Undergraduate Thesis Proposal

Bisimilarity Theory for ntcc Calculus

Researcher
Luis Fernando Pino Duque

Project Supervisor
Juan Francisco Díaz Frias, Ph.D.
Professor
School of System's Engineering and Computing
Engineering Faculty
Universidad del Valle

Project Co-Supervisor
Frank D. Valencia, Ph.D.
CNRS Research Scientist (Chargé de Recherche) at
Laboratoire d'Informatique (LIX)
École Polytechnique de Paris
in the INRIA team COMÈTE.

Santiago de Cali - August 26, 2009

Contents

1 Introduction

In today's world, technology is one of the most important cores for the development of the society. This role has made access to high-tech devices increasingly frequent. Elements like Web, wireless networks, high capacity laptops, mobile devices, among others, have allowed advance towards information globalization, but carrying with it new challenges and problems that have to be solved for assure a reliable, correct and secure service.

Computer science offers a framework in which information technology can be formalized, allowing establishing conditions where they work correctly. There exist many factors that affect the correctness of such technologies, one of the most recent and challenging is the concurrency, and this consists in many processes making use of a system in a simultaneous way. It is here where an area of computer science named concurrency theory goes into action.

In this theory, process calculi are distinguished, its intention is to model and reason about concurrent systems. Such calculi are capable to express systems formally, hence it is possible to argue about them for obtaining correct results. There are many examples such as CCS [1], π-calculus [2], sπ-calculus (π-calculus for arguing about security), among others, they have been specialized for solving specific problems due to everyone of them count on with a modeling approach and an associated expressiveness (things that can be modeled with it). The calculus that will be studied is called `ntcc` calculus, which uses logic, constraints and can express process behavior along time.

This project aims to develop a bisimilarity theory for `ntcc` calculus, which will allow to analyze processes behavior and will make easier to implement tools associated with this calculus. It also aims to provide an initial verification prototype for `ntcc` calculus (with respect to bisimilarity theory), whose objective is to open the door for developing future applications that help to automate activities related with the use of this calculus.

1.1 Problem Description

Concurrency theory investigates how to analyze those systems where many processes act in a simultaneous way, and arguing about them for obtaining conclusions about correctness, security, reliability and other important aspects.

Process calculi are used for this purpose, because they allow making process modeling in concurrent systems, and depending of its specialization they are able to express, until certain point, a series of actions that can be object of study for a subsequent reasoning.

Calculus like CCS and π-calculus have developed a notion of equivalence called bisimilarity. This notion is very strong, since it allows to reason (in a simple way) about the behavioral equivalence between processes, and this is very useful for making verification,

[1] Calculus of Communicating Systems more information in:
`http://en.wikipedia.org/wiki/Calculus_of_communicating_systems`

[2] More information in: `http://en.wikipedia.org/wiki/Pi-calculus`

reasoning, and other tasks.

Concurrent constraint programming (CCP [3]) is a formalism which analyses concurrent systems using constraints and logic. It has been extended and specialized in order to model other important aspects. Especially asynchronous and non-deterministic behavior, having in mind the time units in which processes are executed. This extension has been called `ntcc`.

Such calculus has been developed during the last decade and it has a robust theoretical base. But there is a problem, this calculus lacks of a behavioral equivalence as the bisimilarity theory which is so strong and important in concurrency theory. It is here where this project aims to contribute to `ntcc`, by giving to the calculus a bisimilarity theory that allows to determine when two processes behaves equivalently according to the definition developed.

Once defined the theory, it is necessary to develop verification techniques, whose objective is to find a way to determine if two processes are bisimilar. By using these techniques it will be possible have in mind another important problem, which consists in the lack of practical tools on which it can be modeled and argued about the equivalences between processes written in this calculus. Hence, through using of the techniques developed it will be possible to make an implementation that will provide the possibility of establish if two processes are bisimilar or not.

Therefore, another objective pursued by this project is to implement an initial prototype for `ntcc` calculus, whose purpose is to use (in a practical way) the bisimilarity notion for modeling concurrent systems, this will be very specific to a subset of the calculus (due to its complexity) and will allow to open the gap in the use of bisimilarity for this kind of applications.

[3] Concurrent constraint programming - more information in:
`http://en.wikipedia.org/wiki/Concurrent_constraint_logic_programming`

2 Objectives

2.1 Main Objective

Propose a bisimilarity theory for `ntcc` calculus.

2.2 Specific Objectives

- Define the concept of bisimilarity for `ntcc` calculus.
- Develop an axiom set that will constitute the base of bisimilarity theory.
- Establish the properties derived from bisimilarity theory.
- Define verification techniques that will provide the possibility of determining the bisimilarity equivalence between the processes written in `ntcc` calculus
- Implement an initial prototype that allows describing processes in `ntcc` calculus and verifying if they are bisimilar.

3 Justification

Bisimilarity theory is one of the most representative and substantial equivalence in concurrency theory. As described in previous sections, most important calculi use this notion for several types of applications (formal verification, simulation, among others).

Likewise, `ntcc` calculus lacks of a behavioral equivalence like bisimilarity, so it is not possible to make applications that require it. This is why the importance of this project lies in the development of a bisimilarity theory for `ntcc` calculus, which will be innovative and will bring new forms of applying the calculus to problems of real life.

Moreover, as a complement of theoretical development, the project aims to open the gap in process verification with `ntcc` calculus. This will be reflected in the implementation of an initial prototype that will use bisimilarity notion.

The `ntcc` calculus has been chosen due to its importance in AVISPA research group, since this group has decided to strengthen this calculus through REACT project [4], which is in the conclusion phase and it will continue in REACT+ project. The main idea is to strengthen `ntcc` calculus for applying it to problems of real life. Hence, bisimilarity theory will contribute in the consolidation of such calculus, bringing new application possibilities.

In conclusion, through this project the `ntcc` calculus will be reinforced with a bisimilarity notion, and with an initial prototype that uses it in a practical way.

[4] Robust theories for Emerging Applications in Concurrency Theory
More information in: `http://cic.puj.edu.co/wiki/doku.php?id=grupos:avispa:react`

4 Background

4.1 Technical Background

4.1.1 Process Calculi[15]

The process calculi are a diverse family of related approaches to formally modeling concurrent systems. Process calculi provide a tool for the high-level description of interactions, communications, and synchronizations between a collection of independent agents or processes.

There are many different process calculi in the literature mainly agreeing in their emphasis upon algebra. The main representatives are CCS [4] , CSP [5] and the process algebra ACP [6, 7]. The distinctions among these calculi arise from issues such as the process constructions considered (i.e., the language of processes), the methods used for giving meaning to process terms (i.e. the semantics), and the methods to reason about process behavior (e.g., process equivalences or process logics). Some other issues addressed in the theory of these calculi are their expressive power, and analysis of their behavioral equivalences. It will be described some of the issues named previously.

The Language of Processes. A common feature of the languages of process calculi is that they pay special attention to economy. That is, there are few operators or combinators, each one with a distinct and fundamental role. Process calculi usually provide the following combinators:

- *Action*, for representing the occurrence of atomic actions.
- *Product*, for expressing the parallel composition.
- *Summation*, for expressing alternate course of computation.
- *Restriction* (or *Hiding*), for delimiting the interaction of processes.
- *Recursion*, for expressing infinite behavior.

We presuppose an infinite set $\mathcal{N}$ of *names* $a, b, ...$ and then introduce a set of *co-names* $\overline{\mathcal{N}} = \{\overline{a} \mid a \in \mathcal{N}\}$ disjoint from $\mathcal{N}$. The set of *labels*, ranged over by l and l', is $\mathcal{L} = \mathcal{N} \cup \overline{\mathcal{N}}$. The set of *actions* Act, ranged over by the boldface symbols $\mathbf{a}$ and $\mathbf{b}$ extends $\mathcal{L}$ with a new symbol τ. The action τ is said to be the *silent* (*internal* or *unobservable*) action. The actions a and $\overline{a}$ are thought of as being *complementary*, so we decree that $\overline{\overline{a}} = a$. The syntax of processes is given by:

$$P, Q, ... ::= 0 \mid \mathbf{a}.P \mid P + Q \mid P \mid\mid Q \mid P\backslash a \mid A\langle b_1, ..., b_n \rangle$$

Intuitive Description. The intuitive meaning of the process terms is as follows. The process 0 does nothing. $\mathbf{a}.P$ is the process which performs an atomic action $\mathbf{a}$ and then behaves as P. The summation $P + Q$ is a process which may behave as either P or Q. $P \mid\mid Q$ represents the parallel composition of P and Q. Both P and Q can proceed independently but they can also synchronize if they perform complementary actions. The restriction $P\backslash a$ behaves as P except that it cannot perform the actions a or $\overline{a}$. The names

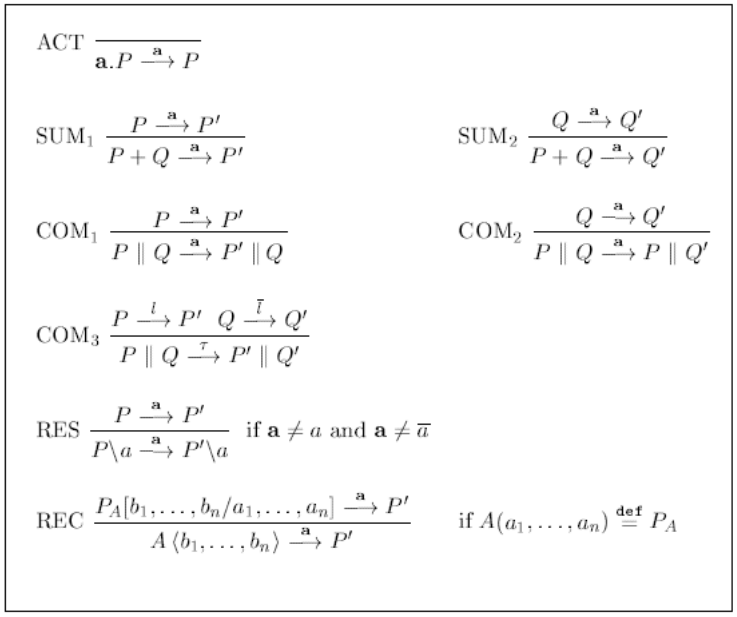

Figure 1: An operational semantics example, CCS (taken from [18], Page 35)

a and $\overline{a}$ are said to be *bound* in $P\backslash a$. $A\langle b_1, ..., b_n\rangle$ denotes the invocation to a unique recursive definition of the form $A(a_1, ..., a_n) \overset{def}{=} P_A$ where all the non-bound names of process P_A are in $\{a_1, ..., a_n\}$. Obviously P_A may contain invocations to A. The process $A\langle b_1, ..., b_n\rangle$ behaves as $P_A[b_1, ..., b_n/a_1, ..., a_n]$, i.e., P_A with each a_i replaced by b_i - with renaming of bound names wherever necessary to avoid captures.

Semantics of Processes. The methods by which process terms are endowed with meaning may involve at least three approaches: *operational, denotational* and *algebraic semantics*. Traditionally, CCS and CSP emphasize the use of the operational and denotational method, respectively, whilst the emphasis of ACP is upon the algebraic method. For this work it is important to describe operational semantics and behavioral equivalence, more specifically bisimilarity.

Operational semantics. An operational semantics interprets a given process term by using transitions (labeled or not) specifying its computational steps. A labeled transition $P \xrightarrow{a} Q$ specifies that P performs $\mathbf{a}$ and then behaves as Q. The relations $\xrightarrow{a}$ are defined to be the smallest which obey the rules in Figure 1. In these rules the transition below the line is to be inferred from those above the line.

4.1.2 Bisimilarity [?]

Once defined operational semantics, then it can be introduced the typical notions of process equivalence. Especially bisimilarity of CCS calculus, due to its importance in this project. We need a little notation: The empty sequence is denoted by ϵ. Given a sequence of actions $s = \mathbf{a}_1.\mathbf{a}_2.... \in Act^*$, define $\xrightarrow{s}$ as

$$(\xrightarrow{\tau})^* \xrightarrow{a_1} (\xrightarrow{\tau})^* ... (\xrightarrow{\tau})^* \xrightarrow{a_n} (\xrightarrow{\tau})^*$$

Notice that $\stackrel{\epsilon}{\Longrightarrow} = \xrightarrow{\tau}{}^*$. It is used $P \stackrel{s}{\Longrightarrow}$ to mean that there exists a P' s.t., $P \stackrel{s}{\Longrightarrow} P'$ and similarly for $P \xrightarrow{s}$.

Strong Bisimilarity. Intuitively, P and Q are strongly bisimilar if whenever P performs an action **a** evolving into P' then Q can also perform a and evolve into a Q' strongly bisimilar to P', and similarly with P and Q interchanged.

The above intuition can be formalized as follows. A symmetric relation B between process terms is said to be a strong bisimulation iff for all $(P, Q) \in B$,

$$\text{If } P \xrightarrow{a} P' \text{ then for some } Q', Q \xrightarrow{a} Q' \text{ and } (P', Q') \in B$$

We say that P is *strongly bisimilar* to Q, written $P \sim Q$ iff there exists a strong bisimulation containing the pair (P, Q).

Weak Bisimilarity. This version abstracts away from silent actions. Bisimilarity can be obtained by replacing the transitions $\xrightarrow{a}$ above with the (sequences of observable) transitions $\stackrel{s}{\Longrightarrow}$ where $s \in \mathcal{L}^*$. We shall use $\approx$ to stand for (weak) bisimilarity. Notice that $P \not\sim \tau.P$ but $P \approx \tau.P$.

4.1.3 Concurrent Constraint Programming [16]

In his seminal Ph.D. thesis [1], Saraswat proposed concurrent constraint programming as a model of concurrency based on the shared-variables communication model and a few primitive ideas taking root in logic. As informally described later, the ccp model elegantly combines logic concepts and concurrency mechanisms.

The ccp model. A concurrent system is specified in the ccp model in terms of *constraints* over the variables of the system. A constraint is a first-order formula representing *partial information* about the values of variables. As an example, for a system with variables x and y taking natural numbers as values, the constraint $x + y > 16$ specify possible values for x and y (those satisfying the inequation). The ccp model is parameterized by a *constraint system*, which specifies the constraints of relevance for the kind of system under consideration, and an *entailment relation* $\models$ between constraints (e.g, $x + y > 16 \models x + y > 0$

During a ccp computation, the state of the system is specified by an entity called the *store* in which information about the variables of the system resides. The store is represented as a constraint, and thus it may provide only partial information about the variables. Conceptually, the store in ccp is the *medium* through which agents interact with each other.

A ccp process can update the state of the system only by adding (or *telling*) information to the store. This is represented as the (logical) conjunction of the store representing

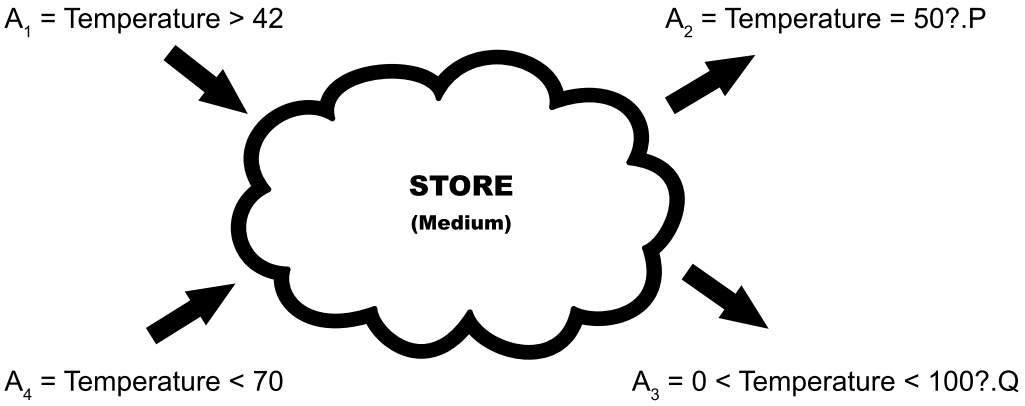

Figure 2: A simple CCP scenario

the previous state and the constraint being added. Hence, updating does not change the values of the variables as such, but constraints further some of the previously possible values.

Furthermore, ccp processes can synchronize by querying (or *asking)* information from the store. Asking is blocked until there is enough information in the store to *entail* (i.e., answer positively) the query, i.e. the ask operation determines whether the constraint representing the store entails the query.

A ccp computation terminates whenever it reaches a point, called a *resting* or a *quiescent* point, in which no more information can be added to the store. The output of the computation is defined to be the final store, also called *quiescent store*.

Example. (Taken from [16]) For making a clearer description of ccp model, consider the simple scenario illustrated in figure 2. There are four agents (or processes) wishing to interact through an initially empty store. Let, starting from the upper leftmost agent in a clockwise fashion, A_1, A_2, A_3 and A_4, respectively.

In this scenario, A_1 may move first and tell the others through the store the (partial) information that the temperature is greater than 42 degrees. This causes the addition of the item "temperature > 42" to the previously empty store.

Now A_2 may ask whether the temperature is exactly 50 degrees, and if so it wishes to execute a process P. From the current information in the store, however, the exact value of the temperature can not be entailed. Hence, the agent A_2 is blocked, and so is the agent A_3 since from the store it can not be determined either whether the temperature is between 0 and 100 degrees.

However, A_4 may tell the information that the temperature is less than 70 degrees. The store becomes "temperature $> 42 \wedge$ temperature < 70", and now process A_3 can execute Q, since its query is entailed by the information in the store. The agent A_2 is doomed to be blocked forever unless Q adds enough information to the store to entail its query.

The Language of Processes CCP. In the spirit of process calculi, the language of processes in the ccp model is given by a small number of primitive operators or combinators. A typical ccp process language contains the following operators:

- A *tell* operator, telling constraints (e.g., agent A_1 above).
- An *ask* operator, prefixing another process, its continuation (e.g. the agent A_2 above).
- *Parallel composition*, combining processes concurrently. For example the scenario in Figure 2 can be specified as the parallel composition of A_1, A_2, A_3 and A_4.
- *Hiding* (also called *restriction* or *locality*), introducing local variables, thus restricting the interface through which a process can interact with others.
- *Summation*, expressing a nondeterministic combination of agents to allow alternate courses of action.
- *Recursion*, defining infinite behavior.

It is worth pointing out that without summation, the ccp model is deterministic, in the sense that the final store is always the same, independently of the execution order (scheduling) of the parallel components[2].

CCP Syntax. *Processes $P, Q, ...$ in CCP are built from constraints in the underlying constraint system by the following syntax*:

$$P, Q := \mathbf{tell}(c) \mid \mathbf{when}\ c\ \mathbf{do}\ P \mid P \mid\mid Q \mid (\mathbf{local}\ x)P \mid q(x)$$

The process $\mathbf{tell}(c)$ adds the constraint c to the store. The process $\mathbf{when}\ c\ \mathbf{do}\ P$ asks if c can be deduced from the store. If so, it behaves as P. In other case, it waits until the store contains at least as much information as c. The parallel composition of P and Q is represented as $P \mid\mid Q$. The process $(\mathbf{local}\ x)P$ behaves like P, except that all the information on x produced by P can only be seen by P and the information on x produced by other processes cannot be seen by P. The process $q(y)$ is an identifier with arity $|y|$. We assume that every such an identifier has a unique (recursive) definition of the form

$$q(x) \stackrel{def}{=} Q$$

with x pairwise distinct and $|x| = |y|$. The process $q(y)$ behaves then as $Q[y/x]$.

4.1.4 CCP-based calculi and ntcc calculus [17]

Several extensions of the basic constructs presented above have been studied in the literature in order to provide settings for the programming and specification of systems with the declarative flavor of concurrent constraint programming. For this project, it is important to describe `ntcc` calculus, which is an extension to `tcc` model, including asynchronous and nondeterministic behavior.

Temporal CCP(`tcc`) The `tcc` model takes the view of reactive computation as proceeding *deterministically* in discrete time units (or time intervals). In other words, time is conceptually divided into discrete intervals. In each time interval, a deterministic CCP processes receives a stimulus (i.e. a constraint) from the environment, it executes with this stimulus as the *initial store* and when it reaches its resting point, it responds to the environment with the final store. Furthermore, the resting point determines a residual process, which is then executed in the next time interval.

The `tcc` calculus introduces constructs to (1) *delay* the execution of a process. And (2) *time-out* (or weak pre-emption) operations that waits during the current time interval for a given piece of information to be present. If it is not, they trigger a process in the *next time interval*.

Deterministic `tcc` syntax. Processes $P, Q, ...$ in tcc are built from constraints in the underlying constraint system by the following syntax:

$$P, Q := \mathbf{skip} \mid \mathbf{tell}(c) \mid \mathbf{when}\ c\ \mathbf{do}\ P \mid P \mid\mid Q \mid (\mathbf{local}\ x)P \mid \mathbf{next}\ P \mid \mathbf{unless}\ c\ \mathbf{next}\ P \mid\ !P$$

The processes $\mathbf{tell}(c)$, $\mathbf{when}\ c\ \mathbf{do}\ P$, $P \mid\mid Q$ and $(\mathbf{local}\ x)P$ are similar to those in CCP. The process $\mathbf{next}\ P$ delays the execution of P to the next time interval. The *time-out* $\mathbf{unless}\ c\ \mathbf{next}\ P$ is also a unit-delay, but P is executed in the next time unit iff c is not entailed by the final store at the current time interval. Finally, the *replication* $!P$ means $P \mid\mid \mathbf{next}\ P \mid\mid \mathbf{next}^2 P \mid\mid ...$, i.e., unboundly many copies of P but one at a time.

The `ntcc` calculus. The above syntax has been extended to deal with non-deterministic behavior and asynchrony in the `ntcc` calculus

Syntax of `ntcc`. The `ntcc` processes result from adding to the syntax of `tcc` the following constructs:

$$\sum_{i \in I} \mathbf{when}\ c_i\ \mathbf{do}\ P_i \mid\ \star P$$

The guarded-choice $\sum_{i \in I} \mathbf{when}\ c_i\ \mathbf{do}\ P_i$ where I is a finite set of indices, represents a process that, in the current time interval, must non-deterministically choose one of the P_j ($j \in I$) whose corresponding guard (constraint) c_j is entailed by the store. The chosen alternative, if any, precludes the others. If no choice is possible then the summation remains blocked until more information is added to the store.
The operator "$\star$" allows to express asynchronous behavior through the time intervals. Intuitively, process $\star P$ represents $P + \mathbf{next}\ P + \mathbf{next}^2 P + ...$, i.e., an arbitrary long but

finite delay for the activation of P.

4.1.5 State of the Art

This project aims to develop a bisimilarity theory for `ntcc` calculus, hence it is important to highlight other similar theories developed for the most representative process calculi, like CCS and π-calculus. Moreover, which tools using this theories have been developed. In the next section, those aspects are going to be presented.

1. **Scientific Background**

   - **CCS Bisimilarity[18].** As described previously (with more detail), bisimilarity can be defined like this:

     Definition. *A symmetric relation B between process terms is said to be a strong bisimulation iff for all $(P, Q) \in B$,*

     $$\text{If } P \xrightarrow{a} P' \text{ then for some } Q', Q \xrightarrow{a} Q' \text{ and } (P', Q') \in B$$

     We say that P is *strongly bisimilar* to Q, written $P \sim Q$ iff there exists a strong bisimulation containing the pair (P, Q).

   - **The π-calculus bisimilarity [18]** For the π-calculus the bisimilarity definition is analogous, the difference lies in the behavior of this processes, because they are not like in CCS. Then, there are some changes (that will not be mentioned completely) because of a higher complexity in the execution of them.

     Definition. A binary relation over $\mathcal{S}$ is a strong simulation if, whenever $P\mathcal{S}Q$,

     $$\text{If } P \xrightarrow{\alpha} A \text{ then exists a } B \text{ such that } A\mathcal{S}B \text{ and } Q \xrightarrow{\alpha} B$$

     If both $\mathcal{S}$ and its converse are strong simulations then $\mathcal{S}$ is a strong bisimulation. Two agents A and B are strongly equivalent, written $A \sim B$, if the pair (A, B) is in some strong bisimulation.

2. **Technological Background**

   - **The Edinburgh Concurrency Workbench (CWB)** [5] This workbench is an automated tool which caters for the manipulation and analysis of concurrent systems. In particular, the CWB allows for various equivalence, preorder and model checking using a variety of different process semantics. For example, with the CWB it is possible to:
     - define behaviors given either in an extended version of CCS or in SCCS, and perform various analyses on these behaviors, such as analyzing the state space of a given process, or checking various semantic equivalences and preorders;

[5] `http://homepages.inf.ed.ac.uk/perdita/cwb/summary.html`

- define propositions in a powerful modal logic and check whether a given process satisfies a specification formulated in this logic;
- play Stirling-style model-checking games to understand why a process does or does not satisfy a formula;
- derive automatically logical formulae which distinguish nonequivalent processes;
- interactively simulate the behavior of an agent, thus guiding it through its state space in a controlled fashion.

- **The Mobility Workbench (MWB)** [6] MWB is a similar to the previous tool, but its application is focused in π-calculus instead of CCS. In an analogous way this workbench is used to model concurrent systems and it allows to reason about equivalences, behaviors, among other functionalities.

4.2 Concepts

4.2.1 Process [8]

A process is an instance of a computer program, consisting of one or more threads, that is being sequentially executed by a computer system that has the ability to run several computer programs concurrently.

4.2.2 Calculus [9]

Calculus is referred to any method or system of calculation guided by the symbolic manipulation of expressions.

4.2.3 Concurrency [10]

Concurrency is a property of systems in which several computations are executing simultaneously, and potentially interacting with each other

4.2.4 Axiom [11]

An axiom is a proposition that is not proved or demonstrated but considered to be either self-evident, or subject to necessary decision. Therefore, its truth is taken for granted, and serves as a starting point for deducing and inferring other (theory dependent) truths.

4.2.5 Theorem [12]

A theorem is a statement proved on the basis of previously accepted or established statements such as axioms.

4.2.6 Constraint [16]

A constraint is a first-order formula representing *partial information* about the values of variables.

[6] `http://www.it.uu.se/research/group/mobility/mwb`

4.2.7 Semantics [14]

Considered as an application of mathematical logic, semantics reflects the meaning of programs or functions. In this regard, semantics permits programs to be separated into their syntactical part (grammatical structure) and their semantic part (meaning).

4.3 Project Context

Actually, a research project named REACT (*Robust theories for Emerging Applications in Concurrency Theory*) is in course. This project is a joint research effort between the AVISPA Research Group [7] (Universidad del Valle in agreement with Universidad Javeriana at Cali, Colombia -*which the researcher is a member*-), the Musical Representations Team at IRCAM [8](Institut de Recherche et Coordination Acoustique/Musique) and the INRIA Team Comète [9](LIX, École Polytechnique de Paris, France), the main objectives of this project focus on developing more robust CCP theories for dealing with applications in the areas of Security Protocols, Biology and Multimedia Semantic Interaction.

As it can be seen, the application of process calculus such as `ntcc` is important in different areas. This project will help for developing and implementing new tools based in these calculus, since this equivalence theory can be used as a support for determining similarities between processes in a different way than the existing ones in `ntcc` calculus.

Thus the main advantage is obtaining a theory that will allow performing automated process verification, which is widely used in the application areas above mentioned. In addition, providing to `ntcc` an initial prototype for process modeling using bisimilarity notion.

An additional advantage of this project is to consolidate relationship between AVISPA and Ecole Polytechnique at Paris, through teacher Frank Valencia (Project Co-Supervisor), since this project feeds the interests that have been built-up jointly.

5 Requirements

Basically it is required an undergraduate student dedicated full-time to the project, as well it is necessary a project supervisor that guides the development in order to execute the tasks directed to achieve the objectives. An expert in the topic, in this case the project co-supervisor, which is necessary to have constant communication for obtaining the information needed properly.

About technological resources, it is necessary to have a computer with enough capacities and tools in order to work correctly, in addition this equipment must have internet access for getting the information required for project development.

[7] `http://cic.puj.edu.co/wiki/doku.php?id=grupos:avispa:avispa`
[8] `http://www.ircam.fr/`
[9] `http://www.lix.polytechnique.fr/`

A basic budget is illustrated in the next figure:

Item	**Cost**
Researcher	5'000.000
Computing Equipment	2'000.000
Books, magazines, photocopies	400.000
Food and Transport	1'000.000
Indirect Costs	250.000
Total	8'650.000

6 Methodology

The execution of this project can be summarized in a set of five global activities, which encapsulate the development of every one of the objectives that has to be accomplished. Those activities are enumerated in the next way:

- **Bibliographic Review** In this phase it has to be performed a bibliographic review for the topics that will help to develop the project, i.e., the purpose is to go into concepts like: CCS and π-calculus bisimilarity, a wide knowledge in `ntcc` calculus and verification techniques. For this it will be carried out queries to articles, books, internet and interviews with experts in the topics. As a result it has to be obtained a strong theoretical base that will allow project development.

- **Definition of `ntcc` bisimilarity** For this stage the time will be dedicated mostly in formally defining bisimilarity concept, basically a formal revision of the concept will be done and a definition of the requirements that the notion must fulfill, so subsequently the definition can be done in a clear and concrete way. The result is a formal definition of bisimilarity.

- **Bisimilarity Properties** In this phase it will be done the definition of the properties that the notion (before defined) fulfills. For this, the formal base will be taken and it will be done proofs that allows to obtain the essential properties that bisimilarity meets. As a result bisimilarity now will have its properties.

- **Verification techniques and Prototype** For this phase the verification techniques that will be used for `ntcc` process verification are going to be defined. The purpose of this stage is to get a set of techniques that will be used in the implementation of the prototype, since it is aimed that this will be able to use bisimilarity notion by checking if two processes are bisimilar or not. With these techniques, the next step is the implementation of the prototype, whose function will be to do process verification with respect to bisimilarity. As a result there will be a prototype that uses the notion developed.

- **Research Practice and Final Report** In the final stage the last adjustments to the theory will be made, for this the researcher will carry out a research practice at École Polytechnique de Paris (France), where the work done will be verified. Once made the revision, then the final report will be made, so the results and contributions can be evaluated.

Stage	Estimated Time
Bibliographical Review	3 Months
Bisimilarity Definition	1 Month
Bisimilarity Properties	3 Months
Verification Techniques and Prototype	3 Months
Research Practice and Final Report	2 Months
Total Estimated	12 Months

References

[1] V. Saraswat, *Concurrent Constraint Programming*. The MIT Press, Cambridge, MA, 1993

[2] V. Saraswat, M. Rinard, y P. Panangaden. *The semantic foundations of concurrent constraint programming*. In *POPL '91*, pages 333-352, 21-23 January 1991.

[3] R. Milner. *Operational and Algebraic Semantics of Concurrent Processes*, pages 1203-1241. Elsevier, 1990.

[4] R. Milner. *Communication and Concurrency. International Series in Computer Science*. Prentice Hall, 1989. SU Fisher Research 511/24.

[5] C. A. R. Hoare. *Communications Sequential Processes*. Prentice-Hall, Englewood Cliffs (NJ), USA, 1985.

[6] J.A. Bergstra and J.W. Klop. *Algebra of communicating processes with abstraction*. Theoretical Computer Science, 37(1):77-121, 1985.

[7] J. C. M. Baeten and W. P. Weijland. *Process Algebra*. Cambridge University Press, 1990.

[8] Wikipedia Contributors. *Process (computing)* [online]. Wikipedia, The Free Encyclopedia, 2009 [query date: April 13, 2009].
Available at: `http://en.wikipedia.org/wiki/Process_(computing)`.

[9] Wikipedia Contributors. *Calculus* [online]. Wikipedia, The Free Encyclopedia, 2009 [query date: April 13, 2009].
Available at: `http://en.wikipedia.org/wiki/Calculus`.

[10] Wikipedia Contributors. *Concurrency* [online]. Wikipedia, The Free Encyclopedia, 2009 [query date: April 13, 2009].
Available at: `http://en.wikipedia.org/wiki/Concurrency_(computer_science)`.

[11] Wikipedia Contributors. *Axiom* [online]. Wikipedia, The Free Encyclopedia, 2009 [query date: April 13, 2009].
Available at: `http://en.wikipedia.org/wiki/Axiom`.

[12] Wikipedia Contributors. *Theorem* [online]. Wikipedia, The Free Encyclopedia, 2009 [query date: April 13, 2009].
Available at: `http://en.wikipedia.org/wiki/Theorem`.

[13] Wikipedia Contributors. *Constraint* [online]. Wikipedia, The Free Encyclopedia, 2009 [query date: April 13, 2009].
Available at: `http://en.wikipedia.org/wiki/Constraint_(mathematics)`.

[14] Wikipedia Contributors. *Semantics* [online]. Wikipedia, The Free Encyclopedia, 2009 [query date: April 13, 2009].
Available at: `http://en.wikipedia.org/wiki/Semantics`.

[15] Catuscia Palamidessi y Frank D. Valencia. *Languages for concurrency*. EATCS. 2006.

[16] Frank Valencia. *Decidability of Infinite-State Timed CCP Processes and First-Order LTL*. Theor. Comput. Sci. 330(3): 577-607. Elsevier. 2005

[17] Carlos Olarte, Camilo Rueda y Frank Valencia. *Concurrent Constraint Programming: Calculi, Languages and Emerging Applications*.

[18] R. Milner. *Communicating and Mobile Systems: the π-calculus*. Cambridge University Press, 1999.