COMMUNICATING SEQUENTIAL PROCESSES. THE FIRST 25 YEARS pdf

1: Sequential | Definition of Sequential by Merriam-Webster

This volume, like the symposium CSP25 which gave rise to it, commemorates the semi-jubilee of Communicating Sequential Processes. 1 Tony Hoare's paper "Communicating Sequential Processes" is today widely regarded as one of the most in?uential papers in computer science.

I am currently investigating further extensions to this logic, and working out the semantic underpinnings needed to validate logics that combine concurrency with procedures and communication. These languages combine simply-typed call-by-name procedures, and statically scoped blocks, with communication and shared-memory concurrency, respectively. I have introduced a family of trace-based semantic models suitable for modelling programs in several concurrency paradigms, including shared-memory as well as asynchronous communicating processes and CSP-style synchronously communicating processes. I have worked with my erstwhile students Shai Geva, Denis Dancanet, Kathy Van Stone, Michel Schellekens on various aspects of intensional semantics, and I continue to maintain an interest in this area. With Susan Older, I worked on models for various notions of fairness. My former student, Juergen Dingel, developed a methodology for systematic parallel programming. Background A mathematical model, or semantics, for a programming language can be used to support program analysis and synthesis, to validate a compiler, or may serve as the foundation for the design of a logic for program specification and proof. I work mainly on the development of denotational semantic models, characterized by the property that the meaning of a program phrase is defined in terms of the meanings of its syntactic sub-phrases. This structural property ensures support for compositional syntax-directed reasoning. I also work with operational semantic models, in which program behavior is typically explained in terms of the computational behavior of an abstract machine. I aim to use semantic models to guide the design of techniques and logics for program proving. My work aims to produce tractable semantic models whose structure is as simple as possible while accurately representing program behavior. This principle is usually expressed more formally as a desire for full abstraction with respect to the relevant notion of program behavior: Full abstraction means that the model is just concrete enough to support compositional program analysis in a manner faithful to the underlying notion of behavior. This principle can be used as a criterion for judging the utility of a semantic model with respect to a specific notion of program behavior. However, by itself full abstraction is not a panacea, and it can be difficult to achieve: Elegance is a virtue, and arguably a necessity. My foundational research has always been guided by such principles, seeking to simplify without over-simplification. My main focus is on concurrent programming languages. A long-term goal is to improve our ability to design efficient, correct parallel systems and provide foundational support for the development of tools to facilitate concurrent programming design and analysis. Concurrent programs are widely used, hard to get right, and difficult to analyze. In addition to partial correctness and total correctness, we often need to be able to establish safety properties, of the form that something bad never happens, and liveness properties, of the form that something good happens eventually. It is notoriously difficult to ensure that concurrent process interactions are sufficiently disciplined to preclude undesirable phenomena such as races, in which one process changes a piece of state that is being used by another process, with unreliable results. Rather than relying on possibly unrealistic assumptions about the granularity of hardware primitives, it is preferable to use semantic models and proof techniques that can guarantee correct behavior and the absence of races. Further, we need to find ways to avoid or mitigate the combinatorial explosion inherent in analyzing the ways in which concurrent processes might interact. In summary, my work addresses the need for semantic models, specification methods, and logics that support correctness proofs with a guarantee of race-freedom and absence of other runtime errors. Theoretical Computer Science, Vol. A Brief History of Shared Memory.

COMMUNICATING SEQUENTIAL PROCESSES. THE FIRST 25 YEARS pdf

2: CiteSeerX â€" A Distributed Protocol for Channel-Based Communication with Choice

Communicating Sequential Processes. The First 25 Years Symposium on the Occasion of 25 Years of CSP, London, UK, July , Revised Invited Papers.

It had a substantially different syntax than later versions of CSP, did not possess mathematically defined semantics, [10] and was unable to represent unbounded nondeterminism. In contrast to later versions of CSP, each process was assigned an explicit name, and the source or destination of a message was defined by specifying the name of the intended sending or receiving process. The parallel composition [west:: Roscoe developed and refined the theory of CSP into its modern, process algebraic form. In September , that book was still the third-most cited computer science reference of all time according to Citeseer albeit an unreliable source due to the nature of its sampling. Most of these changes were motivated by the advent of automated tools for CSP process analysis and verification. Applications[edit] An early and important application of CSP was its use for specification and verification of elements of the INMOS T Transputer , a complex superscalar pipelined processor designed to support large-scale multiprocessing. CSP was employed in verifying the correctness of both the processor pipeline, and the Virtual Channel Processor which managed off-chip communications for the processor. For example, the Bremen Institute for Safe Systems and Daimler-Benz Aerospace modeled a fault management system and avionics interface consisting of some 23, lines of code intended for use on the International Space Station in CSP, and analyzed the model to confirm that their design was free of deadlock and livelock. Similarly, Praxis High Integrity Systems applied CSP modeling and analysis during the development of software approximately , lines of code for a secure smart-card Certification Authority to verify that their design was secure and free of deadlock. Praxis claims that the system has a much lower defect rate than comparable systems. However, the "Sequential" part of the CSP name is now something of a misnomer, since modern CSP allows component processes to be defined both as sequential processes, and as the parallel composition of more primitive processes. The relationships between different processes, and the way each process communicates with its environment, are described using various process algebraic operators. Using this algebraic approach, quite complex process descriptions can be easily constructed from a few primitive elements. CSP provides two classes of primitives in its process algebra: Events Events represent communications or interactions. They are assumed to be indivisible and instantaneous. They may be atomic names e. Primitive processes Primitive processes represent fundamental behaviors: CSP has a wide range of algebraic operators. The principal ones are: Prefix The prefix operator combines an event and a process to produce a new process.

COMMUNICATING SEQUENTIAL PROCESSES. THE FIRST 25 YEARS pdf

3: Kahn process networks - Wikipedia

Get this from a library! Communicating Sequential Processes. The First 25 Years Symposium on the Occasion of 25 Years of CSP, London, UK, July , Revised Invited Papers.

The method includes generating a behaviorally equivalent CSP description of the business process and trace equivalent CSP description of event logs. Further the generation of CSP processes for a business process includes segregating a business process model into a set of workflow patterns with connectivity between the workflow patterns, generating a CSP process corresponding to each workflow pattern, composing the CSP processes in parallel with connectivity between the CSP processes, and synchronizing the CSP processes on common activities of the CSP processes. Lastly the generation of a CSP description of the event log is performed by constructing a CSP process for each trace in the event log and combining the CSP descriptions using external choice operator. For example, auditing can be performed by process aware information systems. Auditing been made mandatory in the U. Business activities need to be monitored for auditing an organization in conjunction with business process modeling and simulation. These process models can predict the behavior of systems. This calls for simultaneously tackling two problems, that of modeling a process and monitoring a process. Moreover, each event refers to a case process instance , an activity, and some additional data. A process log conforms to a process model if the process can replay each trace or event sequence in the log, i. One way to check conformance of a process is to enumerate all finite traces unraveling a loop only finite number of times if one such is present in the process, and then carry out membership testing for each of the trace in the log. This checking takes quadratic time in terms of a maximum length of the traces. However, the number of possible sequences generated by a process model may grow exponentially, in particular for a model showing concurrent behavior. Hence, it is often quite resource intensive, and thus expensive, to accomplish conformance testing. While systems, methods, and computer-readable media are described herein by way of examples and embodiments, those skilled in the art recognize that the invention is not limited to the embodiments or drawings described. Rather, the intent is to cover all modifications, equivalents and alternatives falling within the spirit and scope of the appended claims. Any headings used herein are for organizational purposes only and are not meant to limit the scope of the description or the claims. An arbitrary process log finite collection of event logs is used to create a simple trace-equivalent CSP process using prefix and external choice operators. This is called implementation. The problem also assumes the existence of a structured BPM process as a reference model. This is also translated to a CSP description using a pattern-oriented approach, in which this CSP model is referred to peas a specification. Finally, both models are fed into the FDR model checker, and it is determined if the implementation trace refines the specification. Metrics can be provided based on conformance checking, related to fitness, closeness, and appropriateness of the event logs with the reference process models. A business process flow must be properly modeled before being implemented as work flows. Using control elements such as and-splits, and-joins, or-splits, and or-joins various activities can be coordinated in a work flow. In a structured process, each split control element e. Further such split-join pairs are also properly nested. Of course, not all process models are structured. In fact, unstructured processes are widely used since they are more expressive. There are practical systems which allow for specification of structured processes only, e. Only structured processes are explicitly discussed herein as reference models for conformance checking. However, many unstructured processes can be converted to structured models preserving trace equivalence. Communicating Sequential Processes CSP is a formal language for describing patterns of interaction in concurrent systems. It is a member of the family of mathematical theories of concurrency known as process algebras, or process calculi. CSP has been practically applied in industry as a tool for specifying and verifying the concurrent aspects of a variety of different systems. The advantage gained by using CSP models are that they can be readily fed into the FDR model checker for automated analyses, such as refinement checking. The conformance checking problem is equivalent to trace refinement checking between the generated CSP processes. Process logs are translated to trace-equivalent CSP processes using prefix and external choice operators. This technique is implemented on

top of the FDR model checker. The compositional nature of structured processes allow them to be mapped easily to CSP descriptions. Moreover, using the FDR model checker, it is possible to list all the error traces in the log. Using this information metrics are defined related to fitness, closeness, and appropriateness, in accordance with the requirement of conformance checking. The control flow relation links two nodes in the graph showing the proper execution order in a BPD. An activity refers to the work required to achieve an objective. In a BPD, there are start events denoting the beginning of a process, and end events denoting the end of a process. In a flow graph the control flow relation linking two nodes is represented by a directed edge capturing the execution order between tasks of a BPD. A sequence is made of a node that has an incoming and an outgoing arc. It is represented by diamond. A fork AND-split node separates two concurrent paths and allows independent execution between concurrent paths within a BPD. It is modeled by connecting two or more outgoing control flow relations to a task. For synchronizing concurrent paths, a synchronizer AND-join is used so that it can link all the incoming edges to it. A synchronizer delays its completion until all incoming control flow relations leading into the task complete their execution. From a choice XOR-split node, two or more outgoing control flow relations diverge resulting in mutually exclusive alternative paths. This forces one to select only one alternative outgoing control flow relation at run-time. A merge XOR-join node is the counterpart of the choice node and connects incoming mutually exclusive alternative paths into one path. A BPM process is well-formed if and only if 1 it has exactly one start node with no incoming edges and one outgoing edge from it, 2 it has exactly one end node with one incoming edge to it and no outgoing edges, 3 there is only one incoming edge to a task and exactly one outgoing edge to a task, 4 each fork and choice has exactly one incoming edge and at least two outgoing edges, 5 each synchronizer and merge has at least two incoming edges and exactly one outgoing edge, 6 every node is on a path from the start node to some end node, and 7 there exists at least one task in any path from a split element to a join element this is to avoid triviality. Moreover, a well-formed BPM process is consistent if the underlying control flow graph is a directed acyclic graph, i. Unless otherwise mentioned, from now on, we shall consider only consistent BPM processes. The semantics of control elements of a BPM process are similar to that of workflows. For a split-parallel element fork , all the outgoing branches can be executed concurrently. Moreover, a join-parallel element synchronizer can be executed only when all of its incoming branches have been executed. Let us assume a split-choice choice element to have the semantics of an exclusive choice, i. A combination of split parallel and exclusive choice simulate the operation of an inclusive choice. Either of the following can be used as the semantics of a join-choice merge element. In case of single execution, the join-choice element is executed only once following whichever outgoing edge is completed while the other branches are discarded as and when they finish. In multiple executions semantics, whenever any of the incoming branches is completed the join choice element is executed, which may however give rise to multiple instances. A BPM process is sound if it does not produce deadlocks and lack of synchronization. A deadlock implies that the process will never terminate. A lack of synchronization allows multiple instances of the same task to occur to occur in a process. Multiple instances can lead to undesirable results, such as redundant activities, competition for resources, and dangling activities e. Given a BPM process P, an execution path exp can be defined as a sequence of nodes, such that two consecutive nodes belong to the control relation, and exp begins and ends at the unique start and final event respectively. A fragment of an execution path is given by f exp which may not necessarily begin at a start node or end at an end node execution paths restricted to tasks are considered. An ordinary trace is a fragment of a complete trace, which again projects a fragment of an execution on the set of activities only. The set of all legal traces of a process P is denoted as Tp. This definition of a BPM process closely matches with that of a workflow. However, only structured BPM processes are used, which are much like structured workflows. Structured processes can be represented in XML such that split and join control nodes correspond to start and end tags, as is followed in BPEL notation. A BPM process is structured if and only if it can be built inductively as follows: If P1 and P2 are structured processes and P1 contains an edge e, the result of replacing e with P2 in P1 still makes the resulting process structured. In Communicating Sequential Process CSP , a process depicts a kind of behavior, a behavior is made of events which are atomic and asynchronous between the environment which can be another process and the process. A simplified syntax

of CSP is shown as follows:

COMMUNICATING SEQUENTIAL PROCESSES. THE FIRST 25 YEARS pdf

4: Communicating sequential processes - Wikipedia

The First 25 Years: Symposium on the Occasion of 25 Years of CSP, London, UK, July , Revised Invited Papers. [Ali E Abdallah; Cliff B Jones; Jeff W Sanders] -- This book commemorates the work done by Tony Hoare and published under the title Communicating Sequential Processes in the August issue of the Communications of ACM.

Events CSP is built on the idea of events. An event is a synchronisation that several concurrent processes can engage in. CHP has barriers, which are a synchronisation that several concurrent processes can engage in, and thus they serve as a useful stand-in for events. The main difference is that barriers have the notion of membership, support dynamically enrolling on and resigning from the barrier at run-time â€" and they need to be allocated before use. Processes CSP separates event synchronisations the act of engaging in a single event from processes which can be composed of many synchronisations ; in Haskell terms they should have different types. However, separating the types of an event synchronisation from a process would bring a whole load of pain â€" not least that I could not easily define a CHP monad â€" so I discard this in favour of everything being a process; an event synchronisation and a process are both of type CHP a, and thus there is no difference at the type-level between engaging in one event and engaging in many events. Sequencing The simplest form of sequencing in CSP is the prefix operator. After a it behaves like P. CSP also has a semi-colon operator for sequencing two processes which relies on notions of termination â€" a complex issue in formal systems of computation! To perform a communication, we need a mechanism for performing an output to a channel, and a corresponding mechanism to input from the other end of the channel. This translates to a call to writeChannel and monadic sequencing in CHP: To put it differently, this performs the action of reading a value from channel c, and binds the return value to x in the right-hand side of the arrow. This should sound familiar to Haskell programmers â€" this is monadic bind! Indeed, here is the CHP rendering: As an example, here is a CSP process H that copies values from its left channel to its right channel: So for some integer channel c, c. Thus channels in CSP are a sort of syntactic sugar on top of events â€" but in CHP we deal with actual channels that can communicate values, and they are distinct from barriers. CSP is declarative, so it does not support the idea of assignment, or altering the value of a variable. Values are only introduced through the aforementioned binding of inputs, through constants, or by parameterisation of processes. Which means that slotting CSP into Haskell comes fairly naturally. We have seen that the sequencing of communications maps well to a monad the CHP monad. Parallel Composition CSP allows for parallel composition of processes. P Q is near-identical to the CHP version: Choice CSP has the notion of external choice. After the first event has occurred, the subsequent behaviour of the object is described by P if the first event was x, or by Q if the first event was y. CSP has rules for what happens if the first events are not distinct; in CHP this is considered to be a programmer error. Note that the choice is only of the first events, not of later events in P and Q; this is also the case in CHP, although it requires a certain amount of wizardry underneath to pick out the leading event of a CHP code block. As an example, Roscoe defines a counting process: It is only obliged to communicate if the environment offers both a and b. In the first case, the choice of what happens is in the hands of the environment, in the second it is in the hands of the process. That is, nondeterministic choice is a modelling construct useful for capturing and reasoning about the behaviour of processes, but it is not something you would mean to write in a program a bit like bottom in Haskell, perhaps. Primitive Processes CSP has two useful primitive processes: SKIP is the process which is always ready in a choice, and always immediately terminates successfully. So in CHP it has the behaviour of return , but with the special property that it can be used in a choice. STOP is the process which is never ready in a choice, and never terminates. In CHP it acts the same â€" it is primarily useful for not being ready in a choice and thus for dummy items in a choice ; running a process that has the sole purpose of doing nothing and not terminating is rarely useful! It is useful for formal reasoning about programs, and expressing semantics, but just as you do not need to know the lambda calculus to program Haskell, you do not need to know CSP in order to program CHP. CSP also provided the idea of traces, which feature in CHP, and were briefly described in a previous post on using traces for testing. Unfortunately you always need those brackets around the input, so the do version becomes:

COMMUNICATING SEQUENTIAL PROCESSES. THE FIRST 25 YEARS pdf

5: 25 UK species' genomes sequenced for first time

This volume, like the symposium CSP25 which gave rise to it, commemorates the semi-jubilee of Communicating Sequential Processes. 1 Tony Hoare's paper "Communicating Sequential Processes" is today wid.

6: The Engineering Design Process

Ad Peeters, Implementation of handshake components, Proceedings of the international conference on Communicating Sequential Processes: the First 25 Years, July , , London, UK Robert G. Babb, II, Data-driven implementation of data flow diagrams, Proceedings of the 6th international conference on Software engineering, p

7: Ali E. Abdallah (Author of Communicating Sequential Processes. The First 25 Years)

Ali E. Abdallah is the author of Communicating Sequential Processes. The First 25 Years (avg rating, 0 ratings, 0 reviews, published), Formal As.

8: Communicating Sequential Processes for Java (JCSP)

In computer science, communicating sequential processes (CSP) is a formal language for describing patterns of interaction in concurrent systems. It is a member of the family of mathematical theories of concurrency known as process algebras, or process calculi, based on message passing via channels.

9: An Introduction to Communicating Sequential Processes | Communicating Haskell Processes

Communicating Sequential Processes C. A. R. Hoare The theory of communicating processes is a new branch presented in the ï¬•nal or even the second year of a.

COMMUNICATING SEQUENTIAL PROCESSES. THE FIRST 25 YEARS pdf

Camps and Trails in China Manifest Dreams into Reality European textile design of the 1920s Daimios head, a masque of old Japan Mission, black list #1 American short stories minorities Intelligence, surveillance and reconnaissance Food microbiology an introduction 4th edition Apology, Crito, and Phaedo of Socrates (Dodo Press) Pt. 1. The sixties : the Bhulabhai Memorial Institute decade SKIBSAKSJESELSKAPET SOLVANG ASA Historical dictionary of the Crimean War Concepts in bioenergetics. Female Fitness on Foot Microsoft lumia 640 user guide The land of liberty. Quick and Creative Literature Response Activities (Grades K-3) Robotic exploration of the solar system part 3 The new states of Asia Practical Network Cabling (Practical) The Classical Pugilism Bare-knuckle Boxing Companion, Vol. 2 A song of achilles The countess, or, Memoirs of women of leisure Napoleon and the Invasion of Britain Church and state in Britain since 1820. Plekhanov the role of the individual in history A History of the Origin and Progress of Chivalric Freemasonry in the British Isles, Particularly Ireland The great show in Kobol-land. Synergetics of measurement, prediction, and control Infection in the critically ill Business permit application form Nhra 2015 rule book Grapes and raisins : from muscadines to Thompson seedless A study of the guided waves in a three layer guide with a nonlinear film Folk tales and fantasies Alone in the middle Rich mans relatives 2010 toyota corolla s owners manual The All-Star Game Evolution of electric batteries in response to industrial needs