Technical Report
NTCC Semantics based on Chu Spaces

Andrés A. Aristizábal P. Camilo Rueda*

Frank D. Valencia†

August, 2009

Abstract

By analyzing several extant models of concurrency and overviewing the most general and important concepts in the NTCC calculus, a timed extension of the CCP model, we explore the possibility of using Event Structures, a popular model for representing concurrency, so we can give an adequate semantics for each NTCC construct in a straightforward and understandable way. To give a precise representation by means of event structures we had to take into account several conditions such as guaranteeing true concurrency and establishing adequate representations for specific constructs. Finally, borrowing some concepts from Pratt, we construct a triadic event structure which could fulfill our requirements. Throughout the article we use an appropriate and compact representation of event structures such as Chu Spaces, since it is a more comfortable model to work with.

1 Introduction

1.1 Modeling Concurrency

Sequential and Concurrent composition, two very different but both well studied concepts, the first deals with events happening just one after the other, meanwhile the second one tries to represent multiple events executing at the same time. Since development of technology first dealt with procceses happening one after the other, the theory of sequentiallity appeared earlier and such theories as Automota became very popular. But, since the arrive of parallelism and new technolgy such as multi processors whom could deal with multiple processes, concurrency exploded and lots of models were developed to learn about this particular behavior. So, just as the sequential compositon in the earlier years of computer science, concurrency composition has achieved a great notority and a

*Pontificia Universidad Javeriana - Cali

†LIX Ecole Polythecnique

vast diversity of extant models started appearing, in such a way that this kind of behavior could be represented in a more precise way. There are lots of easily recognized models for representing sequential behavior such as Automata Theory and its formal languages which abstract them, but there are also different kinds of models which try to approach concurrency in a more desirable way.

1.1.1 Petri Nets and Mazurkiewicz traces

A petri net is a directed bipartite graph, where its nodes represent transitions, places its statements or so called conditions, and its directed arcs (which run between statementes and transtions) the way in which ecah place is a pre or post condition of a different transition. It is said that each place in a petri net may contain any non negative number of tokens which represent the truth value of the condition. A marking or state is the distribution of tokens over the places of the net. In this way, it can be said that a petri net can fire whenever there is a token at the end of all its input arcs (preconditions). And a firing results when all this tokens at the end of its input arcs are consumed and immediately placed at the end of its output arcs (postconditions). While a sequential run of a petri net is a sequence of firings, a concurrent run is a causal net, an acyclic petri net where every statement is a precondition of at most one transition and a postcondition of at most one transition, and every transition has at least one precondition and at least one postcondition.

Mazurkiewicz traces deal directly with Petri nets and its way of representing concurrency notions. They work in a very similar way with Petri nets as formal languages with state automata. The latter ones abstract state automata by means of strings and so do the first ones with petri nets by modeling their runs as equivalence classes of strings. Mazurkiewicz traces works identifying different transitions in an acyclic petri net by means of an important concept called action independence, indicating that both transitions fired independently. Even though Mazurkiewicz traces identify concurrency, it has a major problem, independence is global.

1.1.2 Posets and Pomsets

A poset is a set among with a binary relation which describes for certain pairs, that one of them precedes the other, that in such a way that the set can be partially ordered.
A pomset or partially ordered multiset is a Σ-labelled poset $(A, \leq, \Sigma,)$. That is, that for every event $a \in A$ there is a label $\lambda(a) \in \Sigma$. A pomset identifies concurrency by means of independence of events and not of actions.

1.1.3 Event Structures

As a simple and general approach we can say that an event structure is a pair (E, F) where E is the set of possible event occurrences and F is a family of

configurations, where a configuration is a set of events which occur at some stage of the process. An event structure must fulfill four axioms, they must be coherent, stable, coincidence-free and finitary.
A more precise and complete definition is the following:

Definition 1.1. An event structure $(A, con, \vdash)$ consists of three elements where:

- a set A of events
- Con, the consistency predicate, a non empty subset of ${}_fA$ (finite subsets of A) satisfying

$$X \in Con \wedge Y \subseteq X \Rightarrow Y \in Con$$

- The enabling relation $\vdash$, a subset of $Con \times A$ satisfying

$$X \vdash a \wedge X \subseteq Y \Rightarrow Y \vdash a$$

According to this definition we can state that a configuration F is a subset of events $x \in E$ which is consistent in the sense that any finite subset should be in the consistency predicate and it should be enabled by a set of events which had occurred previously in the process.

Definition 1.2. A prime event structure $(A, \leq, \#)$ consists of a partial order $(A, \leq)$ and a symmetric irreflexive binary relation $\#$ of conflict, satisfying the following condition: for all events $a, b, c \in A$, $a\#b$ and $b \leq c$ implies $a\#c$

Definition 1.3. A configuration of a prime event structure is a conflict-free downset i.e. given $a \in s$, if $b \leq a$ then $b \in s$ (downset), and if $a\#b$ then $b \notin s$

1.2 Constraint Systems

Definition 1.4. A simple constraint system is a structure $\langle D, \vdash \rangle$ where D is a non-empty set of tokens (which represents the constraints) and $\vdash \subseteq pD \times D$ (where pD is the set of finite subsets of D) an entailment relation satisfying the following:

- $u \vdash P$ whenever $P \in u$
- $u \vdash Q$ whenever $u \vdash P$ for all $P \in v$, and $v \vdash Q$

1.2.1 The CCP Model

CCP as well described by Saraswat [Sar93], is a powerful model of concurrent computation based on the notions of a store as a set of constraints and processes as information traducers.

The concurrent system in this model is composed by a collection of agents

which interact one with each other by means of shared variables. These concurrent agents compute partial information about the values of these variables, by means of constraints, which consequently are recorded in a store. This store is where all information or constraints over the variables reside. It monotonically grows in time. The store is considered the medium by which agents interact. There are two kinds of agents, the ones that change the state of the system by adding or telling information to the store via a new constraint and the ones that deduce or ask information from the store.

1.2.2 The NTCC calculus

A a general approach, we must say that a temporal CCP works as an extension of CCP by allowing agents to be constrained by time conditions. Specifically we define NTCC as a process calculus which studies temporal CCP as a model of concurrency for discrete-timed systems. This calculus generalizes the TCC calculus, which is a temporal CCP model for deterministic and synchronous timed reactive systems. It captures several aspects of timed systems. As tcc, it can model unit delays, time-outs, pre-emption and synchrony. Additionally it allows modeling unbounded but finite delays, bounded eventuality, asynchrony and nondeterminism.

2 Event Structures as Chu Spaces

2.1 A compact representation of an Event Structure

A Chu space is said to be a very nice and compact representation of an event structure. Since the most obvious and explicit way to write a Chu Space (A, X, Σ) is as binary matrix ($\Sigma : A \times X \to 2$ where $2 = \{0, 1\}$) of dimension $|A| \times |X|$ where A is the set of events and X is the set of states of the process represented by the Chu space, we can easily depict an event structure $\langle A, \leq, \# \rangle$ as that matrix. Of course the entries for that matrix come from the set Σ. An event $e \in A$ can be seen as a function $e : S \to \Sigma$ giving a value to that event in each state. Reciprocally, a state s can be seen as a function $s : A \to \Sigma$ giving a value for each event in that state. Obviously, $e(s) = s(e)$. For representing prime event structures the binary set $\Sigma = \{0, 1\}$ suffices, with 1 meaning that the event has occurred and with 0 just the opposite. In this way, each state corresponds to a configuration of the event structure (i.e. events e that occur in configuration s are all of those such that $s(e) = 1$)

Definition 2.1. A Chu space (A, S, Σ) consists of:

- a set A of events, $a \in A$ is a function $a : S \to \Sigma$
- a set S of states, $s \in S$ is a function $s : A \to \Sigma$

together with the condition

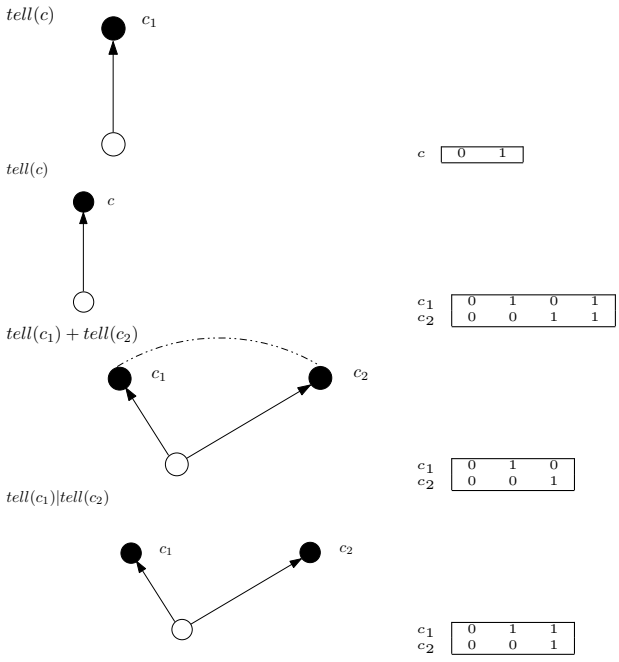
$$\forall_{a \in A, s \in S}. a(s) = s(a)$$

A distinction is usually made in event structures between events and actions. The occurrence of an event performs an action. An event is an instance of an action, so, in this way, two different events might perform the same action. In NTCC for instance, an event could be thought of as the addition of information while its action as the actual information it adds. The table below shows some processes, their events and their actions and how they differ in number according to its respective process:

process	events and actions
tell(c) ∥ tell(c)	two events, one action
(when a then do tell(c)) ∥ (when b then do tell(d))	four events, four actions
when a then do (tell(c) ∥ when b then do tell(d)) + when b then do (tell(d) ∥ when a then do tell(c))	four events, four actions

2.1.1 Representing NTCC constructs

Here we present the event structures and Chu spaces for some NTCC constructs which are straightforward to model using ordinary event structures and Chu spaces. (Note that arrows represent causal dependency $\leq$ and dotted arcs the conflict relation #):



The set of all configurations determine the event structure. These are the set of configurations according to the event structures mentioned before:

c	0	1			$\{\{\},\{c\}\}$
c_1	0	1	0	1	$\{\{\},\{c_1\},\{c_2\},\{c_1,c_2\}\}$
c_2	0	0	1	1	
c_1	0	1	0		$\{\{\},\{c_1\},\{c_2\}\}$, $c_1 \# c_2$
c_2	0	0	1		
c_1	0	1	1		$\{\{\},\{c_1\},\{c_1,c_2\}\}$, $c_1 \leq c_2$
c_2	0	0	1		

In the above we assume that events and actions are just the same. We say that what is observed of an event is the fact that the constraint involved in it has occurred (value 1 in some matrix entry). Here there is a major problem since a constraint cannot be an event. In the following section we will see how this difficulty can be overcomed.
Semantics for the ask operation seems to suggest that the constraint in the guard is actually posted, but the intended interpretation of a state is different. It refers to the fact that any information consistent with a state s should contain information of all actions of events e such that $s(e) = 1$

2.2 A labelled Approach

Since events produce observations other than themselves (actions), event structures has to be extended in order to cope with the extra information.

Definition 2.2. A labelled event structure $(A, \leq, \#, \lambda, V)$ consists of an event structure $(A, \leq, \#)$,and an event labelling function $\lambda : A \to V$

A labelled Chu space is defined smilarly. Let us now denote a the event posting any constraint c and the action $\lambda(a)$ associated to the event a equals to the particular post of constraint c. Let a A be the set of events and c a constraint, then $\exists_{a \in A}.\lambda(a) = c$

2.2.1 Our particular labels

- $\lambda :$ $events \to actions \cup \{\bot\}$ (Relates each event with an action)
- $\phi :$ $events \to \rho((actions, i)) \cup \{\bot\}$ (Relates each event to the set of actions in which it depends (ask) or not (unless) and the time unit in which the event associated to the action occurs)
- $\psi :$ $events \to \mathbb{N} \cup \{\bot\}$ (Relates each event with the instant of time when that event should be executed)
- $\beta :$ $events \to (events, i) \cup \{\bot\}$ (Relates each copy of an event to its seed and the number of that particular copy)
- $\omega :$ $events \to \rho(events) \cup \{\bot\}$ (Relates each event with the set of events which cannot be executed while that event is being executed (or))

2.2.2 Representing NTCC constructs

$tell(c)$:
$\exists_{a \in A}.\lambda(a) \vdash c$

a	0 1

$\lambda : a \mapsto c$, $\phi : a \mapsto \{\}$,
$\psi : a \mapsto 1$, $\beta : a \mapsto \bot$,
$\omega : a \mapsto \{\}$

when c_1 then do $tell(c_2)$:
$\exists_{a,b \in A}.\lambda(a) \vdash c_1 \wedge \lambda(b) \vdash c_2$

a	0	1	1
b	0	0	1

$\lambda : b \mapsto c_2, a \mapsto \bot$
$\phi : b \mapsto (c_1, 1), a \mapsto \bot$
$\psi : b \mapsto 1, a \mapsto \bot$
$\beta : b \mapsto \bot, a \mapsto \bot$
$\omega : b \mapsto \{\}, a \mapsto \bot$

$tell(c_1) \| tell(c_2)$:
$\exists_{a,b \in A}.\lambda(a) \vdash c_1 \wedge \lambda(b) \vdash c_2$

a	0	1	0	1
b	0	0	1	1

$\lambda : b \mapsto c_2, a \mapsto c_1$
$\phi : b \mapsto \{\}, a \mapsto \{\}$
$\psi : b \mapsto 1, a \mapsto 1$
$\beta : b \mapsto \bot, a \mapsto \bot$
$\omega : b \mapsto \{\}, a \mapsto \{\}$

$tell(c_1) + tell(c_2)$:
$\exists_{a,b \in A}.\lambda(a) \vdash c_1 \wedge \lambda(b) \vdash c_2$

a	0	1	0
b	0	0	1

$\lambda : b \mapsto c_2, a \mapsto c_1$
$\phi : b \mapsto \{\}, a \mapsto \{\}$
$\psi : b \mapsto 1, a \mapsto 1$
$\beta : b \mapsto \bot, a \mapsto \bot$
$\omega : b \mapsto \{a\}, a \mapsto \{b\}$

2.3 A Triadic solution for true concurrency

According to a very popular assumption such as fixed granularity, which states that actions in a system are built up from atomic actions which are only observable at the begining and at the end, but not in the period between, the interleaving law $(a \| b = ab + ba)$ holds.
True concurrent NTCC models aim at invalidating this interleaving law.

$$(a \to P) \| (b \to Q) = (a \to (P \| (b \to Q))) + (b \to (Q \| (a \to P)))$$

It is easy to show that in the ordinary Chu space and Event structure model of NTCC, this law still holds. Since simple event structures do not have the capacity to cope with extra information which may invalidate the interleaving law, the first option is to introduce extra events whose relation to the actual events are kept track via labels. For instance $tell(c_1) \| tell(c_2) st. \exists_{a,b \in A} \lambda(a) \vdash c_1 \wedge \lambda(b) \vdash c_2$ would involve three events, the third one modelling the fact that a and b could occur simultaneously. The problem with this approach is just that it obscures the notion of what is indeed an event in a system. Moreover, it makes it unnecesarily complex the definition of some constructs, in particular the parallel composition.
A much more adequate alternative is proposed by Pratt [Pra03], which intends to consider event occurrence not as a two valued fact but as a more nuanced

three-valued affair. Instead of having the two usual possibilities for an event such as done or not done, it adds a most important one, the transition which stays in between the ordinary ones. In that way, an event may be in one of three states: *not yet occurred, already occurred* and *occurring now.* Configurations would not be a set of events but a set of events together with their degree of occurence, (for instance, 1 or 2). In the Chu Space representation this simply amounts to consider $\Sigma = \{0, 1, 2\}$, with value 1 for *occurring* and 2 for *already occurred.*

2.3.1 Representing NTCC Constructs

Taking into account these new triadic Chu spaces, we have the semantics for some processes:

$tell(c)$:

$\exists_{a \in A}.\lambda(a) \vdash c$

a	0	1	2	2	2
b	0	0	0	1	2

$\lambda : a \mapsto c, \phi : a \mapsto \{\}$,
$\psi : a \mapsto 1, \beta : a \mapsto \bot$,
$\omega : a \mapsto \{\}$

when c_1 then do $tell(c_2)$:

$\exists_{a,b \in A}.\lambda(a) \vdash c_1 \wedge \lambda(b) \vdash c_2$

a	0	1	2	2	2
b	0	0	0	1	2

$\lambda : b \mapsto c_2, a \mapsto \bot$
$\phi : b \mapsto (c_1, 1), a \mapsto \bot$
$\psi : b \mapsto 1, a \mapsto \bot$
$\beta : b \mapsto \bot, a \mapsto \bot$
$\omega : b \mapsto \{\}, a \mapsto \bot$

$tell(c_1) \| tell(c_2)$:

$\exists_{a,b \in A}.\lambda(a) \vdash c_1 \wedge \lambda(b) \vdash c_2$

a	0	1	2	0	1	2	0	1	2
a	0	0	0	1	1	1	2	2	2

$\lambda : b \mapsto c_2, a \mapsto c_1$
$\phi : b \mapsto \{\}, a \mapsto \{\}$
$\psi : b \mapsto 1, a \mapsto 1$
$\beta : b \mapsto \bot, a \mapsto \bot$
$\omega : b \mapsto \{\}, a \mapsto \{\}$

$tell(c_1) + tell(c_2)$:

$\exists_{a,b \in A}.\lambda(a) \vdash c_1 \wedge \lambda(b) \vdash c_2$

a	0	1	2	0	0
b	0	0	0	1	2

$\lambda : b \mapsto c_2, a \mapsto c_1$
$\phi : b \mapsto \{\}, a \mapsto \{\}$
$\psi : b \mapsto 1, a \mapsto 1$
$\beta : b \mapsto \bot, a \mapsto \bot$
$\omega : b \mapsto \{a\}, a \mapsto \{b\}$

Notice how the fifth state in the parallel composition reflects the fact that both events a and b can be occuring at the same time.
For the whole proof see A

3 Our Chu Spaces semantics for NTCC

3.1 Some important definitions

- $1..n$ is the set of numbers where its elements go from 1 to n

- $neg(c)$ we use this as a function applied to a constraint c to know if it has a negation or not. This is used to differentiate between a dependence of an action by an ask and a one from an unless.
- 3^C means the possible states of a Chu space where the events belong to the set of events C and the valuations to the triadic set $\Sigma = \{0, 1, 2\}$

3.2 Skip

$[\![skip]\!] \triangleq (C, W, \lambda, \phi, \psi, \beta, \omega)$ where:

$C = E$

$\lambda(e) = \bot$

$\phi(e) = \bot$

$\psi(e) = \bot$

$\beta(e) = \bot$

$\omega(e) = \bot$

$W = \{w \in \{0\}^C\}$

3.3 Tell

$[\![tell(c)]\!] \triangleq (A, X, \lambda, \phi, \psi, \beta, \omega)$ where:

$A = E$

$\exists_{a \in A}.\lambda(a) \vdash c \wedge \forall_{e \in A - \{a\}}.\lambda(e) = \bot$

$$\phi(e) = \begin{cases} \{\} \; if \; e = a \\ \bot \; otherwise \end{cases}$$

$$\psi(e) = \begin{cases} 1 \; if \; e = a \\ \bot \; otherwise \end{cases}$$

$$\beta(e) = \bot$$

$$\omega(e) = \begin{cases} \{\} \; if \; e = a \\ \bot \; otherwise \end{cases}$$

$X = \{x \in 3^A | \forall_{a' \in A - \{a\}}.x(a') \neq 0 \Rightarrow x(a) = 2\}$

3.4 Ask

$[\![\text{when } c \text{ then do } P]\!] \triangleq (C, W, \lambda, \phi, \psi, \beta, \omega)\ [\![P]\!] = (A, X, \lambda_1, \phi_1, \psi_1, \beta_1, \omega_1)$ where:

$C = E$

Let Λ be the set of all possible λ labels of each possible Chu Space, then $\exists_{a \in C}.\lambda(a) = \bot \wedge \exists_{\lambda' \in \Lambda - \{\lambda\}}.\lambda'(a) \vdash c \wedge \forall_{e \in C}.\lambda(e) = \lambda_1(e)$

$$\phi(e) = \begin{cases} \phi_1(e) \cup \{(c,1)\}\ if\ \lambda_1(e) \neq \bot \\ \bot\ otherwise \end{cases}$$

$$\psi(e) = \begin{cases} \bot\ if\ e = a \\ \psi_1(e)\ otherwise \end{cases}$$

$$\beta(e) = \begin{cases} \bot\ if\ e = a \\ \beta_1(e)\ otherwise \end{cases}$$

$$\omega(e) = \begin{cases} \bot\ if\ e = a \\ \omega_1(e)\ otherwise \end{cases}$$

$W = \{w \in 3^C | \exists_{a' \in A}.\lambda(a') \neq \bot \wedge w(a') \neq 0 \Rightarrow w(a) = 2\}$

3.5 Choice

$[\![P+Q]\!] \triangleq (C, W, \lambda, \phi, \psi, \beta, \omega)\ [\![P]\!] = (A, X, \lambda_1, \phi_1, \psi_1, \beta_1, \omega_1)\ [\![Q]\!] = (B, Y, \lambda_2, \phi_2, \psi_2, \beta_2, \omega_2)$ where:

$C = E$

$W = \{w \in 3^C | w \downarrow_A \in X \vee w \downarrow_B \in Y\}$

$$\lambda(e) = \begin{cases} \lambda_1(e)\ if\ \lambda_1(e) \neq \bot \\ \lambda_2(e)\ if\ \lambda_2(e) \neq \bot \\ \bot\ otherwise \end{cases}$$

$$\phi(e) = \begin{cases} \phi_1(e)\ if\ \lambda_1(e) \neq \bot \\ \phi_2(e)\ if\ \lambda_2(e) \neq \bot \\ \bot\ otherwise \end{cases}$$

$$\psi(e) = \begin{cases} \psi_1(e)\ if\ \lambda_1(e) \neq \bot \\ \psi_2(e)\ if\ \lambda_2(e) \neq \bot \\ \bot\ otherwise \end{cases}$$

$$\beta(e) = \begin{cases} \beta_1(e)\ if\ \lambda_1(e) \neq \bot \\ \beta_2(e)\ if\ \lambda_2(e) \neq \bot \\ \bot\ otherwise \end{cases}$$

$$\omega(e) = \begin{cases} \omega_1(e) \cup \{e \in B | \lambda_2(e) \neq \bot\}\ if\ \lambda_1(e) \neq \bot \\ \omega_2(e) \cup \{e \in A | \lambda_1(e) \neq \bot\}\ if\ \lambda_2(e) \neq \bot \\ \bot\ otherwise \end{cases}$$

3.6 Parallel composition

$[\![P \| Q]\!] \triangleq (C, W, \lambda, \phi, \psi, \beta, \omega)\ [\![P]\!] = (A, X, \lambda_1, \phi_1, \psi_1, \beta_1, \omega_1)\ [\![Q]\!] = (B, Y, \lambda_2, \phi_2, \psi_2, \beta_2, \omega_2)$

$C = E$

$W = (\bigcup_{i \in \{i \in \mathbb{N} | (e \in \{e \in A | \lambda_1(e) \neq \bot\} \wedge i = \psi_1(e)) \vee (e \in \{e \in B | \lambda_2(e) \neq \bot\} \wedge i = \psi_2(e))\}} \{w = w_1 \cup w_2 \cup w_3 \cup w_4 | w_1 \in presentpast(P, i) \wedge w_2 \in presentpast(Q, i) \wedge w_3 = future(P, i) \wedge w_4 = future(Q, i) \wedge ask(P, Q, w) \wedge ask(Q, P, w) \wedge unless(P, Q, w) \wedge unless(Q, P, w) \wedge or(P, w) \wedge or(Q, w) \wedge finalstates(P, Q, w, w_1, w_2, w_3, w_4)\}) \cup \{e \in A | \lambda_1(e) = \bot\} \cap \{e \in B | \lambda_2(e) = \bot\}$

where the set of states W is the union of states from X and Y from chu spaces P and Q ordered by time and restricted to several conditions. Each time unit i determines a set of states from X and Y which had happened before or at that particular time. (this states are determined by function presentpast) and a set of states which happens after time unit i (future). All these states are restricted by other functions (ask,unless,or and finalstates).

- Ask: Guarantees that if an event has happened then each constraint in which it depends has associated an event that has happened at the established time unit.
- Unless: Guarantees that if an event has happened then each constraint in which it depends (which comes with a negation) has every event associated to it in 0 at the established time unit.
- Or: Guarantees that if an event has started its execution every event associated to in its Or set (ω) must be in 0.
- Finalstates: Guarantees that the the events which have a time unit less than i have reached a final state, which means the maximum amount of events, with ψ less than i, executed with all the restrictions stated before.
  For technical definitions see B.1

$$\lambda(e) = \begin{cases} \lambda_1(e)\ if\ \lambda_1(e) \neq \bot \\ \lambda_2(e)\ if\ \lambda_2(e) \neq \bot \\ \bot\ otherwise \end{cases}$$

$$\phi(e) = \begin{cases} \phi_1(e)\ if\ \lambda_1(e) \neq \bot \\ \phi_2(e)\ if\ \lambda_2(e) \neq \bot \\ \bot\ otherwise \end{cases}$$

$$\psi(e) = \begin{cases} \psi_1(e)\ if\ \lambda_1(e) \neq \bot \\ \psi_2(e)\ if\ \lambda_2(e) \neq \bot \\ \bot\ otherwise \end{cases}$$

$$\beta(e) = \begin{cases} \beta_1(e)\ if\ \lambda_1(e) \neq \bot \\ \beta_2(e)\ if\ \lambda_2(e) \neq \bot \\ \bot\ otherwise \end{cases}$$

$$\omega(e) = \begin{cases} \omega_1(e)\ if\ \lambda_1(e) \neq \bot \\ \omega_2(e)\ if\ \lambda_2(e) \neq \bot \\ \bot\ otherwise \end{cases}$$

3.7 Unless

$[\![\text{unless } c \; NextP]\!] \triangleq (C, W, \lambda, \phi, \psi, \beta, \omega)$ $[\![P]\!] = (A, X, \lambda_1, \phi_1, \psi_1, \beta_1, \omega_1)$

$C = E$

Let Λ be the set of all possible λ labels of each possible Chu Space, then $\exists_{a \in C}.\lambda(a) = \bot \wedge \forall_{\lambda' \in \Lambda - \{\lambda\}}.\lambda'(a) \not\vdash c \wedge \forall_{e \in C}.\lambda(e) = \lambda_1(e)$

$$\phi(e) = \begin{cases} \phi_1(e) \cup \{(\neg c, 1)\} \; if \; \lambda_1(e) \neq \bot \\ \bot \; otherwise \end{cases}$$

$$\psi(e) = \begin{cases} \bot \; if \; e = a \; or \; if \; \lambda_1(e) = \bot \\ \psi_1(e) \; otherwise \end{cases}$$

$$\beta(e) = \begin{cases} \bot \; if \; e = a \\ \beta_1(e) \; otherwise \end{cases}$$

$$\omega(e) = \begin{cases} \bot \; if \; e = a \\ \omega_1(e) \; otherwise \end{cases}$$

$W = \{w \in 3^C | \exists_{a' \in C - \{a\}}.w(a') \neq 0 \Rightarrow w(a) \neq 2\}$

3.8 Unbounded finite delay

$[\![\star P]\!] \triangleq (C, W, \lambda, \phi, \psi, \beta, \omega)$ $[\![P]\!] = (A, X, \lambda_1, \phi_1, \psi_1, \beta_1, \omega_1)$

$C = E$

$W = X$

Let e'_{ij} be events non obsevable in A and several clones of e_j which belongs to A. Then:

$\lambda(e) = \lambda_1(e)$

$\phi(e) = \phi_1(e)$

$\psi(e) = \psi_1(e)$

$\beta(e) = \beta_1(e)$

$\omega(e) = \omega_1(e) \cup \{e'_{jh} \in \{e \in A | \lambda_1(e) = \bot\} | h \in 1..\infty \wedge j \in 1..|\{e \in A | \lambda_1(e) \neq \bot\}|\}$

$\lambda(e'_{ij}) = \lambda_1(e_j)$

$\phi(e'_{ij}) = \phi_1(e_j)$

$\psi(e'_{ij}) = \psi_1(e_j)$

$\beta(e'_{ij}) = (e_j, i)$

$\omega(e'_{ij}) = \beta^{-1}(\omega_1(e_j), i) \cup \{e'_{jh} \in \{e \in A | \lambda_1(e) = \bot\} | h \in 1..\infty - \{i\} \wedge j \in 1..|\{e \in A | \lambda_1(e) \neq \bot\}|\} \cup \{e \in A | \lambda_1(e) \neq \bot\}$

For technical definitions see B.2

3.9 Replication

$[\![!P]\!] \triangleq (C, W.\lambda, \phi, \psi, \beta, \omega)$ $[\![P]\!] = (A, X, \lambda_1\phi_1, \psi_1, \beta_1, \omega_1)$

$C = E$

Let e'_{ij} be events non obsevable in A and several clones of e_j which belongs to A. Then:

$\lambda(e) = \lambda_1(e)$

$\phi(e) = \phi_1(e)$

$\psi(e) = \psi_1(e)$

$\beta(e) = \beta_1(e)$

$\omega(e) = \omega_1(e)$

$\lambda(e'_{ij}) = \lambda_1(e_j)$

$\phi(e'_{ij}) = \phi_1(e_j)$

$\psi(e'_{ij}) = \psi_1(e_j)$

$\beta(e'_{ij}) = (e_j, i)$

$\omega(e'_{ij}) = \beta^{-1}(\omega_1(e), i)$

$W = (\bigcup_{i \in \{i \in \mathbb{N} | (e \in \{e \in A | \lambda_1(e) \neq \bot\} \wedge i = \psi_1(e)) \vee (e \in \{e \in B | \lambda_2(e) \neq \bot\} \wedge i = \psi_2(e))\}} \{w = w_1 \cup w_2 | w_1 \in presentpast(i) \wedge w_2 = future(i) \wedge ask(w) \wedge ask(w) \wedge or(w) \wedge finalstates(w, w_1, w_2)\}) \cup \{e \in A | \lambda_1(e) = \bot\}\}$

where the set of states W is the union of states from X from chu space P and its copies ordered by time and restricted to several conditions. Each time unit i determines a set of states from X and Y which had happened before or

at that particular time. (this states are determined by function presentpast) and a set of states which happens after time unit i (future). All these states are restricted by other functions (ask,unless,or and finalstates).

- Ask: Guarantees that if an event has happened then each constraint in which it depends has associated an event that has happened at the established time unit.
- Unless: Guarantees that if an event has happened then each constraint in which it depends (which comes with a negation) has every event associated to it in 0 at the established time unit.
- Or: Guarantees that if an event has started its execution every event associated to in its Or set (ω) must be in 0.
- Finalstates: Guarantees that the the events which have a time unit less than i have reached a final state, which means the maximum amount of events, with ψ less than i, executed with all the restrictions stated before. For technical definitions see B.3

3.10 Next

$[\![Next\, P]\!] \triangleq (C, W, \lambda, \phi, \psi, \beta, \omega)$ $[\![P]\!] = (A, X, \lambda_1\phi_1, \psi_1, \beta_1, \omega_1)$

$C = E$

$W = X$

$\lambda(e) = \lambda_1(e)$

$$\phi(e) = \begin{cases} \perp \, if \, \lambda_1(e) = \perp \\ \{(c, i'') | (c, i') \in \phi_1(e) \wedge i'' = i' + 1\} \, otherwise \end{cases}$$

$$\psi(e) = \begin{cases} \perp \, if \, \lambda_1(e) = \perp \\ \psi_1(e) + 1 \, otherwise \end{cases}$$

$\beta(e) = \beta_1(e)$

$\omega(e) = \omega_1(e)$

3.11 Local

$[\![\text{local } x \text{ in } P]\!] \triangleq (C, W, \lambda, \phi, \psi, \beta, \omega)$ $[\![P]\!] = (A, X, \lambda_1, \phi_1, \psi_1, \beta_1, \omega_1)$ where:

$C = E$

$W = X$

For all events $e \in A$ $\lambda(e) = \exists_x \lambda(e)$

$\phi(e) = \phi_1(e)$

$\psi(e) = \psi_1(e)$

$\beta(e) = \beta_1(e)$

$\omega(e) = \omega_1(e)$

4 Algorithm for recovering the store

$i = max(\{i \in \mathbb{N} \mid e \in \{e \in A | \lambda(e) \neq \bot \wedge s(e) = 2\} \wedge \psi(e) = i\})$
$store = \{\}$
for $e \in \{e \in A | \lambda(e) \neq \bot \wedge \psi(e) = i\}$ **do**
  if $s(e) = 2$ **then**
    $store = store \cup \{\lambda(e)\}$
  else
    $store = store$
  end if
end for

5 Concluding Remarks and Future Work

The use of a new denotational semantics for NTCC based on Chu Spaces allows us to give an alternative mathematical meaning to such a relevant language for such important fields as biology, security, concurrency, constraint problems, etc.

Specifically, the use of Chu Spaces for representing each NTCC constructor allows us to guarantee several intrinsic properties from the calculus as well as another properties outside the language.

One of the essential features about using Chu Spaces for representing each constructor in NTCC is its versatily. We can use binary Chu Spaces and triadic Chu Spaces for different porpuses and to vefiry different properties. By using Chu Spaces with an arity of two we can guarantee interleaving and by using an arity of three we can represent true concurrency.

The use of Chu Spaces gives NTCC another important benefit, its generality, popularity and, of course, the amount of people who uses this particular structure as the mathematical basis for different languages.

As a future work we propose several improvements.
As a short term aim we propound the possibility of establishing an equiva-

lence between our semantics based on Chu Spaces proposed in this report, with NTCC's own denotational semantics in such a way that we can guaratee the importance and value of our work.

As mid term objectives we expect to use our same idea for representing a broader language such as CPP.

References

[BHMR94] Francisco Bueno, Manuel V. Hermenegildo, Ugo Montanari, and Francesca Rossi. From eventual to atomic locally atomic cc programs: A concurrent semantics. In *ALP '94: Proceedings of the 4th International Conference on Algebraic and Logic Programming*, pages 114–132, London, UK, 1994. Springer-Verlag.

[Hoa78] C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, 21(8):666–677, 1978.

[Mil82] R. Milner. *A Calculus of Communicating Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1982.

[Mil99] Robin Milner. *Communicating and mobile systems: the π-calculus*. Cambridge University Press, New York, NY, USA, 1999.

[NV04] Mogens Nielsen and Frank D. Valencia. Notes on timed ccp. In *In 4th Advanced Course on Petri Nets ICPN03. LNCS*. Springer-Verlag, 2004.

[Pra] Vaughan Pratt. Introduction to chu spaces.

[Pra03] Vaughan R. Pratt. Transition and cancellation in concurrency and branching time. *Mathematical. Structures in Comp. Sci.*, 13(4):485–529, 2003.

[Sar93] Vijay A. Saraswat. *Concurrent constraint programming*. MIT Press, Cambridge, MA, USA, 1993.

[SJG94] Vijay A. Saraswat, Radha Jagadeesan, and Vineet Gupta. Foundations of timed concurrent constraint programming. In *Proceedings of the Ninth Annual IEEE Symposium on Logic in Computer Science*, pages 71–80. IEEE Computer Press, 1994.

[SR90] Vijay A. Saraswat and Martin Rinard. Concurrent constraint programming. In *POPL '90: Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 232–245, New York, NY, USA, 1990. ACM.

[SRP91] Vijay A. Saraswat, Martin Rinard, and Prakash Panangaden. The semantic foundations of concurrent constraint programming. In *POPL '91: Proceedings of the 18th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 333–352, New York, NY, USA, 1991. ACM.

[SRP95] Vijay A. Saraswat, Martin Rinard, and Prakash Panangaden. Semantic foundations of concurrent constraint programming. pages 283–302, 1995.

[Val01] Frank D. Valencia. Temporal concurrent constraint programming. In *CP '01: Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming*, page 786, London, UK, 2001. Springer-Verlag.

[Win87] Glynn Winskel. Event structures. In *Proceedings of an Advanced Course on Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986-Part II*, pages 325–392, London, UK, 1987. Springer-Verlag.

[Win89] Glynn Winskel. An introduction to event structures. In *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, School/Workshop*, pages 364–397, London, UK, 1989. Springer-Verlag.

Appendices

A True concurrency - Invalidating the interleaving law

By means of a counter example it is very easy to show that with these semantics, the interleaving law does not hold and so, true concurrency is guaranteed. In this case we just use the triadic case, since the example is a lot shorter and doing it with the tetradic is really straightforward. So, we now have:

$$\exists_{a,b,c,d \in A} \lambda(a) = c_1 \wedge \lambda(b) = c_3 \wedge \lambda(c) = c_2 \wedge \lambda(d) = c_4$$
$$(c_1 \rightarrow tell(c_2) \| (c_3 \rightarrow tell(c_4))) = (c_1 \rightarrow (tell(c_2) \| c_3 \rightarrow tell(c_4))) + (c_3 \rightarrow (tell(c_4) \| c_1 \rightarrow tell(c_2)))$$

We traslate the first part into the new semantics:

$$(c_1 \rightarrow (tell(c_2) \| c_3 \rightarrow tell(c_4)))$$

Obtained by:

$c_1 \rightarrow tell(c_2)$:

a	0	1	2	2	2
c	0	0	0	1	2

$c_3 \rightarrow tell(c_4)$:

b	0	1	2	2	2
d	0	0	0	1	2

and so we have:

$c_1 \rightarrow tell(c_2) \| c_3 \rightarrow tell(c_4)$:

a	0	1	2	2	2	0	**1**	2	2	2	0	0	0	1	1	1	2	2	2	2	2	2	2	2
b	0	0	0	0	0	1	**1**	1	1	1	2	2	2	2	2	2	2	2	2	2	2	2	2	2
c	0	0	0	1	2	0	**0**	0	1	2	0	0	0	0	0	0	0	1	2	0	0	1	1	2	2
d	0	0	0	0	0	0	**0**	0	0	0	0	1	2	0	1	2	0	0	0	1	2	1	2	1	2

On the other hand we have the second part which we translate into the new semantics:

$$(c_1 \rightarrow (tell(c_2) \| c_3 \rightarrow tell(c_4))) + (c_3 \rightarrow (tell(c_4) \| c_1 \rightarrow tell(c_2)))$$

Obtained by

$tell(c_2) \| c_3 \rightarrow tell(c_4)$:

b	0	1	2	2	2	0	1	2	2	2	0	1	2	2	2
c	0	0	0	0	0	1	1	1	1	1	2	2	2	2	2
d	0	0	0	1	2	0	0	0	1	2	0	0	0	1	2

$tell(c_4) \| c_1 \rightarrow tell(c_2)$:

a	0	1	2	2	2	0	1	2	2	2	0	1	2	2	2
c	0	0	0	1	2	0	0	0	1	2	0	0	0	1	2
d	0	0	0	0	0	1	1	1	1	1	2	2	2	2	2

and

$c_1 \rightarrow (tell(c_2) \| c_3 \rightarrow tell(c_4)) :$

a	0	1	2	2	2	2	2	2	2	2	2	2	2	2	2	2
b	0	0	0	1	2	2	2	0	1	2	2	2	0	1	2	2	2
c	0	0	0	0	0	0	0	1	1	1	1	1	2	2	2	2	2
d	0	0	0	0	0	1	2	0	0	0	1	2	0	0	0	1	2

$c_3 \rightarrow (tell(c_4) \| c_1 \rightarrow tell(c_2)) :$

a	0	0	0	1	2	2	2	0	1	2	2	2	0	1	2	2	2
b	0	1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
c	0	0	0	0	0	1	2	0	0	0	1	2	0	0	0	1	2
d	0	0	0	0	0	0	0	1	1	1	1	1	2	2	2	2	2

and so we have:

$$(c_1 \rightarrow (tell(c_2) \| c_3 \rightarrow tell(c_4))) + (c_3 \rightarrow (tell(c_4) \| c_1 \rightarrow tell(c_2))) :$$

a	0	1	2	2	2	0	2	2	2	0	0	0	1	1	1	2	2	2	2	2	2	2	2	2
b	0	0	0	0	0	1	1	1	1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
c	0	0	0	1	2	0	0	1	2	0	0	0	0	0	0	0	1	2	0	0	1	1	2	2
d	0	0	0	0	0	0	0	0	0	0	1	2	0	1	2	0	0	0	1	2	1	2	1	2

Comparing the triadic Chu spaces representing the two sides of the equation we can observe that there are not the same, but just. There is only one state missing in the second Chu space compared to the first one. The state where events a and b are occurring simultaneously (written in bold face). That is enough to invalidate the interleaving law.

B Technical Definitions

B.1 Parallel Composition

$[\![P \| Q]\!] \triangleq (C, W, \lambda, \phi, \psi, \beta, \omega)\ [\![P]\!] = (A, X, \lambda_1, \phi_1, \psi_1, \beta_1, \omega_1)\ [\![Q]\!] = (B, Y, \lambda_2, \phi_2, \psi_2, \beta_2, \omega_2)$

$C = E$

$W = (\bigcup_{i \in \{i \in \mathbb{N} | (e \in \{e \in A | \lambda_1(e) \neq \bot\} \wedge i = \psi_1(e)) \vee (e \in \{e \in B | \lambda_2(e) \neq \bot\} \wedge i = \psi_2(e))\}} \{w = w_1 \cup w_2 \cup w_3 \cup w_4 | w_1 \in presentpast(P, i) \wedge w_2 \in presentpast(Q, i) \wedge w_3 = future(P, i) \wedge w_4 = future(Q, i) \wedge ask(P, Q, w) \wedge ask(Q, P, w) \wedge unless(P, Q, w) \wedge unless(Q, P, w) \wedge or(P, w) \wedge or(Q, w) \wedge finalstates(P, Q, w, w_1, w_2, w_3, w_4)\}) \cup \{e \in A | \lambda_1(e) = \bot\} \cap \{e \in B | \lambda_2(e) = \bot\}$

where:

$presentpast(P, i) = X \downarrow_{\{e \in A | \lambda_1(e) \neq \bot \wedge \psi_1(e) \leq i\}}$

$future(P, i) = \{0\}^{\{e \in A | \lambda_1(e) \neq \bot \wedge \psi_1(e) > i\}}$

$ask(P, Q, w) = \forall_{e \in \{e \in A | \lambda_1(e) \neq \bot\}}.w(e) \neq 0 \Rightarrow \forall_{(c, i') \in \phi_1(e)}.\neg neg(c).((\exists_{e' \in \{e'' \in A | \lambda_1(e'') \neq \bot \wedge \psi_1(e) = i'\}}.\lambda_1(e') = c \wedge w(e') = 2) \vee (\exists_{e' \in \{e'' \in B | \lambda_2(e'') \neq \bot \wedge \psi_1(e) = i'\}}.\lambda_2(e') = c \wedge w(e') = 2))$

$unless(P, Q, w) = \forall_{e \in \{e \in A | \lambda_1(e) \neq \bot\}}.w(e) \neq 0 \Rightarrow \forall_{(c, i') \in \phi_1(e)}.neg(c).((\forall_{e' \in \{e'' \in A | \lambda_1(e'') \neq \bot \wedge \psi_1(e) = i'\}}.\lambda_1(e') = c \wedge w(e') \neq 2) \wedge (\forall_{e' \in \{e'' \in B | \lambda_2(e'') \neq \bot \wedge \psi_1(e) = i'\}}.\lambda_2(e') = c \wedge w(e') \neq 2))$

$or(P,w) = \forall_{e \in \{e \in A | \lambda_1(e) \neq \bot\}}.w(e) \neq 0 \Rightarrow \forall_{e' \in \omega_1(e)} w(e') = 0$

$finalstates(P,Q,w,w_1,w_2,w_3,w_4) = \forall_{w'=w_1 \cup w_2 \cup w_3 \cup w_4}.ask(P,Q,w') \wedge ask(Q,P,w') \wedge unless(P,Q,w') \wedge unless(Q,P,w') \wedge or(P,w') \wedge or(Q,w').|\{e \in C | (\lambda_1(e) \neq \bot \wedge \psi_1(e) < i) \vee (\lambda_2(e) \neq \bot \wedge \psi_2(e) < i) \wedge w'(e) = 2\}| \leq |\{e \in C | (\lambda_1(e) \neq \bot \wedge \psi_1(e) < i) \vee (\lambda_2(e) \neq \bot \wedge \psi_2(e) < i) \wedge w(e) = 2\}|$

$$\lambda(e) = \begin{cases} \lambda_1(e) \; if \; \lambda_1(e) \neq \bot \\ \lambda_2(e) \; if \; \lambda_2(e) \neq \bot \\ \bot \; otherwise \end{cases}$$

$$\phi(e) = \begin{cases} \phi_1(e) \; if \; \lambda_1(e) \neq \bot \\ \phi_2(e) \; if \; \lambda_2(e) \neq \bot \\ \bot \; otherwise \end{cases}$$

$$\psi(e) = \begin{cases} \psi_1(e) \; if \; \lambda_1(e) \neq \bot \\ \psi_2(e) \; if \; \lambda_2(e) \neq \bot \\ \bot \; otherwise \end{cases}$$

$$\beta(e) = \begin{cases} \beta_1(e) \; if \; \lambda_1(e) \neq \bot \\ \beta_2(e) \; if \; \lambda_2(e) \neq \bot \\ \bot \; otherwise \end{cases}$$

$$\omega(e) = \begin{cases} \omega_1(e) \; if \; \lambda_1(e) \neq \bot \\ \omega_2(e) \; if \; \lambda_2(e) \neq \bot \\ \bot \; otherwise \end{cases}$$

B.2 Unbounded finite delay

$[\![\star P]\!] \triangleq (C, W, \lambda, \phi, \psi, \beta, \omega) \; [\![P]\!] = (A, X, \lambda_1, \phi_1, \psi_1, \beta_1, \omega_1)$

$C = E$

$W = X$

Let $i \in 1..\infty, \; j \in 1..|\{e \in A | \lambda_1(e) \neq \bot\}|$

Let $e \in \{e \in A | \lambda_1(e) \neq \bot\}$

Let $e_j \in \{e \in A | \lambda_1(e) \neq \bot\}$ and $\forall_{j \in 1..|\{e \in A | \lambda_1(e) \neq \bot\}|} \forall_{h \in 1..|\{e \in A | \lambda_1(e) \neq \bot\}|-\{j\}}.e_j \neq e_h$

Let $e'_{ij} \in \{e \in A | \lambda_1(e) = \bot\}$ and $\forall_{i \in 1..\infty} \forall_{j \in 1..|\{e \in A | \lambda_1(e) \neq \bot\}|} \forall_{h \in 1..\infty-\{i\}} \forall_{k \in 1..|\{e \in A | \lambda_1(e) \neq \bot\}|-\{j\}}.e'_{ij} \neq e'_{hk} \wedge e'_{ij} \neq e'_{ik} \wedge e'_{ij} \neq e'_{hj}$

then

$\lambda(e) = \lambda_1(e)$

$\phi(e) = \phi_1(e)$

$\psi(e) = \psi_1(e)$

$\beta(e) = \beta_1(e)$

$\omega(e) = \omega_1(e) \cup \{e'_{jh} \in \{e \in A | \lambda_1(e) = \bot\} | h \in 1..\infty \wedge j \in 1..|\{e \in A | \lambda_1(e) \neq \bot\}|\}$

$\lambda(e'_{ij}) = \lambda_1(e_j)$

$\phi(e'_{ij}) = \phi_1(e_j)$

$\psi(e'_{ij}) = \psi_1(e_j)$

$\beta(e'_{ij}) = (e_j, i)$

$\omega(e'_{ij}) = \beta^{-1}(\omega_1(e_j), i) \cup \{e'_{jh} \in \{e \in A | \lambda_1(e) = \bot\} | h \in 1..\infty - \{i\} \wedge j \in 1..|\{e \in A | \lambda_1(e) \neq \bot\}|\} \cup \{e \in A | \lambda_1(e) \neq \bot\}$

B.3 Replication

$[\![!P]\!] \triangleq (C, W.\lambda, \phi, \psi, \beta, \omega)$ $[\![P]\!] = (A, X, \lambda_1\phi_1, \psi_1, \beta_1, \omega_1)$

$C = E$

Let $i \in 1..\infty,\ j \in 1..|\{e \in A | \lambda_1(e) \neq \bot\}|$

Let $e \in \{e \in A | \lambda_1(e) \neq \bot\}$

Let $e_j \in \{e \in A | \lambda_1(e) \neq \bot\}$ and $\forall_{j \in 1..|\{e \in A | \lambda_1(e) \neq \bot\}|} \forall_{h \in 1..|\{e \in A | \lambda_1(e) \neq \bot\}| - \{j\}} . e_j \neq e_h$

Let $e'_{ij} \in \{e \in A | \lambda_1(e) = \bot\}$ and $\forall_{i \in 1..\infty} \forall_{j \in 1..|\{e \in A | \lambda_1(e) \neq \bot\}|} \forall_{h \in 1..\infty - \{i\}} \forall_{k \in 1..|\{e \in A | \lambda_1(e) \neq \bot\}| - \{j\}} . e'_{ij} \neq e'_{hk} \wedge e'_{ij} \neq e'_{ik} \wedge e'_{ij} \neq e'_{hj}$

then

$\lambda(e) = \lambda_1(e)$

$\phi(e) = \phi_1(e)$

$\psi(e) = \psi_1(e)$

$\beta(e) = \beta_1(e)$

$\omega(e) = \omega_1(e)$

$\lambda(e'_{ij}) = \lambda_1(e_j)$

$\phi(e'_{ij}) = \phi_1(e_j)$

$\psi(e'_{ij}) = \psi_1(e_j)$

$\beta(e'_{ij}) = (e_j, i)$

$\omega(e'_{ij}) = \beta^{-1}(\omega_1(e), i)$

$W = (\bigcup_{i \in \{i \in \mathbb{N} | (e \in \{e \in A | \lambda_1(e) \neq \bot\} \wedge i = \psi_1(e)) \vee (e \in \{e \in B | \lambda_2(e) \neq \bot\} \wedge i = \psi_2(e))\}} \{w = w_1 \cup w_2 | w_1 \in presentpast(i) \wedge w_2 = future(i) \wedge ask(w) \wedge ask(w) \wedge or(w) \wedge finalstates(w, w_1, w_2)\}) \cup \{e \in A | \lambda_1(e) = \bot\}\}$

where:

$presentpast(i) = \{x \in 3^C | e \in C \wedge \lambda(e) \neq \bot \wedge \psi(e) \leq i\}$

$future(i) = \{0\}^{\{e \in C | \lambda(e) \neq \bot \wedge \psi(e) > i\}}$

$ask(w) = \forall_{e \in \{e \in C | \lambda(e) \neq \bot\}}.w(e) \neq 0 \Rightarrow \forall_{(c,i') \in \phi(e)}.\neg neg(c).\exists_{e' \in \{e'' \in C | \lambda(e'') \neq \bot \wedge \psi(e) = i'\}}.\lambda(e') = c \wedge w(e') = 2$

$unless(w) = \forall_{e \in \{e \in C | \lambda(e) \neq \bot\}}.w(e) \neq 0 \Rightarrow \forall_{(c,i') \in \phi(e)}.neg(c).\forall_{e' \in \{e'' \in C | \lambda(e'') \neq \bot \wedge \psi(e) = i'\}}.\lambda(e') = c \wedge w(e') \neq 2$

$or(w) = \forall_{e \in \{e \in C | \lambda(e) \neq \bot\}}.w(e) \neq 0 \Rightarrow \forall_{e' \in \omega(e)} w(e') = 0$

$finalstates(w, w_1, w_2) = \forall_{w' = w_1 \cup w_2}.ask(w') \wedge unless(w') \wedge or(w').|\{e \in C | \lambda_(e) \neq \bot \wedge \psi(e) < i \wedge w'(e) = 2\}| \leq |\{e \in C | \lambda(e) \neq \bot \wedge \psi(e) < i) \wedge w(e) = 2\}|$

C Some examples

C.1 $tell(c)$

$\exists_{a \in E}.\lambda(a) = c$:

a	0	1	2

$\lambda : a \mapsto c$, $\phi : a \mapsto \{\}$,
$\psi : a \mapsto 1$, $\beta : a \mapsto \bot$,
$\omega : a \mapsto \{\}$

C.2 when c then do $tell(c')$

$\exists_{a \in E}.\lambda(a) = c \wedge \exists_{b \in E}.\lambda(b) = c'$:

a	0	1	2	2	2
b	0	0	0	1	2

$\lambda : b \mapsto c', a \mapsto \bot$
$\phi : b \mapsto (c, 1), a \mapsto \bot$
$\psi : b \mapsto 1, a \mapsto \bot$
$\beta : b \mapsto \bot, a \mapsto \bot$
$\omega : b \mapsto \{\}, a \mapsto \bot$

C.3 $tell(c) + tell(c')$

$\exists_{a \in E}.\lambda(a) = c \wedge \exists_{b \in E}.\lambda(b) = c'$:

a	0	0	0	1	2
b	0	1	2	0	0

$\lambda : b \mapsto c', a \mapsto c$
$\phi : b \mapsto \{\}, a \mapsto \{\}$
$\psi : b \mapsto 1, a \mapsto 1$
$\beta : b \mapsto \bot, a \mapsto \bot$
$\omega : b \mapsto \{a\}, a \mapsto \{b\}$

C.4 $tell(c) \| Next\ tell(c')$

$\exists_{a \in E}.\lambda(a) = c \wedge \exists_{b \in E}.\lambda(b) = c'$:

a	0	1	2	2	2
b	0	0	0	1	2

$\lambda : b \mapsto c', a \mapsto c$

$\phi : b \mapsto \{\}, a \mapsto \{\}$
$\psi : b \mapsto 2, a \mapsto 1$
$\beta : b \mapsto \bot, a \mapsto \bot$
$\omega : b \mapsto \{\}, a \mapsto \{\}$

C.5 unless c then do $tell(c')$

$\{a, a'\} = \{e \in E | \lambda(e) = c\} \wedge \exists_{b \in E}.\lambda(b) = c'$:

a	0	0	0	0	0	1	1	1	2	2	2
a'	0	0	0	1	2	0	1	2	0	1	2
b	0	1	2	0	0	0	0	0	0	0	0

$\lambda : a \mapsto \bot, a' \mapsto \bot, b \mapsto c'$
$\phi : a \mapsto \bot, a' \mapsto \bot, b \mapsto \{\}$
$\psi : a \mapsto \bot, a' \mapsto \bot, b \mapsto 1$
$\beta : a \mapsto \bot, a' \mapsto \bot, b \mapsto \bot$
$\omega : a \mapsto \bot, a' \mapsto \bot, \mapsto \{\}$

C.6 Next Next tell(c)

$\exists_{a \in E}.\lambda(a) = c$:

a	0	1	2

$\lambda : a \mapsto c$, $\phi : a \mapsto \{\}$,
$\psi : a \mapsto 3$, $\beta : a \mapsto \bot$,
$\omega : a \mapsto \{\}$