Configuration Structures and Logical Equivalence

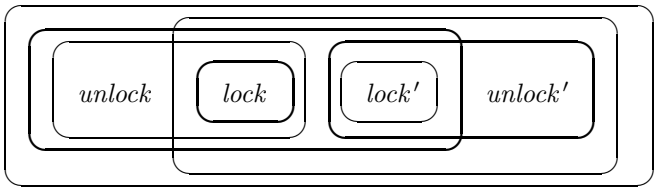
Pietro Cenciarelli

University of Rome, "La Sapienza"
Department of Computer Science - Via Salaria 113, 00198 Roma.
cenciarelli@dsi.uniroma1.it

Abstract. *Configuration theories* [Cen02] describe concurrent systems axiomatically. Rules for composing configurations (of events) are represented by sequents $\Gamma \vdash_\rho \Delta$, where Γ and Δ are sequences of posets (of events) and ρ is a matrix of monotone maps from the components of Γ to the components of Δ. The structural rules of Gentzen's sequent calculus are decorated by suitable operations on matrices, where *cut* corresponds to product. The calculus is interpreted in a rather general class of configuration structures called *monotone*. Two such structures are called *logically equivalent* if they satisfy the same sequents. This notion of equivalence is shown to be intermediate between (pomset) trace equivalence and (history preserving) bisimulation. A new form of sequent, more expressive than in [Cen02], is also proposed and some of the classical closure properties adopted in literature for configuration structures are thereof axiomatised. In this more expressive setting the notion of logical equivalence becomes "resource sensitive," and it is shown to correspond to isomorphism when models are restricted to tree-like structures.

1 Introduction

Two processes P and Q operating in parallel compete for a lock on shared data. The following structure models the parallel composition $P \,|\, Q$, where P executes $lock; \ldots unlock$; and the same does Q. The identifiers *lock* and $lock'$ represent *events* occurring in computation, namely the execution of a "lock" action respectively by P and Q. Similarly for *unlock* and $unlock'$.



(1)

Sets of events, called *configurations* and depicted here as rounded squares surrounding their elements, represent consistent states of computation. The $\{unlock, lock\}$ configuration, for example, represents the state reached by the

system after having performed a lock action *first* and then an unlock (while Q remains dormant). We know the lock came first because we see a $\{lock\}$ subconfiguration but not an $\{unlock\}$. Note that there is no configuration $\{lock, lock'\}$ and this represents the *mutual exclusion* of the two processes from the shared resource.

Diagram (1) depicts a *configuration structure* [Win82], a model introduced by Winskel as an alternative presentation of (prime) *event structures* [NPW81]. Different closure conditions have been proposed over the years to make configuration structures mathematically tractable. In [GG01] van Glabbeek and Goltz characterise the class of configuration structures where the *causal dependency* between events can be faithfully represented by means of partial orders. Such structures, called *stable*, are required to be closed under bounded unions and bounded intersections. Stable structures possess useful semantic properties. For example, when a state A is part of the "history" of a state B, then B is reachable from A by a sequence of atomic steps of computation. Unfortunately stability is not satisfied by structures such as (1), which arise naturally in the semantics of concurrent systems. We open the present paper by proposing a more general class of models: the *monotone* configuration structures, of which (1) is an example. Section 2 studies the closure properties of such models.

Monotone configuration structures are designated models of *configuration theories*, an axiomatic approach to the semantics of concurrent systems proposed in [Cen02] and surveyed here in Section 3. A configuration theory is a set of *poset sequents* closed under deduction. In Figure 1 we see two such sequents. They spell roughly: "every *unlock* action must be preceded by a *lock* action" and "in between any two *lock* actions an *unlock* must occur." Poset sequents are made of partially ordered sets (posets) of events, where the order (represented in the picture by the vertical bars) is interpreted as causal dependency.

Fig. 1. A naive axiomatisation of locks.

In Section 4 we investigate how fit poset sequents are for describing concurrent systems and, in particular, how accurate they are in discriminating process behaviour. We discover that the natural notion of *logical equivalence* arising from interpreting sequents (two structures are equivalent when they satisfy the same sequents) is intermediate between *pomset trace equivalence* and *history preserving bisimulation*. These two equivalences are chosen here as representative respectively of "linear time" and "branching time" semantics within causality-based models because they do not rely on the assumption of *action atomicity*

[LC87]. Other notions, such as interleaving trace equivalence or pomset bisimulation equivalence for example, are less stable with respect to changes in the level of abstraction at which systems are described as they are not preserved under action refinement (see discussion in [GG01]).

In [Cen02] we axiomatised the memory-cache interaction protocol adopted in Java by means of poset sequents, and derived formally a nontrivial property of the Java memory model involving basic actions of the virtual machine. However, when it comes to describing the management of locks, which involves a complex interplay of subconfigurations, causal dependency alone proves inadequate. Example 2 shows that the sequent to the right of Figure 1 actually fails to prevent locks from being granted when the shared resource is still busy. A correct axiomatisation is obtained in Section 5, where we introduce a more expressive form of sequent, called *structure sequents*, whose components are not just posets but configuration structures. In this new, more expressive setting the classical closure properties adopted in literature for configuration structures (such as coincidence freeness or closure under bounded unions) are easily axiomatised. In Section 6 we show that, when referred to structure sequents, logical equivalence becomes "resource sensitive," and that it corresponds to *isomorphism* when models are restricted to tree-like structures.

Notation. We write function composition in diagrammatical order.

2 Monotone Configuration Structures

A *set system* consists of a set E and a collection $\mathcal{A}$ of subsets of E [GP95]. Without loss of generality we may assume that each element of E belongs to at least one element of $\mathcal{A}$. Hence we write just $\mathcal{A}$ for a set system and let $|\mathcal{A}|$ be the set $\bigcup \mathcal{A}$. If $A \in \mathcal{A}$ we write $sub\,(A)$ the set $\{B \in \mathcal{A} \,|\, B \subseteq A\}$. If $A, B \in sub\,(C)$ for some $C \in \mathcal{A}$ we say that A and B are *bound* in $\mathcal{A}$. The sets in a system $\mathcal{A}$ are called *configurations* when used for modeling a concurrent system, while the elements of the set $|\mathcal{A}|$ are called *events*. If $B \in \mathcal{A}$ and $A \in sub\,(B)$, then A is called a *subconfiguration* of B.

Notation. Events are written a, b, c . . . . When events are labelled (e.g when part of a labelled structure or sequent) we adopt the convention that the same metavariable is used for all events with the same label, distinguishing different occurrences with primes and indices. For example a set of three events, two with label α and a third labelled β, may be written $\{a, a', b\}$. When no confusion arises, we push notation even furhter and write such a set as $\{a, a, b\}$, making repetition count. When labels carry special meaning evocative identifiers are used: $\{lock, unlock, lock, unlock\}$ denotes a set of four events, two of which performing an action of locking and the others unlocking. Sometimes we represent configuration structures in pictures, where configurations are drawn as circles or polygons surrounding the elements that they contain. For example,

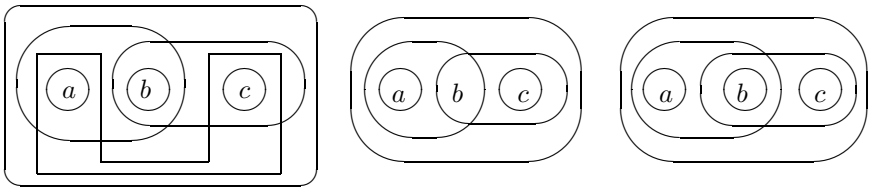
represents the structure $\{\emptyset, \{a\}, \{a, b\}, \{c\}\}$. Unless otherwise stated, we always assume the empty configuration, without representing it explicitely. □

In [Win87] several closure conditions on the set of configurations of a structure $\mathcal{A}$ are given in order to get a precise match with *general event structures* (generalising those of [NPW81]). They are: *finiteness* (if an event belongs to a configuration A, then it also belongs to a finite subconfiguration of A), *coincidence-freeness* (if two distinct events belong to a configuration A, then there exists a subconfiguration of A containing exactly one of them), closure under *bounded unions* and *non-emptyness* of $\mathcal{A}$.

We call *configuration structures* (or just *structures*), and write them $\mathcal{C}, \mathcal{D} \ldots$, the set systems satisfying all of the above requirements, *except* closure under bounded unions (this is not standard in literature). Coincidence-freeness endows each configuration C with a *canonical* partial order: $a \leq_C b$ if and only if, for all $D \in \mathit{sub}(C)$, $b \in D$ implies $a \in D$. This relation is called *causal dependency*. If $a \in C$, we write $a{\downarrow_C}$ the set $\{b \in C \mid b \leq_C a\}$.

A structure $\mathcal{C}$ is called *connected* if, for all configurations $C \neq \emptyset$, there exists $a \in C$ such that $C - \{a\} \in \mathcal{C}$. Clearly connectedness implies coincidence freeness and moreover, having assumed $\mathcal{C}$ nonempty and finitary, it also implies that $\emptyset \in \mathcal{C}$ (*rootedness*). Conversely, coincidence freeness does not imply connectedness, even when in conjunction with closure under bounded unions. Following [GG01] we call *stable* a configuration structure which is connected, closed under nonempty bounded unions and nonempty bounded intersections. Stability was introduced for *event structures* in [Win87]. Stable structures are precisely those where the order on a configuration determines its subconfigurations (see [GG01, Proposition 5.4 and Theorem 5.2]).

Example 1 Each of the structures depicted below, which we name (from left to right) 1.1, 1.2 and 1.3, features a maximal configuration $\{a, b, c\}$ with the discrete order. Only 1.1 is stable.



□

While having nice algebraic consequences, stability is a rather strong requirement which is not satisfied, as noted regretfully in [Win87], when events can be caused in several compatible ways. Here we seek weaker conditions to yield more general models, though retaining nice algebraic properties (see Theorem 2).

In a stable structure $\mathcal{C}$ causal dependency is preserved by inclusions: if $C \in \mathcal{C}$ and $D \in \mathit{sub}(C)$, the inclusion $D \subseteq C$ is a monotone map from $(D, \leq_D)$ to $(C, \leq_C)$. We call this property *monotonicity*.

Definition 1 *A configuration structure is called* monotone *if, for all configurations C and $D \in sub(C)$, $a \leq_D b$ implies $a \leq_C b$.*

The structure of diagram (1) is monotone but not stable, and so is 1.3 of Example 1, while 1.2 is not even monotone. In [Cen02] we noted that a structure is monotone (then called *conservative*) if and only if it has downwards-closed bounded intersections. Below we extend this result by giving a further characterization of monotonicity which does not refer to the order (condition 4).

Theorem 2 *The following conditions on a structure $\mathcal{C}$ are equivalent:*

1. *$\mathcal{C}$ is monotone;*
2. *$\mathcal{C}$ is closed under principal ideals;*
3. *$a\downarrow_A = a\downarrow_B$ for all A and B bound in $\mathcal{C}$ and $a \in A \cap B$;*
4. *$A \cap B = \bigcup\{D \in \mathcal{C} \mid D \subseteq A \cap B\}$ for A and B bound in $\mathcal{C}$.*

Proof. We develop the proof only for 1, 3 and 4. (1$\Rightarrow$3) Let $C \in \mathcal{C}$ and let $A, B \in sub(C)$. If $\mathcal{C}$ is monotone then $b \leq_A a$ implies $b \leq_C a$. Then, if $a \in A \cap B$, $b \leq_B a$ follows from the definition of $\leq_C$. So $a\downarrow_A \subseteq a\downarrow_B$. The argument is symmetrical. (3$\Rightarrow$4) $\bigcup\{D \in \mathcal{C} \mid D \subseteq A \cap B\} \subseteq A \cap B$ trivially. If $a \in A \cap B$ then $a \in a\downarrow_A$ and $a \in a\downarrow_B$. By hypothesis $a\downarrow_A = a\downarrow_B \subseteq A \cap B$. Then, since 3$\Rightarrow$ 2, $a \in \bigcup\{D \in \mathcal{C} \mid D \subseteq A \cap B\}$. (4$\Rightarrow$1) Suppose $\mathcal{C}$ is not monotone. There must exist $C \in \mathcal{C}$ and $A, B \in sub(C)$ such that $a \leq_A b$, $b \in B$ and $a \notin B$. If $a \leq_A b$ then all subconfigurations of A containing b must also contain a and hence, if $a \notin B$, there cannot be a configuration $D \subseteq A \cap B$ such that $b \in D$. Therefore, $A \cap B \not\subseteq \bigcup\{D \in \mathcal{C} \mid D \in A \cap B\}$. □

Note that, since condition 1 implies 4, monotone structures which are closed under bounded unions are also closed under bounded binary intersections. This implication also follows from [Win82, Proposition 1.8]. Theorem 2 shows that, like in event structures, the principal ideals of a monotone structure $\mathcal{C}$ are configurations. However, note that, unlike in event structures, principal ideals are not the complete primes of the poset $(\mathcal{C}, \subseteq)$, and neither are they *compact* elements.

Let $\mathcal{C}$ be a structure and let $A, B \in \mathcal{C}$. When $B = A \cup \{a\}$ for some event $a \notin A$ we write $A \to B$, or $A \xrightarrow{a} B$ to make a explicit. The reflexive and transitive closure of $\to$ is written $\hookrightarrow$ and, when this relation holds, we also denote by $A \hookrightarrow B$ the inclusion map. This map is monotone (with respect to causal dependency) when $\mathcal{C}$ is monotone. When $\mathcal{C}$ is connected and closed under bounded unions, $A \in sub(B)$ implies $A \hookrightarrow B$. This is not true in general for connected monotone structures.

3 Configuration Theories

Capital letters $A, B \ldots$ are used here to denote posets, while $\Gamma, \Delta \ldots$ denote sequences of posets. Unless such a sequence is introduced explicitly by an equation $\Gamma = A_1, \ldots A_m$, we write Γ_i for the i-th component of Γ. The concatenation of

two sequences Γ and Δ is written Γ, Δ. If $\Gamma = A_1, \ldots A_m$ and $\Delta = B_1 \ldots B_n$ are finite sequences of posets, we write $\rho : \Gamma \to \Delta$ to mean that ρ is an $m \times n$ matrix of monotone functions $\rho_{ij} : A_i \to B_j$. If C is a configuration of a configuration structure and Γ is as above, we call *interpretation* of Γ in C an $m \times 1$ matrix $\Gamma \to C$ of *monotone injective* functions, where C is viewed as endowed with the causal dependency order.

Definition 3 ([Cen02]) *A* poset sequent $\Gamma \vdash_\rho \Delta$ *(just* sequent *for short) consists of two finite sequences Γ and Δ of posets and a matrix $\rho : \Gamma \to \Delta$ of monotone injective functions.*

The posets in a sequent are meant to represent fragments of a configuration of events. The intuitive meaning of a sequent $\Gamma \vdash_\rho \Delta$ is that whenever a single configuration interprets *all* components of Γ, the interpretation extends along ρ to *at least one* component of Δ. Of course the Δ_i may include more events than are mentioned in Γ, thus specifying what is required to happen after (or must have happened before) a certain combination (Γ) of events. We write just ρ for a sequent $\Gamma \vdash_\rho \Delta$ when Γ and Δ are understood or not relevant. On the other hand, we may omit ρ when obvious from the labelling conventions. When drawing sequents in pictures, we generally do not surround the elements of a posets. Hence, we may write $a \vdash a\ b$ instead of $\boxed{a} \vdash \boxed{a\ b}$.

Definition 4 ([Cen02]) *A monotone structure $\mathcal{C}$ is said to* satisfy *a sequent $\Gamma \vdash_\rho \Delta$ when, for any configuration $C \in \mathcal{C}$ and interpretation $\pi : \Gamma \to C$, there exist a configuration $D \in \mathcal{C}$, a component $\Delta_k \in \Delta$ and a monotone injective function $q : \Delta_k \to D$ such that $C \in sub(D)$ and, for all i, the following diagram commutes.*

$$\begin{array}{ccc} \Gamma_i & \xrightarrow{\rho_{ik}} & \Delta_k \\ {\scriptstyle \pi_i}\downarrow & & \downarrow{\scriptstyle q} \\ C & \hookrightarrow & D \end{array}$$

A pathological kind of sequent is $\vdash$, which features empty sequences as antecedent and succedent, and is decorated by the empty matrix. Under the assumption that structures are not empty, this sequents denotes the *absurd*. A sequent of the form $\vdash A$ is satisfied by structures in which every computation is bound to produce a configuration matching A. Similarly the sequent $A \vdash$ is satisfied by structures in which no configuration ever matches A.

A *labelled configuration structure* [GP95] is a structures $\mathcal{C}$ endowed by a labelling function $\lambda : |\mathcal{C}| \to Act$, where Act is a fixed set of labels called *actions*. Similarly, a *labelled sequent* ρ is one in which the elements of posets are assigned labels from Act and the maps in ρ preserve them. Definition 4 extends to labelled sequents and structures by requiring that interpretation maps preserve labels.

Example 2 In Figure 1 we rely on the labelling conventions introduced in Section 2. In particular, the matrix remains implicit in both sequents because there exists a unique label preserving monotone function from the antecedent to the succedent. The sequent to the right is satisfied by the structure of diagram (1) *trivially*, simply because none of the two *lock* actions in (1) depends on the other, and hence there exists no interpretation in (1) of the sequent's antecedent. Unfortunately, just for the same reason, the sequent is also satisfied by the structure obtained by adding to (1) the "forbidden" configuration $\{lock, lock'\}$, thus showing that the axiom does not serve the purpose of enforcing mutual exclusion. □

[true]	$\dfrac{}{\vdash \emptyset}$	[iso]	$\dfrac{\phi \text{ is iso}}{A \vdash_{\phi} B}$
[l-weak]	$\dfrac{\Gamma \vdash_{\rho} \Delta}{\Gamma, \emptyset \vdash_{\rho;\emptyset} \Delta}$	[r-weak]	$\dfrac{\Gamma \vdash_{\rho} \Delta}{\Gamma \vdash_{\rho,\sigma} \Delta, A}$
[l-contr]	$\dfrac{\Gamma, A, A \vdash_{\rho;\sigma;\sigma} \Delta}{\Gamma, A \vdash_{\rho;\sigma} \Delta}$	[r-contr]	$\dfrac{\Gamma \vdash_{\rho,\sigma,\sigma} \Delta, A, A}{\Gamma \vdash_{\rho,\sigma} \Delta, A}$
[l-exc]	$\dfrac{\Gamma, A, B, \Pi \vdash_{\rho;\sigma;\tau;\theta} \Delta}{\Gamma, B, A, \Pi \vdash_{\rho;\tau;\sigma;\theta} \Delta}$	[r-exc]	$\dfrac{\Gamma \vdash_{\rho,\sigma,\tau,\theta} \Delta, A, B, \Pi}{\Gamma \vdash_{\rho,\tau,\sigma,\theta} \Delta, B, A, \Pi}$
[l-cut]	$\dfrac{\Pi \vdash_{\tau} A \qquad \Gamma, A \vdash_{\rho;\sigma} \Delta}{\Gamma, \Pi \vdash_{\rho;\tau\sigma} \Delta}$	[r-cut]	$\dfrac{A \vdash_{\tau} \Pi \qquad \Gamma \vdash_{\rho,\sigma} \Delta, A}{\Gamma \vdash_{\rho,\sigma\tau} \Delta, \Pi}$

Table 1. Structural rules

Table 1 presents a system of structural inference rules proposed in [Cen02]. A *configuration theory* is a set of sequents that is closed under deduction. The rules are shown to be sound with respect to interpretation in monotone structures. Completeness is proven for a restriction of the calculus to *finite* sequents, augmented by a rule for extending the premises of a sequent to larger contexts of events. The rules are roughly the structural rules of Gentzen's sequent calculus, decorated by suitable matrix expressions. We explain the cut rules and refer the reader to [Cen02] for a more formal presentation. In [l-cut] $\rho : \Gamma \to \Delta$ is an $m \times n$ matrix, where $m = |\Gamma|$ and $n = |\Delta|$, while $\sigma : A \to \Delta$ is $1 \times n$. Then $(\rho\,;\,\sigma)$ is the $(m+1) \times n$ matrix $\Gamma, A \to \Delta$ obtained by placing ρ above σ, that is: $(\rho\,;\sigma)_{ij} = \rho_{ij}$ if $i \leq m$, while $(\rho\,;\sigma)_{(m+1)j} = \sigma_{1j}$. Moreover, if $|\Pi| = r$, then $\tau\,\sigma : \Pi \to \Delta$ is the $r \times n$ matrix obtained by multiplying the $r \times 1$ matrix $\tau : \Pi \to A$ with the $1 \times n$ matrix $\sigma : A \to \Delta$, where multiplication of matrix components is just function composition. Similarly, $(\rho, \sigma) : \Gamma \to \Delta, A$ in [r-cut] is the matrix obtained by placing $\rho : \Gamma \to \Delta$ *beside* $\sigma : \Gamma \to A$, while $\sigma\,\tau$ is a product as above.

4 Notions of Equivalence

Here we assess the expressive power of poset sequents, and how accurate they are in specifying system behaviour. We implicitely assume that structures and posets are labelled over one set Act of action labels.

Definition 5 *Two monotone structures $\mathcal{C}$ and $\mathcal{D}$ are called* logically equivalent, *written $\mathcal{C} \cong_{\vdash} \mathcal{D}$, when, for all sequents ρ, $\mathcal{C}$ satisfies ρ if and only if $\mathcal{D}$ satisfies ρ.*

Example 3 The structure $\mathcal{C} = \{\{a\}, \{a,b\}, \{a,b,c\}, \{c\}\}$ is logically equivalent to $\mathcal{D} = \{\{a\}, \{a,b\}, \{a,b,c\}, \{c\}, \{c'\}\}$. Note that neither structure satisfies the sequent $c \vdash \{c, a\}$, stating that any configuration containing a c extends to one containing an a. In fact, once the antecedent is interpreted in the configuration $\{c\}$, no commuting diagram as required by Definition 4 exists because, although $\{c\} \subseteq \{a.b.c\}$, $\{c\} \hookrightarrow \{a,b,c\}$ does *not* hold. In the setting of [Cen02] where $\hookrightarrow$ means just *containment*, $\mathcal{C}$ would satisfy $c \vdash \{c, a\}$ while $\mathcal{D}$ would not.

Example 4 The structure $\{\{a\}, \{b\}, \{a,b,c\}\}$ is *not* logically equivalent to $\{\{a\}, \{b\}, \{a,b,d\}\}$, as the former satisfies $\begin{matrix} d \\ | \\ a \end{matrix} \vdash$ while the latter does not.

Definition 6 *A* history preserving bisimulation *between two structures $\mathcal{C}$ and $\mathcal{D}$ is a relation $R \subseteq \mathcal{C} \times \mathcal{D} \times \mathcal{P}(|\mathcal{C}| \times |\mathcal{D}|)$ such that $(\emptyset, \emptyset, \emptyset) \in R$ and, if $(C, D, f) \in R$, then*

- *f is an isomorphism between $(C, \leq_C)$ and $(D, \leq_D)$;*
- *if $C \xrightarrow{a} C'$ then there exist $D' \in \mathcal{D}$ such that $D \xrightarrow{a} D'$ and $(C', D', f') \in R$, where f' extends f;*
- *if $D \xrightarrow{a} D'$ then there exist $C' \in \mathcal{C}$ such that $C \xrightarrow{a} C'$ and $(C', D', f') \in R$, where f' extends f.*

Two structures $\mathcal{C}$ and $\mathcal{D}$ are *history preserving bisimulation equivalent* (*bisimulation equivalent* for short), written $\mathcal{C} \approx_h \mathcal{D}$, when there exists a history preserving bisimulation between them. The above definition is from [GG01] where *stable* structures endowed with a termination predicate are considered. It rephrases in terms of configuration structures the *partial order equivalence* of [DNM87]. Note that, in general, monotonicity is not preserved by history preserving bisimulation: when $\mathcal{C} \approx_h \mathcal{D}$ and $\mathcal{C}$ is monotone, $\mathcal{D}$ need not be monotone.

Partially ordered multisets, or *pomsets* [Pra86], are isomorphism classes of posets (labelled over a set Act). We write $[A]$ the pomset (the isomorphism class) of a poset A. Viewing the configurations of a structure $\mathcal{C}$ as posets, we write $Poms(\mathcal{C})$ the set $\{[C] \,|\, C \in \mathcal{C}\}$ of its pomsets. Two structures $\mathcal{C}$ and $\mathcal{D}$ are called *pomset trace equivalent* [GG01, 8.1], written $\mathcal{C} \cong_t \mathcal{D}$, when $Poms(\mathcal{C}) = Poms(\mathcal{D})$. Again, the definition in [GG01] refers to stable structures.

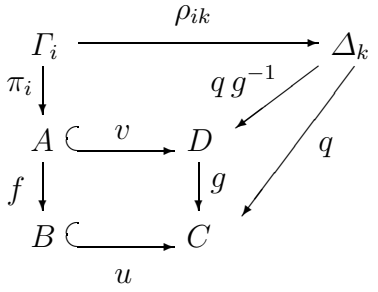
Example 5 Connected structures may be depicted as graphs where nodes represent configurations and edges represent transitions $\xrightarrow{a}$. For example, the structures $\mathcal{C} = \{\{a\}, \{a,b\}, \{a,c\}\}$ and $\mathcal{D} = \{\{a\}, \{a,b\}, \{a'\}, \{a',c\}\}$ are drawn respectively $\begin{smallmatrix} b & & c \\ & \diagdown\!\diagup & \\ & a & \end{smallmatrix}$ and $\begin{smallmatrix} b & & c \\ \mid & & \mid \\ a & \diagdown\!\diagup & a \end{smallmatrix}$. This is a textbook example of trace equivalent processes that are not bisimulation equivalent. □

Surprisingly enough, bisimulation does not imply pomset trace equivalence: the structures of Example 4 are bisimulation equivalent although the first features a configuration $\{a,b,c\}$ that the other does not have. In fact, in the general case of monotone structures, no inclusion holds among the three notions of equivalence we are considering. In order to relate them we start by observing that both $\{a,b,c\}$ and $\{a,b,d\}$ in Example 4 are *not* reachable. It is easy to see that, when models are restricted to *connected* structures, $\approx_h$ implies $\cong_t$. The following theorem states that so it happens with $\cong_\vdash$.

Theorem 7 *Let $\mathcal{C}$ and $\mathcal{D}$ be monotone and connected structures. If $\mathcal{C} \approx_h \mathcal{D}$ then $\mathcal{C} \cong_\vdash \mathcal{D}$.*

Proof. Let $\mathcal{C} \approx_h \mathcal{D}$. We show that, if $\mathcal{C}$ satisfies $\Gamma \vdash_\rho \Delta$ then so does $\mathcal{D}$. Let $A \in \mathcal{D}$ and let $\pi : \Gamma \to A$ be an interpretation of Γ in A. By simulating in $\mathcal{C}$ the computation $\emptyset \hookrightarrow A$ we get a configuration $B \in \mathcal{C}$ related with A by an isomorphism $f : A \to B$. Since $\mathcal{C}$ satisfies ρ, the interpretation $\pi\, f : \Gamma \to B$ yields $\Delta_k \in \Delta$, $C \in \mathcal{C}$ and $q : \Delta_k \to C$ such that $u : B \hookrightarrow C$ and $\pi\, f\, u = \rho_{ik}\, q$ for all i. Since A and B are related by a history preserving bisimulation, the computation u is simulated in $\mathcal{D}$ by a $v : A \hookrightarrow D$ such that $f\, u = v\, g$, as in the diagram. Then, $\pi_i v = \pi_i v\, g\, g^{-1} = \pi_i f\, u\, g^{-1} = \rho_{ik}\, q\, g^{-1}$ for all i, as required.

□



The result does not extend to the *weak* history preserving bisimulation of [GG01, Def 9.3], where the isomorphism between related configurations is not part of the relation itself. Note also that Theorem 7 fails when $\hookrightarrow$ is interpreted as containment (just consider the bisimulation equivalent structures of Example 3).

The setting of connected structures is still too general to relate trace and logical equivalence. The structures 1.1 and 1.3 of Example 1 are logically equivalent although the former features a (reachable) configuration that the latter does not have. On the other hand $\cong_t \not\Rightarrow \cong_\vdash$, as shown by the following example.

Example 6 $\{\{a\},\{a,b\}\}$ is trace equivalent to $\{\{a\},\{a,b\},\{a'\}\}$. However the former satisfies $\vdash b$, while the latter doesn't. □

Logical equivalence becomes strictly stronger than pomset trace eqivalence if we restrict to stable and finitely branching structures. A structure is called *finitely branching* if the number of configurations of cardinality n is finite, for every finite n.

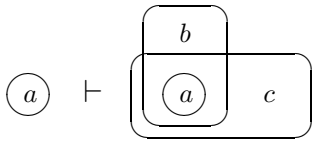
Theorem 8 *Let $\mathcal{C}$ and $\mathcal{D}$ be stable, finitely branching structures. If $\mathcal{C} \cong_{\vdash} \mathcal{D}$ then $\mathcal{C} \cong_t \mathcal{D}$.*

Proof. Let $\mathcal{C}$ and $\mathcal{D}$ be stable, finitely branching structures, and suppose there exists no configuration in $\mathcal{D}$ isomorphic to some $C \in \mathcal{C}$. Let $r : C \to D \in \mathcal{D}$ be a monotone injective function. Either i) $r(C) \notin \mathcal{D}$, or else ii) there exist $a, b \in C$ such that $a \not\leq_C b$ and $r(a) \leq_D r(b)$. Since *stable* structures are closed under taking downwards closed subsets of configurations, if i) holds then $r(C)$ is not downwards closed and hence, by the definition of $\hookrightarrow$, there can exist no injection $C \hookrightarrow B \in \mathcal{C}$ factorising through r. Neither can there be such a factorisation if ii) holds because, by the definition of causal dependency, $a \not\leq_C b$ implies $a \not\leq_B b$. Call *minimal* a map $C \to D$, where $D \in \mathcal{D}$, if it factorises through no subconfiguration of D, and consider the (possibly empty) sequence $\rho = \rho_1, \rho_2 \ldots$ containing *all* the monotone injective minimal maps $\rho_i : C \to D_i$ where $D_i \in \mathcal{D}$. Since $\mathcal{D}$ is finitely branching, this sequence is finite. Clearly $\mathcal{D}$ satisfies $C \vdash_\rho D_1, D_2 \ldots$. However $\mathcal{C}$ does not satisfy it because, choosing the identity interpretation of the antecedent in itself, satisfaction would require an injection $C \hookrightarrow B \in \mathcal{C}$ factorising through some ρ_k, which was shown impossible.

5 Structure Sequents

The structures of Example 5 are logically equivalent, although $\mathcal{D}$ cannot simulate $\mathcal{C}$. What distinguishes the two structures is that in $\mathcal{C}$ two *conflicting* events, b and c, may *both* follow a certain configuration, $\{a\}$, but not so in $\mathcal{D}$. The sequent $\{a\} \vdash \{a,b\},\{a,c\}$, for example, would not make this distinction. The reason is clear from Definition 4: poset sequents predicate over the structure (causal dependency) of *single* configurations, and hence they cannot express conflict.

In this section we introduce a more expressive form of sequent $\Gamma \vdash_\rho \Delta$, called *structure sequent*, where the components of Γ and Δ are not meant as pieces of configurations, but rather as pieces of configuration structures, while the ρ_{ij} are structure morphisms. In this new setting, as we shall see below, structure $\mathcal{C}$ of Example 5 satisfies the sequent below while $\mathcal{D}$ does not.



Definition 9 *An* embedding *of a set system $\mathcal{A}$ in a set system $\mathcal{B}$ consists of a function $\phi : \mathcal{A} \to \mathcal{B}$ and a function $f : |\mathcal{A}| \to |\mathcal{B}|$ such that:*

- *ϕ preserves containment,*
- *for all $A \in \mathcal{A}$, $f|_A : A \to |\mathcal{B}|$ is one-to-one, and*
- *for all A, B bound in $\mathcal{A}$ and for all $a \in A$, $f(a) \in \phi(B)$ if and only if $a \in B$.*

The third clause above requires that embeddings *reflect* intersections of bound configurations. From this condition it follows immediately that $f(A) \subseteq \phi(A)$ for all $A \in \mathcal{A}$. If $\mathcal{C}$ and $\mathcal{D}$ are the structures of Example 5, there exists an obvious embedding of $\mathcal{D}$ in $\mathcal{C}$ but none in the opposite direction.

Definition 10 *A morphism $\mathcal{A} \to \mathcal{B}$ of set systems is a function $f : |\mathcal{A}| \to |\mathcal{B}|$ such that there exists an embedding (ϕ, f) of $\mathcal{A}$ in $\mathcal{B}$.*

Definition 10 weakens the standard notion of synchronous morphism found in literature ([Win87,GG01]), which requires $f(A)$ to be a configuration. We call *strict* the morphisms satisfying this requirement. For example, a morphism $\{\{unlock\}\} \to \{\{lock\}, \{lock, unlock\}\}$ is needed to act as ρ in a sequent expressing that any *unlock* action must be preceeded by a *lock* action, as in Section 1. This is indeed a morphism according to Definition 10, but not a strict one as in [Win87].

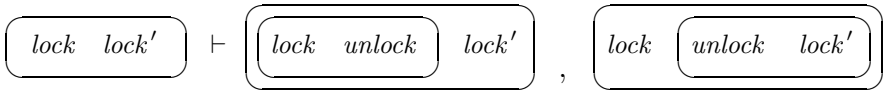
Definition 11 *A* structure sequent $\Gamma \vdash_\rho \Delta$ *(just* sequent *for short) consists of two finite sequences Γ and Δ of set systems and a matrix $\rho : \Gamma \to \Delta$ of set system morphisms.*

Let $\mathcal{C}$ and $\mathcal{D}$ be configuration structures. We call $\mathcal{D}$ a *substructure* of $\mathcal{C}$ if $\mathcal{D} \subseteq \mathcal{C}$ and moreover, for all $D \in \mathcal{D}$ and $C \in \mathcal{C}$, if $C \subseteq D$ then $C \in \mathcal{D}$. Note that $D \in sub(C)$ holds if and only if, viewing a configuration as the structure consisting of all its subconfigurations, D is a substructure of C. Hence we extend to structures the notation introduced for configurations, and write $\mathcal{D} \in sub(\mathcal{C})$ if $\mathcal{D}$ is a substructure of $\mathcal{C}$. Clearly, if $\mathcal{D} \in sub(\mathcal{C})$ the injection $|\mathcal{D}| \to |\mathcal{C}|$ is a set system morphism. Our notion of substructure is weaker than that used by Winskel [Win82, 2.3] for interpreting recursive processes in event structures. According to our definition, for example, $\{\{a\}, \{b\}, \{a, b\}, \{a, c\}\}$ is a substructure of $\{\{a\}, \{b\}, \{a, b\}, \{a, c\}, \{b, c\}\}$, but not according to Winskel's. It is easy to check that any substructure $\mathcal{D}$ of a monotone structure $\mathcal{C}$ is monotone.

A configuration structure $\mathcal{C}$ is said to *satisfy* a structure sequent $\Gamma \vdash_\rho \Delta$ when, for any $\mathcal{E} \in sub(\mathcal{C})$ and interpretation $\pi : \Gamma \to \mathcal{E}$, there exist a substructure $\mathcal{D}$ of $\mathcal{C}$, a component $\Delta_k \in \Delta$ and a morphism $q : \Delta_k \to \mathcal{D}$ such that $\mathcal{E} \in sub(\mathcal{D})$ and, for all i, the following diagram commutes.

$$\begin{array}{ccc} \Gamma_i & \xrightarrow{\rho_{ik}} & \Delta_k \\ {\scriptstyle \pi_i}\downarrow & & \downarrow{\scriptstyle q} \\ \mathcal{E} & \xrightarrow{\subseteq} & \mathcal{D} \end{array}$$

Example 7 In Example 2 we failed to find a poset sequent ruling out computations in which a process is granted a lock which is currently held by someone else. We succeed here with the following structure sequent, which is satisfied (not trivially now) by the structure of diagram (1).



It reads: if a configuration includes two *lock* actions by distinct processes P and Q, then there must exist either an *intermediate* state (the subconfiguration $\{lock, unlock\}$) where P has released the lock and Q has not yet acquired it *or* (the comma) one where Q has released the lock before P gets it. □

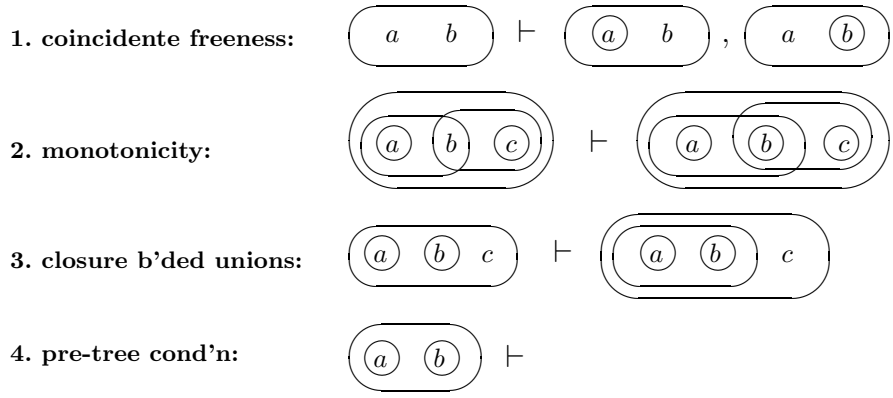


Table 2. Axiomatising classes of structures

Besides being used for specifying features of programming languages, such as the Java policy for granting locks, structure sequents allow compact axiomatisations of classes of models. In Table 2 we axiomatise some of the structure properties defined in Section 2. The axiom of monotonicity is understood in view of Theorem 2, point 4. As an example we prove the third axiom correct. This is in fact the only axiom in the table to rely on the additional assumption that configurations are finite.

Proposition 12 *If all configurations of a structure $\mathcal{C}$ are finite, then $\mathcal{C}$ is closed under binary bounded unions if and only if it satisfies axiom* 3 *of Table 2.*

Proof. We show the *if* implication (the converse is simpler). Let $\mathcal{C}$ satisfy axiom 3 and let the configurations $A, B, C \in \mathcal{C}$ be such that $A \cup B \subseteq C$. If $A \subseteq B$ or $B \subseteq A$ then $A \cup B \in \mathcal{C}$ trivially. Otherwise, let $a \in A - B$ and $b \in B - A$. Suppose there exists an event $c \in C - (A \cup B)$. Then, there exists an embedding

(ϕ, p) of the axiom's antecedent in $\mathcal{C}$ such that $\phi(\{a\}) = A$, $\phi(\{b\}) = B$ and $\phi(\{a, b, c\}) = C$. Since $\mathcal{C}$ satisfies the axiom, there must exists an embedding (ψ, q) of the succedent in $\mathcal{C}$, where $c \notin \psi(\{a, b\})$. Either $\psi(\{a, b\}) = (A \cup B)$, or there exists an event $c' \in \psi(\{a, b\}) - (A \cup B)$ and the argument repeats. Since configurations are finite we eventually conclude that $A \cup B \in \mathcal{C}$. □

The pre-tree condition characterises structures, called *pre-trees* in [Win82], such that, if two configurations A and B are *bound*, then either $A \subseteq B$ or $B \subseteq A$. Pre-trees are clearly closed under bounded unions. In table 3 we show a simple derivation of axiom 3 from axiom 4 by means of two inference rules from [Cen02]:

$$[\,\text{falsum}\,]\ \frac{\Gamma \vdash}{\Gamma \vdash_\sigma \Pi} \qquad [\,\circ\,]\ \frac{\mathcal{A} \vdash_r \mathcal{B}}{\mathcal{A}_{A \circ a} \vdash_{r+a} \mathcal{B}_{\phi_1(A) \circ a}, \ldots \mathcal{B}_{\phi_n(A) \circ a}}\ (*)$$

$(*)$ for all embeddings (ϕ_i, r) of $\mathcal{A}$ in $\mathcal{B}$. Here $\mathcal{A}_{A \circ a}$ denotes the set system obtained by adding a new fresh event a to a configuration $A \in \mathcal{A}$:

$$\mathcal{A}_{A \circ a} = \{X \subseteq |\mathcal{A}| \uplus \{a\} \mid X - \{a\} \in \mathcal{A} \text{ and } (a \in X \text{ iff } A \subseteq X)\},$$

where $\uplus$ is disjoint union. Then, $r + a$ denotes the vector whose componets (all equal) are the obvious extension of r to $|\mathcal{A}| \uplus \{a\}$. This rule adapts to structure sequents the rule [extend] of [Cen02] which was proven sound for poset sequents.

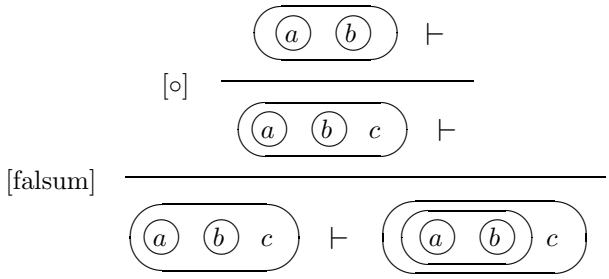


Table 3. pre-trees are closed under bounded unions

Proposition 13 *The* [falsum] *and* [∘] *rules and the structural of Table 1 (understood as referring to structure sequents) are sound.*

6 Logical Equivalence and Isomorphism of Structures

Set systems and their morphisms form a category $\mathcal{S}$ where identities and composition are those of sets. A morphism is *monic* in $\mathcal{S}$ (then called a *monomorphism*) if and only if it is one-to-one. It is *epi* if and only if it is onto. Note however that a morphism which is monic and epi need not be an *isomorphism* in $\mathcal{S}$. For example, the morphism $\{\{a, b\}\} \to \{\{a\}, \{b\}, \{a, b\}\}$ is one-to-one, onto, but it has no inverse in $\mathcal{S}$ (the third clause of Definition 9 forbids).

Lemma 14 *Let $\mathcal{A}$ and $\mathcal{B}$ be finitely branching set systems, and let $f : \mathcal{A} \to \mathcal{B}$ be a strict monomorphism; f is an isomorphism in $\mathcal{S}$ if and only if there exists a strict monomorphism $\mathcal{B} \to \mathcal{A}$.*

Proof. The *only if* implication is immediate. We show the *if*. Let $g : \mathcal{B} \to \mathcal{A}$ be a strict mono, and let $b \in B \in \mathcal{B}$. The sets $b^\dagger = \{x \in |\mathcal{B}| \mid x = (g\, f)^k(b), k \geq 0\}$ and $g(b)^\dagger = \{x \in |\mathcal{A}| \mid x = (f\, g)^k(g(b)), k \geq 0\}$ must both have finite cardinality because each of their elements must belong to a configuration of finite cardinality $|B|$ (because f and g are strict) and there is only a finite number of such configurations (because $\mathcal{A}$ and $\mathcal{B}$ are finitely branching). Moreover, it must be $|b^\dagger| = |g(b)^\dagger|$ because both f and g are one-to-one. Hence, there must exist $a \in g(b)^\dagger$ such that $f(a) = b$. Since b is arbitrary, $f : |\mathcal{A}| \to \mathcal{B}$ is onto, and hence an isomorphism of sets. Similarly, for all $X \subseteq |\mathcal{A}|$, $f(X) \in \mathcal{B}$ implies $A \in \mathcal{A}$. In fact the cardinality of the set $f(X)^\dagger = \{B \in \mathcal{B} \mid B = (g\, f)^k(f(X)), k \geq 0\}$ is finite and must be equal to that of $(f\, g)(X)^\dagger = \{A \in \mathcal{A} \mid A = (f\, g)^k(X), k > 0\}$. This shows that, for all $B \in \mathcal{B}$, $f^{-1}(B) \in \mathcal{A}$, and we can conclude that f is an isomorphism of set systems. □

We write $\mathcal{C} \equiv_\vdash \mathcal{D}$ when, for all *structure* sequents ρ, $\mathcal{C}$ satisfies ρ if and only if $\mathcal{D}$ satisfies ρ. This notion of logical equivalence is rather strong. We know that it is *not weaker* than bisimulation. For example, the bisimulation equivalent structures $\{\{a\}\}$ and $\{\{a\}, \{a\}\}$, respectively written a and $a + a$ in CCS, are distinguished by the sequent $ⓐ\ ⓐ \vdash ⓐ$. While this is generally considered an overdiscrimination, such expressive power may become desirable in the context of resource sensitive computation, for example when processes are *located*. In [CNL99] process algebra axiomatisations are studied which exclude the idempotence axiom $X + X = X$. The proposed argument is that a process $a + a$ exhibits a sort of *cold redundancy*, which makes it more tollerant to faults than a because it may take advantage of the different instances of the available resources.

Are there examples of logically equivalent structures which differ from each other in any "interesting" way? It is easy to find examples of logically equivalent structures which are not isomorphic because of cardinality. For instance $\mathcal{N} = \{\{0\}, \{1\}, \{2\} \dots\}$ and $\mathcal{R} = \{\dots \{2.5\}, \dots \{\pi\}, \dots\}$ (the reals). Indeed $\mathcal{R}$ can be mapped to $\mathcal{N}$ (for example by any constant function) so that both structures satisfy $\vdash \mathcal{R}$. In general, no sequent can distinguish the two. More interesting examples arise when restricting to *finite* sequents, that is sequents involving only finitely many events. No finite sequent distinguishes $\mathcal{N}$ from $\mathcal{N} + 1 = \{\{0\}, \{1\}, \{2\}, \dots \{0, 1, 2 \dots\}\}$, although there exists no morphism $\mathcal{N} + 1 \to \mathcal{N}$ and therefore no isomorphism.

We do not know whether, in general, logical equivalence implies bisimulation. However, we do know that, for tree-like structures, $\equiv_\vdash$ coincides with isomorphism, even when confined to the finite sequents (Corollary 16). To prove the result we find it convenient to work with a one-sided version of $\equiv_\vdash$. Given two monotone structures $\mathcal{C}$ and $\mathcal{D}$, we write $\mathcal{D} \sqsubseteq_\vdash \mathcal{C}$ if, whenever $\mathcal{D}$ satisfies a

(structure) sequent ρ, also $\mathcal{C}$ satisfies ρ. We call *strongly finitary* a configuration structure which is finitely branching and such that all configurations are finite.

Theorem 15 *Let $\mathcal{C}$ and $\mathcal{D}$ be strongly finitary pre-trees. If $\mathcal{D} \sqsubseteq_{\vdash} \mathcal{C}$ then there exists a strict monomorphism $\mathcal{C} \to \mathcal{D}$.*

Sketch of proof. We write $\mathcal{A}_{A;a}$ the set system defined as $\mathcal{A}_{A \circ a}$ (see Section 5) where $\subseteq$ is replaced by *strict* containment. Similarly, let A be a set and let $a, b \in A$. We write $A_{a \equiv b}$ the set obtained by identifying a and b in A. Formally, $A_{a \equiv b}$ is the quotient of A with respect to the smallest equivalence relation $\equiv$ such that $a \equiv b$. We write $[\_]$ the function mapping $x \in A$ to its equivalence class $[x] \in A_{a \equiv b}$. Given a set system $\mathcal{A}$ and $a, b \in |\mathcal{A}|$ we denote by $\mathcal{A}_{a \equiv b}$ the set system $\{X \subseteq |\mathcal{A}|_{a \equiv b} \mid X = [\_](A) \text{ for some } A \in \mathcal{A}\}$. If a and b are *conflicting* elements of $\mathcal{A}$ (that is, there exists no $A \in \mathcal{A}$ such that $a, b \in A$), the map $[\_] : |\mathcal{A}| \to |\mathcal{A}|_{a \equiv b}$ is a set system morphism $\mathcal{A} \to \mathcal{A}_{a \equiv b}$. The construction applies straightforwardly to labelled structures by requiring that a and b have the same label. Assume no strict monomorphism $\mathcal{C} \to \mathcal{D}$ exists. Since $\mathcal{C}$ is strongly finitary, it must have a finite substructure $\hat{\mathcal{C}}$ such that no strict monomorphism $\hat{\mathcal{C}} \to \mathcal{D}$ exists. Let Δ be the list of all (and only) the set systems of the form $\hat{\mathcal{C}}_{A;a}$, for all A and a such that a transition $A \to B$ is possible in $\hat{\mathcal{C}}$ and a has a label occurring in any $D \in \mathcal{D}$ such that $|D| \leq |\hat{\mathcal{C}}|$. Let $\rho : \hat{\mathcal{C}} \to \Delta$ be the matrix of the obvious injections. Similarly, let Γ be the list of all the set systems of the form $\hat{\mathcal{C}}_{a \equiv b}$, for all conflicting $a, b \in |\hat{\mathcal{C}}|$, and let $\sigma : \hat{\mathcal{C}} \to \Gamma$ be the matrix with projections $[\_] : \hat{\mathcal{C}} \to \hat{\mathcal{C}}_{a \equiv b}$ as components. Both Δ and Γ are finite since $\mathcal{C}$ is finitely branching, and hence $\hat{\mathcal{C}} \vdash_{\rho,\sigma} \Delta, \Gamma$ is a well formed structure sequent. The proof develops by showing that $\mathcal{D}$ satisfies (ρ, σ) while $\mathcal{C}$ does not. □

Corollary 16 *Two connected finitely branching pre-trees are logically equivalent ($\equiv_{\vdash}$) if and only if they are isomorphic.*

Proof. By Lemma 14 and Theorem 15.

7 Conclusions

We tried to understand how detailed a configuration theory can be in specifying the behaviour of a concurrent system. Two settings were studied: one in which (poset) sequents predicate over single configurations, and another where pieces of possibly incompatible computations may be addressed within the same (structure) sequent. In the first case we can place logical equivalence in between bisimulation and trace equivalence. This gives us a proof technique for the latter and one for invalidating the former. On the other hand, the logical equivalence arising from structure sequents was proven to coincide with isomorphism for tree-like structures, but no firm result was found for the general case. Still, structure sequents are shown to strech the expressive power of configuration theories to capture and reason about both model theoretic and language related properties.

References

[Cen02] P. Cenciarelli. Configuration Theories. In J. Bradfield, editor, *Proceedings CSL02*, pages 200–215. Springer LNCS 2471, 2002.

[CNL99] F. Corradini, R. De Nicola, and A. Labella. Models of Nondeterministic Regular Expressions. *Journal of Computer and System Sciences*, 59:412–449, 1999.

[DNM87] P. Degano, R. De Nicola, and U. Montanari. Observational Equivalence for Concurrency Models. In *Formal Description of Programming Concepts - Proc of 3rd IFIP WG 2.2*, pages 105–129. North-Holland, 1987.

[GG01] R.J. van Glabbeek and U. Goltz. Refinement of actions and equivalence notions for concurrent systems. *Acta Informatica*, 37:229–327, 2001.

[GP95] R.J. van Glabbeek and G.D. Plotkin. Configuration structures (extended abstract). In D. Kozen, editor, *Proceedings of LICS'95*, pages 199–209. IEEE Computer Society Press, June 1995.

[LC87] L. Pomello L. Castellano, G. De Michelis. Concurrency vs interleaving: an instructive example. In *Bull. EATCS 31*, pages 12–15, 1987.

[NPW81] M. Nielsen, G.D. Plotkin, and G. Winskel. Petri Nets, Event Structures and Domains: Part I. *Theoretical Computer Science*, 13(1):85–108, 1981.

[Pra86] V.R. Pratt. Modeling Concurrency with Partial Orders. *Int. Journal of Parallel Programming*, 15(1):33–71, 1986.

[Win82] G. Winskel. Event Structure Semantics of CCS and Related Languages. *Springer LNCS*, 140, 1982. Proceedings ICALP'82.

[Win87] Glynn Winskel. Event Structures. In G. Rozenberg W. Brauer, W. Reisig, editor, *Petri Nets: Applications and Relationships to Other Models of Concurrency*, number 255 in LNCS. Springer-Verlag, 1987.