Models of computation for concurrency: CSP & π

first activity in the thread on knowledge development within parallel/reconfigurable computing

Verónica Gaspes

Center for Research on Embedded Systems
School of Information Science, Computer and Electrical Engineering

December 9, 2009

Models of Computation for Concurrent Systems

Formal frameworks for providing the underlying semantics for a high-level concurrent or distributed programming language.

Why so many different ones?

- They explain different things: hardware, real-time, programs, protocols ...
- They can be used to formulate and prove different types of properties: correctness, deadlock freedom, timing, liveness ...
- CSP and the π-calculus are natural choices for describing concurrent processes that communicate through message passing. It is not a natural choice for describing abstract data types. It is not a natural choice for describing states with rich or complex data structures.

Models of Computation for Concurrent Systems

Formal frameworks for providing the underlying semantics for a high-level concurrent or distributed programming language.

Why so many different ones?

- They explain different things: hardware, real-time, programs, protocols . . .
- They can be used to formulate and prove different types of properties: correctness, deadlock freedom, timing, liveness . . .
- CSP and the π-calculus are natural choices for describing concurrent processes that communicate through message passing. It is not a natural choice for describing abstract data types. It is not a natural choice for describing states with rich or complex data structures.

Models of Computation for Concurrent Systems

Formal frameworks for providing the underlying semantics for a high-level concurrent or distributed programming language.

Why so many different ones?

- They explain different things: hardware, real-time, programs, protocols ...
- They can be used to formulate and prove different types of properties: correctness, deadlock freedom, timing, liveness ...
- CSP and the π-calculus are natural choices for describing concurrent processes that communicate through message passing. It is not a natural choice for describing abstract data types. It is not a natural choice for describing states with rich or complex data structures.

Models of Computation for Concurrent Systems

Formal frameworks for providing the underlying semantics for a high-level concurrent or distributed programming language.

Why so many different ones?

- They explain different things: hardware, real-time, programs, protocols ...
- They can be used to formulate and prove different types of properties: correctness, deadlock freedom, timing, liveness ...
- CSP and the π-calculus are natural choices for describing concurrent processes that communicate through message passing. It is not a natural choice for describing abstract data types. It is not a natural choice for describing states with rich or complex data structures.

Models of Computation for Concurrent Systems

Formal frameworks for providing the underlying semantics for a high-level concurrent or distributed programming language.

Why so many different ones?

- They explain different things: hardware, real-time, programs, protocols ...
- They can be used to formulate and prove different types of properties: correctness, deadlock freedom, timing, liveness ...
- CSP and the π-calculus are natural choices for describing concurrent processes that communicate through message passing. It is not a natural choice for describing abstract data types. It is not a natural choice for describing states with rich or complex data structures.

Communicating Sequential Processes

The most obvious application of the new ideas is to the specification, design, and implementation of computer systems which continuously act and interact with their environment.

The basic idea is that these systems can be readily decomposed into subsystems which operate concurrently and interact with each other as well as with their common environment.

The parallel composition of subsystems is as simple as the sequential composition of lines or statements in a conventional programming language.

Communicating Sequential Processes

The most obvious application of the new ideas is to the specification, design, and implementation of computer systems which continuously act and interact with their environment.

The basic idea is that these systems can be readily decomposed into subsystems which operate concurrently and interact with each other as well as with their common environment.

The parallel composition of subsystems is as simple as the sequential composition of lines or statements in a conventional programming language.

Communicating Sequential Processes

The most obvious application of the new ideas is to the specification, design, and implementation of computer systems which continuously act and interact with their environment.

The basic idea is that these systems can be readily decomposed into subsystems which operate concurrently and interact with each other as well as with their common environment.

The parallel composition of subsystems is as simple as the sequential composition of lines or statements in a conventional programming language.

Communicating Sequential Processes

The most obvious application of the new ideas is to the specification, design, and implementation of computer systems which continuously act and interact with their environment.

The basic idea is that these systems can be readily decomposed into subsystems which operate concurrently and interact with each other as well as with their common environment.

The parallel composition of subsystems is as simple as the sequential composition of lines or statements in a conventional programming language.

Communicating Sequential Processes

The most obvious application of the new ideas is to the specification, design, and implementation of computer systems which continuously act and interact with their environment.

The basic idea is that these systems can be readily decomposed into subsystems which operate concurrently and interact with each other as well as with their common environment.

The parallel composition of subsystems is as simple as the sequential composition of lines or statements in a conventional programming language.

Communicating Sequential Processes

[. . .] it avoids many of the traditional problems of parallelism in programming-interference, mutual exclusion, interrupts, multithreading, semaphores, etc.

[. . .] it provides a secure mathematical foundation for avoidance of errors such as divergence, deadlock and non-termination, and for achievement of provable correctness in the design and implementation of computer systems.

Communicating Sequential Processes

[. . .] it avoids many of the traditional problems of parallelism in programming-interference, mutual exclusion, interrupts, multithreading, semaphores, etc.

[. . .] it provides a secure mathematical foundation for avoidance of errors such as divergence, deadlock and non-termination, and for achievement of provable correctness in the design and implementation of computer systems.

References

CSP is one outcome of research at the Programming Research Group under the leadership of Prof. Sir Tony Hoare (since start in 1977) at Oxford University.
Since retirement Hoare is with Microsoft Research at Cambridge.

Some sources

www.wotug.org/
www.softeng.ox.ac.uk/subjects/CDS.html
research.microsoft.com/en-us/people/thoare/

I have used a book, Communicating Sequential Processes, first published 1985, revised in 2004 and publicly available under www.usingcsp.com/cspbook.pdf.

References

CSP is one outcome of research at the Programming Research Group under the leadership of Prof. Sir Tony Hoare (since start in 1977) at Oxford University.
Since retirement Hoare is with Microsoft Research at Cambridge.

Some sources

www.wotug.org/
www.softeng.ox.ac.uk/subjects/CDS.html
research.microsoft.com/en-us/people/thoare/

I have used a book, Communicating Sequential Processes, first published 1985, revised in 2004 and publicly available under www.usingcsp.com/cspbook.pdf.

References

CSP is one outcome of research at the Programming Research Group under the leadership of Prof. Sir Tony Hoare (since start in 1977) at Oxford University.
Since retirement Hoare is with Microsoft Research at Cambridge.

Some sources

www.wotug.org/
www.softeng.ox.ac.uk/subjects/CDS.html
research.microsoft.com/en-us/people/thoare/

I have used a book, Communicating Sequential Processes, first published 1985, revised in 2004 and publicly available under www.usingcsp.com/cspbook.pdf.

Short history

Started as a programming language in 1978 . . .

. . . evolved into a model of computation[a], a calculus, a mathematical theory for concurrency . . .

[a] introduces a semantic notion that can be incorporated into programming languages, e.g. rendezvous in Ada

. . . that has been used as foundation for the development of programming languages, hardware synthesis languages, specification and verification tools.

Short history

Started as a programming language in 1978 . . .

. . . evolved into a model of computation[a], a calculus, a mathematical theory for concurrency . . .

[a] introduces a semantic notion that can be incorporated into programming languages, e.g. rendezvous in Ada

. . . that has been used as foundation for the development of programming languages, hardware synthesis languages, specification and verification tools.

Short history

Started as a programming language in 1978 . . .

. . . evolved into a model of computation[a], a calculus, a mathematical theory for concurrency . . .

[a] introduces a semantic notion that can be incorporated into programming languages, e.g. rendezvous in Ada

. . . that has been used as foundation for the development of programming languages, hardware synthesis languages, specification and verification tools.

Short history

Started as a programming language in 1978 . . .

. . . evolved into a model of computation[a], a calculus, a mathematical theory for concurrency . . .

[a] introduces a semantic notion that can be incorporated into programming languages, e.g. rendezvous in Ada

. . . that has been used as foundation for the development of programming languages, hardware synthesis languages, specification and verification tools.

Objects and Events

Objects

think about objects in the world around us, which act and interact with us and with each other in accordance with some characteristic pattern of behaviour.

Events

Choose a different name for each type of behaviour

IN1KR	insertion of 1 SEK
IN5KR	insertion of 5 SEK
SMALL	extraction of small item
LARGE	extraction of large item
OUT1KR	extraction of 1 SEK

Objects and Events

Objects

think about objects in the world around us, which act and interact with us and with each other in accordance with some characteristic pattern of behaviour.

Events

Choose a different name for each type of behaviour

IN1KR	insertion of 1 SEK
IN5KR	insertion of 5 SEK
SMALL	extraction of small item
LARGE	extraction of large item
OUT1KR	extraction of 1 SEK

Objects and Events

Objects

think about objects in the world around us, which act and interact with us and with each other in accordance with some characteristic pattern of behaviour.

Events

Choose a different name for each type of behaviour

IN1KR	insertion of 1 SEK
IN5KR	insertion of 5 SEK
SMALL	extraction of small item
LARGE	extraction of large item
OUT1KR	extraction of 1 SEK

Objects and Events

These types of events are the alphabet of an object. Each occurence of an event is instantaneous or atomic: without duration![a]

[a] Time-consuming actions should be represented by a pair of events, the first denoting its start and the second denoting its finish. The duration of an action is represented by the internal between the occurrence of its start event and the occurrence of its finish event; during such an internal, other events may occur. Two extended actions may overlap in time if the start of each one precedes the finish of the other.

Time issues

No exact timing of events

Designs and reasoning about them are simplified, can be applied to physical and computing systems of any speed and performance.

Timely responses?

Treated independently of the logical correctness of the design.

Simultaneity?

When simultaneity of a pair of events is important (e.g. in synchronisation) it is a single event occurrence.

Process . . .

. . . the behaviour pattern of an object.[a]

[a] For Ada programmers: think of objects as tasks with entries. Think of processes as the sequences of entry calls. There is nothing similar in Java where objects and threads are isolated concepts.

Describing processes

1. Prefix
2. Recursion
3. Choice
4. Mutual recursion
5. Parallel composition
6. Non-deterministic choice
7. Interleaving
8. Sequencing

Process . . .

. . . the behaviour pattern of an object.[a]

[a] For Ada programmers: think of objects as tasks with entries. Think of processes as the sequences of entry calls. There is nothing similar in Java where objects and threads are isolated concepts.

Describing processes

1. Prefix
2. Recursion
3. Choice
4. Mutual recursion
5. Parallel composition
6. Non-deterministic choice
7. Interleaving
8. Sequencing

Process . . .

. . . the behaviour pattern of an object.[a]

[a] For Ada programmers: think of objects as tasks with entries. Think of processes as the sequences of entry calls. There is nothing similar in Java where objects and threads are isolated concepts.

Describing processes

1. Prefix
2. Recursion
3. Choice
4. Mutual recursion
5. Parallel composition
6. Non-deterministic choice
7. Interleaving
8. Sequencing

Operators

Prefix

$\mathrm{x} \rightarrow \mathrm{P}$ pronounced x then P

e.g.

$\textsc{in5kr} \rightarrow \mathrm{STOP}$

$\textsc{in5kr} \rightarrow \textsc{large} \rightarrow \mathrm{STOP}$

Choice

Behaviour can be influenced by the environment. With x different from y a process that does different things depending on whether x or y occur:

$\mathrm{x} \rightarrow \mathrm{P} | \mathrm{y} \rightarrow \mathrm{Q}$

e.g. on the whiteboard.

Recursion

$\mathrm{CLOCK} = \textsc{tick} \rightarrow \mathrm{CLOCK}$

$\mathrm{VMS} = \textsc{in5kr} \rightarrow \textsc{large} \rightarrow \mathrm{VMS}$

Parallel composition

A // B describes the process that behaves such that each event that actually occurs must be a possible event in the independent behaviour of each process separately.

e.g. and explanations on the whiteboard

Operators

Prefix

$\mathrm{x} \rightarrow \mathrm{P}$ pronounced x then P

e.g.

$\textsc{in5kr} \rightarrow \mathrm{STOP}$

$\textsc{in5kr} \rightarrow \textsc{large} \rightarrow \mathrm{STOP}$

Choice

Behaviour can be influenced by the environment. With x different from y a process that does different things depending on whether x or y occur:

$\mathrm{x} \rightarrow \mathrm{P} | \mathrm{y} \rightarrow \mathrm{Q}$

e.g. on the whiteboard.

Recursion

$\mathrm{CLOCK} = \textsc{tick} \rightarrow \mathrm{CLOCK}$

$\mathrm{VMS} = \textsc{in5kr} \rightarrow \textsc{large} \rightarrow \mathrm{VMS}$

Parallel composition

A // B describes the process that behaves such that each event that actually occurs must be a possible event in the independent behaviour of each process separately.

e.g. and explanations on the whiteboard

Operators

Prefix

$\mathrm{x} \rightarrow \mathrm{P}$ pronounced x then P

e.g.

$\textsc{in5kr} \rightarrow \mathrm{STOP}$

$\textsc{in5kr} \rightarrow \textsc{large} \rightarrow \mathrm{STOP}$

Recursion

$\mathrm{CLOCK} = \textsc{tick} \rightarrow \mathrm{CLOCK}$

$\mathrm{VMS} = \textsc{in5kr} \rightarrow \textsc{large} \rightarrow \mathrm{VMS}$

Choice

Behaviour can be influenced by the environment. With x different from y a process that does different things depending on whether x or y occur:

$\mathrm{x} \rightarrow \mathrm{P} | \mathrm{y} \rightarrow \mathrm{Q}$

e.g. on the whiteboard.

Parallel composition

$\mathrm{A} \;//\; \mathrm{B}$ describes the process that behaves such that each event that actually occurs must be a possible event in the independent behaviour of each process separately.

e.g. and explanations on the whiteboard

Operators

Prefix

$\mathrm{x} \rightarrow \mathrm{P}$ pronounced x then P

e.g.

$\textsc{in5kr} \rightarrow \mathrm{STOP}$

$\textsc{in5kr} \rightarrow \textsc{large} \rightarrow \mathrm{STOP}$

Choice

Behaviour can be influenced by the environment. With x different from y a process that does different things depending on whether x or y occur:

$\mathrm{x} \rightarrow \mathrm{P} | \mathrm{y} \rightarrow \mathrm{Q}$

e.g. on the whiteboard.

Recursion

$\mathrm{CLOCK} = \textsc{tick} \rightarrow \mathrm{CLOCK}$

$\mathrm{VMS} = \textsc{in5kr} \rightarrow \textsc{large} \rightarrow \mathrm{VMS}$

Parallel composition

A // B describes the process that behaves such that each event that actually occurs must be a possible event in the independent behaviour of each process separately.

e.g. and explanations on the whiteboard

Operators

Prefix

$\mathrm{x} \rightarrow \mathrm{P}$ pronounced x then P

e.g.

$\textsc{in5kr} \rightarrow \mathrm{STOP}$

$\textsc{in5kr} \rightarrow \textsc{large} \rightarrow \mathrm{STOP}$

Choice

Behaviour can be influenced by the environment. With x different from y a process that does different things depending on whether x or y occur:

$\mathrm{x} \rightarrow \mathrm{P} | \mathrm{y} \rightarrow \mathrm{Q}$

e.g. on the whiteboard.

Recursion

$\mathrm{CLOCK} = \textsc{tick} \rightarrow \mathrm{CLOCK}$

$\mathrm{VMS} = \textsc{in5kr} \rightarrow \textsc{large} \rightarrow \mathrm{VMS}$

Parallel composition

A // B describes the process that behaves such that each event that actually occurs must be a possible event in the independent behaviour of each process separately.

e.g. and explanations on the whiteboard

More operators

Non-deterministic choice $\sqcap$

Motivating example: The combination of change given by a machine may depend on the way in which the machine has been loaded with large and small coins (events not in the alphabet). Choice cannot be observed or influenced!

$$\begin{aligned}\mathrm{CH5D} = (\textsc{in5p} \rightarrow ((&\textsc{out1p} \rightarrow \textsc{out1p} \rightarrow \textsc{out1p} \rightarrow \textsc{out2p} \rightarrow \mathrm{CH5D}) \\ &\sqcap \\ &(\textsc{out2p} \rightarrow \textsc{out1p} \rightarrow \textsc{out2p} \rightarrow \mathrm{CH5D})))\end{aligned}$$

Used in specifications, not in implementations.

More operators

Non-deterministic choice ⊓

Motivating example: The combination of change given by a machine may depend on the way in which the machine has been loaded with large and small coins (events not in the alphabet). Choice cannot be observed or influenced!

$$\begin{aligned} CH5D = (&\textsc{in5p} \rightarrow ((\textsc{out1p} \rightarrow \textsc{out1p} \rightarrow \textsc{out1p} \rightarrow \textsc{out2p} \rightarrow CH5D) \\ &\sqcap \\ &(\textsc{out2p} \rightarrow \textsc{out1p} \rightarrow \textsc{out2p} \rightarrow CH5D))) \end{aligned}$$

Used in specifications, not in implementations.

More operators

Non-deterministic choice ⊓

Motivating example: The combination of change given by a machine may depend on the way in which the machine has been loaded with large and small coins (events not in the alphabet). Choice cannot be observed or influenced!

$$\begin{aligned}\mathrm{CH5D} = (\textsc{in5p} \rightarrow ((&\textsc{out1p} \rightarrow \textsc{out1p} \rightarrow \textsc{out1p} \rightarrow \textsc{out2p} \rightarrow \mathrm{CH5D}) \\ &\sqcap \\ &(\textsc{out2p} \rightarrow \textsc{out1p} \rightarrow \textsc{out2p} \rightarrow \mathrm{CH5D})))\end{aligned}$$

Used in specifications, not in implementations.

More operators

Non-deterministic choice ⊓

Motivating example: The combination of change given by a machine may depend on the way in which the machine has been loaded with large and small coins (events not in the alphabet). Choice cannot be observed or influenced!

$$\begin{aligned}\mathrm{CH5D} = (\textsc{in5p} \rightarrow ((&\textsc{out1p} \rightarrow \textsc{out1p} \rightarrow \textsc{out1p} \rightarrow \textsc{out2p} \rightarrow \mathrm{CH5D}) \\ &\sqcap \\ &(\textsc{out2p} \rightarrow \textsc{out1p} \rightarrow \textsc{out2p} \rightarrow \mathrm{CH5D})))\end{aligned}$$

Used in specifications, not in implementations.

Communication

Communication is a special form of event, where a channel and a value are involved.
Output: $\mathrm{c!v} \rightarrow \mathrm{P}$
Input: $\mathrm{c?x} \rightarrow \mathrm{P(x)}$

Let P and Q be processes, and let c be an output channel of P and an input channel of Q. When P and Q are composed concurrently in the system (P // Q), communication will occur on channel c on each occasion that P outputs a message and Q simultaneously inputs that message.

$\mathrm{c!v} \rightarrow \mathrm{P} \;//\; \mathrm{c?x} \rightarrow \mathrm{Q(x)}$

$\mathrm{c!v} \rightarrow \mathrm{P} \;//\; \mathrm{Q(v)}$

Communication

Communication is a special form of event, where a channel and a value are involved.
Output: $\mathrm{c!v} \rightarrow \mathrm{P}$
Input: $\mathrm{c?x} \rightarrow \mathrm{P(x)}$

Let P and Q be processes, and let c be an output channel of P and an input channel of Q. When P and Q are composed concurrently in the system (P // Q), communication will occur on channel c on each occasion that P outputs a message and Q simultaneously inputs that message.

$\mathrm{c!v} \rightarrow \mathrm{P} \;//\; \mathrm{c?x} \rightarrow \mathrm{Q(x)}$

$\mathrm{c!v} \rightarrow \mathrm{P} \;//\; \mathrm{Q(v)}$

Communication

Communication is a special form of event, where a channel and a value are involved.
Output: $\mathrm{c!v} \rightarrow \mathrm{P}$
Input: $\mathrm{c?x} \rightarrow \mathrm{P(x)}$

Let P and Q be processes, and let c be an output channel of P and an input channel of Q. When P and Q are composed concurrently in the system (P // Q), communication will occur on channel c on each occasion that P outputs a message and Q simultaneously inputs that message.

$\mathrm{c!v} \rightarrow \mathrm{P} \;//\; \mathrm{c?x} \rightarrow \mathrm{Q(x)}$

$\mathrm{c!v} \rightarrow \mathrm{P} \;//\; \mathrm{Q(v)}$

Communication

Communication is a special form of event, where a channel and a value are involved.
Output: $\mathrm{c!v} \rightarrow \mathrm{P}$
Input: $\mathrm{c?x} \rightarrow \mathrm{P(x)}$

Let P and Q be processes, and let c be an output channel of P and an input channel of Q. When P and Q are composed concurrently in the system (P // Q), communication will occur on channel c on each occasion that P outputs a message and Q simultaneously inputs that message.

$\mathrm{c!v} \rightarrow \mathrm{P} \;//\; \mathrm{c?x} \rightarrow \mathrm{Q(x)}$

$\mathrm{c!v} \rightarrow \mathrm{P} \;//\; \mathrm{Q(v)}$

π-calculus

Some sources

Jeannette Wing, FAQ on π
Milner & Parrow & Walker part I and II
Pierce on foundational calculi

The future

Bi-graphs! Check The Space and Motion of Communicating Agents by Robin Milner.

Transfering communication links

[. . .] the basic computational step is the transfer of a communication link between 2 processes; the recipient can then use the link for further interaction with other parties

In other words, the accesible resources can vary over time. It enables the specification and verification of concurrent systems with dynamically evolving communication topologies.

A printer server and a client

The server can send a along b: $\overline{b}a.S$

The client can receive a link along b in order to use it to send data: $b(c).\overline{c}d.P$

The interaction $\overline{b}a.S//b(c).\overline{c}d.P \xrightarrow{\tau} S//\overline{a}d.P$

Transfering communication links

[. . .] the basic computational step is the transfer of a communication link between 2 processes; the recipient can then use the link for further interaction with other parties

In other words, the accesible resources can vary over time. It enables the specification and verification of concurrent systems with dynamically evolving communication topologies.

A printer server and a client

The server can send a along b: $\overline{b}a.S$

The client can receive a link along b in order to use it to send data: $b(c).\overline{c}d.P$

The interaction $\overline{b}a.S//b(c).\overline{c}d.P \xrightarrow{\tau} S//\overline{a}d.P$

Transfering communication links

[. . .] the basic computational step is the transfer of a communication link between 2 processes; the recipient can then use the link for further interaction with other parties

In other words, the accesible resources can vary over time. It enables the specification and verification of concurrent systems with dynamically evolving communication topologies.

A printer server and a client

The server can send a along b: $\overline{b}a.S$

The client can receive a link along b in order to use it to send data: $b(c).\overline{c}d.P$

The interaction $\overline{b}a.S//b(c).\overline{c}d.P \xrightarrow{\tau} S//\overline{a}d.P$

Mobility

What distinguishes π-calculus from earlier process calculi is the ability to pass channels as data along other channels. This feature allows you to express process mobility, which in turn allows you to express changes in process structure. For example, suppose you're talking on your cell phone while driving in your car; the ability to model process mobility is useful for describing how your phone communicates with different base stations along the way.

An example

A server

$$!\text{incr}(a, x).\bar{a} < x + 1 >$$

this process expression says that the incr channel accepts two inputs: one is the name of the channel, a, which we will use to return the result of calling incr, and the other is the argument, x, which will be instantiated with an integer value upon a client call. After the call, the process will send back the result of incrementing its argument, x, on the channel a. The use of the replication operator, !, in the above process expression means that the incr server will happily make multiple copies of itself, one for each client interaction.

An example

A server

$$!\text{incr}(a, x).\bar{a} < x + 1 >$$

this process expression says that the incr channel accepts two inputs: one is the name of the channel, a, which we will use to return the result of calling incr, and the other is the argument, x, which will be instantiated with an integer value upon a client call. After the call, the process will send back the result of incrementing its argument, x, on the channel a. The use of the replication operator, !, in the above process expression means that the incr server will happily make multiple copies of itself, one for each client interaction.

An example

A client

$$(\nu a)(\overline{\text{incr}} < a, 17 > |a(y))$$

In parallel

1. Send on the incr channel both the channel a (for passing back the result value) and the integer value 17,
2. receive on the channel a the result y.

The use of the ν operator guarantees that a private channel of communication is set up for each client interaction with the incr server.

An example

A client

$$(\nu a)(\overline{\text{incr}} < a, 17 > | a(y))$$

In parallel

1. Send on the incr channel both the channel a (for passing back the result value) and the integer value 17,
2. receive on the channel a the result y.

The use of the ν operator guarantees that a private channel of communication is set up for each client interaction with the incr server.