A Gentle Introduction to Languages for Concurrency (A talk based on a recent essay by Palamidessi and Valencia).

Jorge A. Pérez

GRUPO DE INVESTIGACIÓN AVISPA

AVISPA Seminar, November 2006

Concurrency

Concurrency is concerned with the fundamental aspects of systems consisting of multiple computing processes that interact among each other

- What is fundamental in a system? Is there an absolute notion?
- Systems are certainly compositional. Do we have criteria for identifying parts?
- What is an interaction?
- More importantly, what is a process?

Concurrency

It's easy to think in real applications of systems exhibiting concurrency: robotic devices, modern operating systems, computer music, mobile communication systems, large databases, biological phenomena, etc, etc, etc,...

The problem is then to

- describe
- analyze
- reason about

the concurrent behaviour present in such systems in a precise and reliable way.

This talk focuses on main features for describing and analyzing systems.

Models of Concurrency

Models for sequential computation (i.e., the λ calculus) are clearly inadequate for representing concurrent computation, since ...

- they are meant to be finite
- they offer reduced mechanisms for interaction
- they're inherently deterministic

Concurrency then requires models capturing these features. They should be:

- Simple
- Expressive
- Formal
- Practical

As all models, models of concurrency are inherently abstracted versions of real phenomena, based upon some basic principles.

Models of Concurrency

There are several models of concurrency:

- Synchronous Communication, mainly in the form of process calculi
- True Concurrency, mainly as mathematical models inspired in category theory
- Concurrent Constraint Programming, a framework from which several calculi and programming languages are derived

Synchronous Communication

Agents synchronize on information, usually in a point-to-point fashion. This model is enforced by process calculi:

- languages in which the structure of terms represents the structure of processes
- equipped with an operational semantics that represents computational steps.

Some well-known process calculi:

- Robin Milner's CCS (Communicating Concurrent Systems)
- Tony Hoare's CSP (Communicating Sequential Processes)
- Jan Bergstra and Jan W. Klop's ACP (Algebra of Communicating Processes)

Synchronous Communication: Typical constructs

- The parallel composition construct, denoted $\|$.
  For two processes P and Q, $P \| Q$ denotes their interaction.
  A semantics could then say, given that P can evolve to P':

$$P \| Q \longrightarrow P' \| Q$$

- The summation construct, denoted $+$.
  For two processes P and Q, $P + Q$ denotes a choice between the two processes.
  A semantics could then say, given that P can evolve to P':

$$P + Q \longrightarrow P'$$

  or, given that Q can evolve to Q':

$$P + Q \longrightarrow Q'$$

Process Algebras or Process Calculi?

Process calculi give an algebraic treatment to processes:

- Constructors as operators of an algebraic theory. Example: the parallel composition operator.
- The semantics then exploits this algebraic flavor of processes.

True Concurrency

Models of concurrency can be classified according to the kind of concurrency they model:

- in interleaved concurrency systems are described in terms of *atomic actions*. Concurrency is represented by interleaving such actions: to express $A \parallel B$ we say "A then B" OR "B then A", and non-determinism enforces concurrency.
- in true concurrency actions occur simultaneously

Some examples of true concurrency models:

- Petri Nets, defined by Carl Adam Petri in 1962. Extend classic automata theory to deal with concurrent computation. Drawback: They are not compositional.
- Event Structures by Glynn Wynskel.
- Chu Spaces by Vaughn Pratt et al.

Concurrent Constraint Programming

A well-established model (yet not a popular one) based on the notion of partial information over a shared medium.

In addition to "standard" process calculi constructs, there are processes for

- including new information
- asking if a piece of information was ever included
- restricting who can see information and who can't

Processes interact by means of a shared memory (store). They have logic inference capabilities.

CCP therefore combines elements of mature models of concurrency and logic.

We argue that this dual nature could be crucial in several situations.

More on models

Models are generic in nature. They have been extended to cope with interesting phenomena:

- Time
- Mobility
- Broadcasting
- Persistent information

More recently, some extroverted applications of models of concurrency have lead to more specific extensions:

- Security
- Systems Biology
- Artistic applications
- Web services / Long transactions on web

There are many calculi (perhaps too many) that are extensions or improvements of some basic calculi.

Process Calculi

Now we examine main components/features in process calculi. Following our initial concern, we give some intuitions on mechanisms for

- description (language of processes)
- analysis (semantics)

The Language of Processes

A typical process calculus include operators for representing:

- actions
- composition (or interaction)
- choices
- restriction (or hiding)
- infinite behaviour
- Inaction (nil process)

Compare actions:

- CCS: input a and output $\overline{a}$ (a is a name, $\overline{a}$ is its co-name)
- π-calculus: input $x(y)$ and output $x\langle y\rangle$ (x, y are names or channels)
- CCP: adding and querying information: $\mathbf{tell}(x > 5)$ and $\mathbf{ask}(x > 0)\ \mathbf{then}\ P$ (x is a variable and P is a process)

Giving meaning to processes

There are at least three ways of endowing processes with a formal meaning:

- Operational: computation as steps
- Denotational: computation as mathematical structures
- Algebraic: computation as axioms

Operational Semantics

- A method proposed by Gordon Plotkin
- Processes evolve by means of labelled transition systems that formalize computational steps
- A process P that evolves to Q after performing an action a: $P \xrightarrow{a} Q$
- It is customary to define transition systems over configurations, or structures that contain a process and some useful information.
- Transition systems are particularly useful when designing correct implementations of process calculi

Operational Semantics: Examples

- In CCP we observe how both processes and the accumulated information evolve. So configurations are tuples $\langle$ process, store $\rangle$. A rule for adding new nformation could be:

$$\langle \mathbf{tell}(y = 5), x > 0 \rangle \longrightarrow \langle \mathbf{skip}, y = 5 \wedge x > 0 \rangle$$

- In calculi for synchronous communication an operational rule for syncronization could be stated as

$$\frac{P \xrightarrow{l} P' \quad Q \xrightarrow{\bar{l}} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'}$$

  In this case l and $\bar{l}$ are complementary actions for input and output. Their correct syncronization is denoted as the silent action τ. Notice that there no need for special configurations.

Behavioural Equivalences

- Once the operational behaviour has been formalized, it is natural to consider when two processes are equivalent.
  - Example: We would like to know if an implementation is equivalent to its specification
- Of course, there are many ways in which two processes can be compared
- The most popular behavioural equivalence is bisimulation
  - It involves paying attention to the states of each process, and to the possibilities still open at each state.
  - A bisimulation S is a relation between states such that whenever $P\ S\ Q$ then: if state P can do an action to get to state P', then state Q must also be able to do the same action to end in some state Q', where P' and Q' are themselves related by S.
  - There are many different variations of bisimulation, varying on criteria such as their discriminating power, the kind of actions they can handle, etc.

Denotational Semantics

- Defined by Christopher Strachey and Dana Scott
- Interprets processes as mathematical models; their meaning is thus given by mathematical structure
- Therefore, it defines a map $[\![\cdot]\!]$ from processes to such structures
- One strategy is to identify what relevant aspects can be observed from processes (observables) and then to equate a process with the observations that can be made of it.
- Example: In timed CCP one could argue that the denotational semantics of a process is given by the outputs (i.e., constraints) that a process emits to its environment over time

Algebraic Semantics

- Gives meaning to processes by stating laws (or axioms) equating process terms
- Processes and their operations are then seen as structures that obey these laws
- Classical example: the parallel composition operator

$$P \parallel \mathbf{skip} \equiv P, \quad P \parallel Q \equiv Q \parallel P, \quad P \parallel (Q \parallel R) \equiv (P \parallel Q) \parallel R$$

- A form of axiomatic semantics is structural congruence.
  - A set of axioms that defines equality on processes that have minor syntactic differences.
  - It's often used to give flexibility in the definition of an operational semantics. In fact, it might lead to compact definitions of LTS

Final Remarks

- Process calculi are particular expressions of models of concurrency. They capture specific issues of a given phenomena by means of a small set of operators.
- This has lead to many process calculi that model very specific phenomena
- There are different ways for giving formal meaning to processes. There is no absolute rule here: a calculus may provide different semantic approaches (or even combinations of them). One semantics might be more appropriate for certain kinds of analysis than the others.
- Common research problems involve proving if two semantics for the same language are equivalent; this kind of questions may reveal new insights on the nature of the calculi and/or their semantics.

Future Work / Invitation

A current challenge for us is that of checking how flexible CCP calculi can be in different contexts. In fact, our ntcc can be regarded as "generic" for modelling reactive systems.

- Is ntcc a sufficient basis for specialized applications?
- What happens with ntcc foundations when one needs to model systems in areas as different as music, biology and computer security?