}})\}.$$

Let $\gamma \in L$ and $f$ be the injection from $\gamma$ to $\sigma$. We define:

$$\begin{array}{rl} D(\gamma) = & |\{i < i' | \exists j > j', f(\gamma[i]_k) = k^j \wedge f(\gamma[i']_k) = k^{j'}\}| \\ & + \sum_{i | \nexists j, f(\gamma[i]_k) = k^j} |\{i' | \exists j, f(\gamma[i']_k) = k^j\}| \\ & + |\gamma| - n. \end{array}$$

As $w$ is finite, there is a finite prefixe $\gamma$ of $\sigma$ that contains all the events of identifiers $k^0, ..., k^n$. We have $\sigma_0 \in L$, so $L \neq \emptyset$, and we can consider $\delta \in L$ for which $D(\delta)$ is minimal, and we suppose *(ad absurdum)* that $D(\delta) > 0$. $f$ is the injection from $\delta$ to $\sigma$ and $m = |\delta|$.

1. Suppose that for all $j$, $f(\delta[m]_k) \neq k^j$. $\delta[1, ..., m-1] \in L$ and $D(\delta[1, ..., m-1]) = D(\delta) - 1$ which is in contradiction with the fact that $D(\delta)$ is minimal.
2. Otherwise, if $m \neq n$, there is $i$ such that $\delta[i]_k$ has no antecedant by $f$. Let $i_0$ be the biggest such $i$. $i_0 \neq m$ so $\delta[i_0+1]_k$ is defined and has an antecedant $k^j$ by $f$.

   As $w$ is a linearization, the causal history of $k^j$ is contained into $\{k^1, ..., k^n\}$, so $\delta[i_0]_k \not\leq \delta[i_0+1]_k$. We can apply lemma 4, which gives $\delta' \in [\![f_0]\!]_{ic}$ with $\overline{\overline{\delta'}} \equiv \overline{\overline{\delta}}$. We have $\delta' \in L$ and $D(\delta') = D(\delta) - 1$, which is impossible by definition of $\delta$.
3. Otherwise, $m = n$ and $f$ is bijective, so $U = \{i < i' | \exists j > j', f(\gamma[i]_k) = k^j \wedge f(\gamma[i']_k) = k^{j'}\} \neq \emptyset$ Let $(i, i') \in U$ such that $i' - i$ is minimal. Suppose $i' - i > 1$. There is $i''$ such that $i < i'' < i$. As $f$ is bijective, there is $j''$ such that $f(\gamma[i'']_k) = k^{j''}$ and either $(i, i'') \in U$ or $(i'', i') \in U$, which is impossible as $i' - i$ is minimal. So $i' = i + 1$, and there are $j > j'$ such that $f(\gamma[i]_k) = k^j$ and $f(\sigma[i+1]_k) = k^{j'}$.

   As $w$ is a linearization, we have $\neg(k^j \nearrow k^{j'})$, so $k^j \not\leq k^{j'}$, and by injection, $\sigma[i]_k \not\leq \sigma[i+1]_k$. We can apply lemma 4, which gives $\sigma' \in [\![f_0]\!]_{ic}$ with $\overline{\overline{\sigma'}} \equiv \overline{\overline{\sigma}}$. We have $\sigma' \in L$ and $D(\sigma') = D(\sigma) - 1$, which is impossible by definition of $\sigma$.

All three cases are impossible, so $D(\delta) = 0$. We have $m = n$ and for all $i \leq n$, $f(\delta^i) = k^i$. As $w \in Lin(\overline{\overline{\delta}})$, we can conclude by lemma 3 that $w \in [\![f_0]\!]$.

This proves, $Lin(\overline{\overline{\sigma}}) \subset [\![f]\!]_o$. □

**Theorem 1** (Correctness)**.** The behaviors that can be observed from an execution in the instrumented semantics are correct with respect to the standard semantics:

$$\forall f \in \mathcal{O}, \forall \sigma_i \in [\![f]\!]_i, Lin(\overline{\overline{\sigma_i}}) \subset [\![f]\!]$$

*Proof.* Let $\sigma_i \in [\![f_0]\!]_i$. By lemma 2, and property 2, there is a fair execution $\sigma_{ic} \in [\![f_0]\!]_{ic}$ such that $\overline{\overline{\sigma_i}} < \overline{\overline{\sigma_{ic}}}$. Let $w \in Lin(\overline{\overline{\sigma_i}})$. We have $w \in Lin(\overline{\overline{\sigma_{ic}}})$, so by theorem 2, $w \in [\![f_0]\!]$. □

# 7 Discussion and Conclusion

This paper addresses the problem of the formal definition of an operational semantics for a distributed programming language. We based our work on a case study: the Orc language, as it is expressive enough to easily generate many situations found in distributed systems, such as causality, concurrency, preemption and conflict, and remains simple enough to be tractable in a formal work as this one. Our contribution consists of two new semantics for the Orc language. An instrumentation of the standard semantics that extracts as much information as possible from a unique execution and a full concurrent semantics that sums up all the possible behaviors of a program.

Debugging is a direct application of our work, as it requires root causes analysis, replay and even race detection. From the user point of view, a debugger should give the possibility to execute a program, (which is the role of the instrumented semantics), but should also give the possibility to explore more than one possibility in conflictual choices (here, the concurrent semantics becomes necessary). Lemma 2 gives an early response. As the semantics are compatible, it is possible to let the instrumented semantics drive the execution of the concurrent semantics, and to introduce conflicts only when the programmers want to point them out explicitly.

Beyond the Orc language, we think that this work presents a general approach that can be used for other non-deterministic languages with concurrency operators.

Causality often operates through values and variables. It might be possible to use this remark to track causality by a program transformation. This could be useful to improve the efficiency of an implementation.

# References


[1] Roberto Bruni, Hernán Melgratti, and Emilio Tuosto. *Translating Orc features into Petri nets and the Join calculus*. Springer, 2006.

[2] David Kitchin, Adrian Quark, William Cook, and Jayadev Misra. The orc programming language. In *Formal techniques for Distributed Systems*, pages 1–25. Springer, 2009.

[3] Gordon D Plotkin. The origins of structural operational semantics. *The Journal of Logic and Algebraic Programming*, 60:3–15, 2004.

[4] Sidney Rosario, David Kitchin, Albert Benveniste, William Cook, Stefan Haar, and Claude Jard. *Event structure semantics of orc*. Springer, 2008.

[5] Grigore Roşu and Koushik Sen. An instrumentation technique for online analysis of multithreaded programs. *Concurrency and Computation: Practice and Experience*, 19(3):311–325, 2007.

[6] Glynn Winskel. *Event structures*. Springer, 1987.