Theory of concurrency

Natalia Garanina

Institute of Informatics Systems

garanina@iis.nsk.su

Introduction

Lecture 1

Parallel systems and Critical software





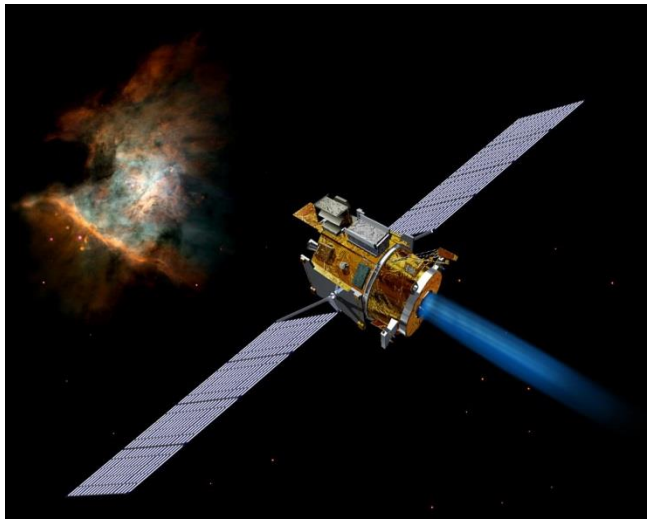

?

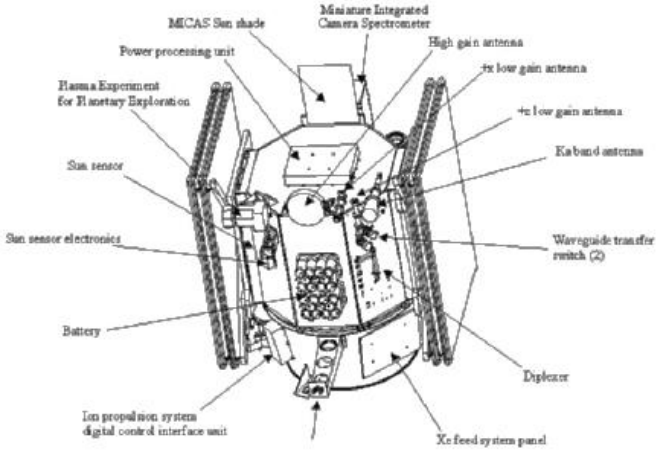
Correctness

Deep Space–1 (NASA)

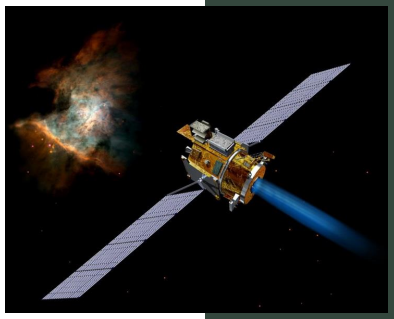
- A classical error of interleaving.
- Does it hold always $0 \leq x \leq 200$?

```
proc Inc = while true do if x < 200 then x := x + 1 fi od
proc Dec = while true do if x > 0 then x := x - 1 fi od
proc Reset = while true do if x = 200 then x := 0 fi od
```





Deep Space–1 (NASA)



- Does it hold always $0 \leq x \leq 200$?

```
proc Inc = while true do if x < 200 then x := x + 1 fi od
proc Dec = while true do if x > 0 then x := x - 1 fi od
proc Reset = while true do if x = 200 then x := 0 fi od
```

- x = 200
- Dec: 200 > 0?; Reset: 200 = 200?, x:=0; Dec: x:= x–1 = –1.

Автономное транспортное средство Alice

- Беспилотный автомобиль Alice построен Caltech
  - для DARPA Urban Challenge 2007 года.

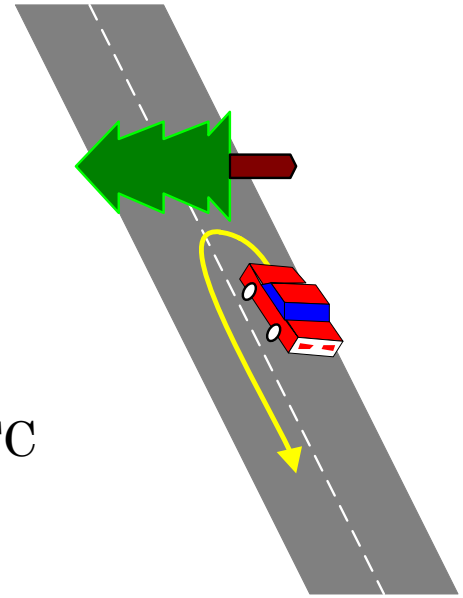
- Условия
  - Полная автономность
  - Частично известная городская среда
    - статические и динамические препятствия

Автономное транспортное средство Alice

- Модифицированный фургон Ford E350
  - механические приводы (тормоз, дроссель, рулевое управление и трансмиссия),
  - датчики (LADAR, RADAR и камеры)
  - Applanix INS (для оценки позиционирования).
- Вычислительная система
  - 25 программ и 200 потоков одновременно выполняющихся на 25 процессорах.
  - Сенсорная подсистема обеспечивает
    - представление окружающей среды
  - Управляющая подсистема управления определяет и выполняет
    - движение транспортного средства для достижения целей
      - пересечение путевых точек GPS, обход препятствий, соблюдение правил дорожного движения и т. д.

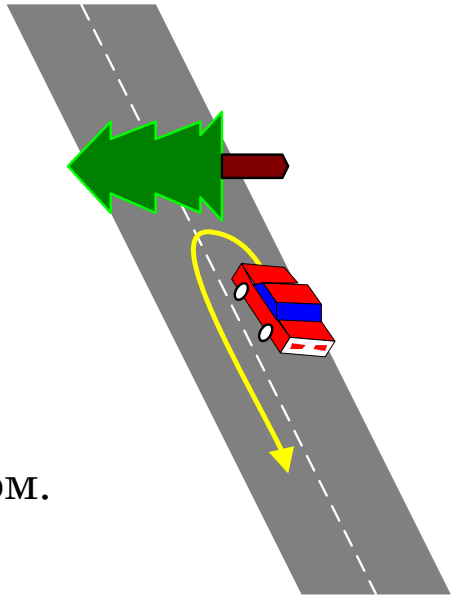
Автономное транспортное средство Alice

- Alice застряла на крутом повороте, опасно дергаясь.
- Это поведение было вызвано плохим взаимодействием между подсистемой предотвращения препятствий (ROA) и относительно медленно реагирующим планировщиком пути.
- Планировщик инкрементально генерирует
  - последовательность путевых точек
    - дорожная карта, препятствия и цели.
- ROA предназначен для быстрого замедления транспортного средства
  - приближение к препятствиям или
    - отклонение от запланированного пути слишком велико
- Контроллер низкого уровня для защиты системы рулевого управления ТС
  - ограничивает скорость рулевого управления на низких скоростях.



Автономное транспортное средство Alice

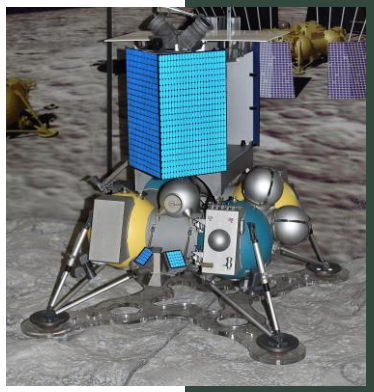
- Планировщик инкрементально *генерирует последовательность путевых точек*.
- ROA *замедляет* ТС.
- Контроллер *ограничивает скорость рулевого управления* на низких скоростях.
- Если на низкой скорости планировщик создает
  - путь с резким поворотом влево,
    - контроллер не даёт выполнить поворот достаточно точно,
      - Alice отклоняется от пути,
        - ROA активируется и замедляет ТС ещё больше.
- Этот цикл продолжается, что приводит к дерганью.
- Планировщик должен учитывать ограничения, налагаемые контроллером.



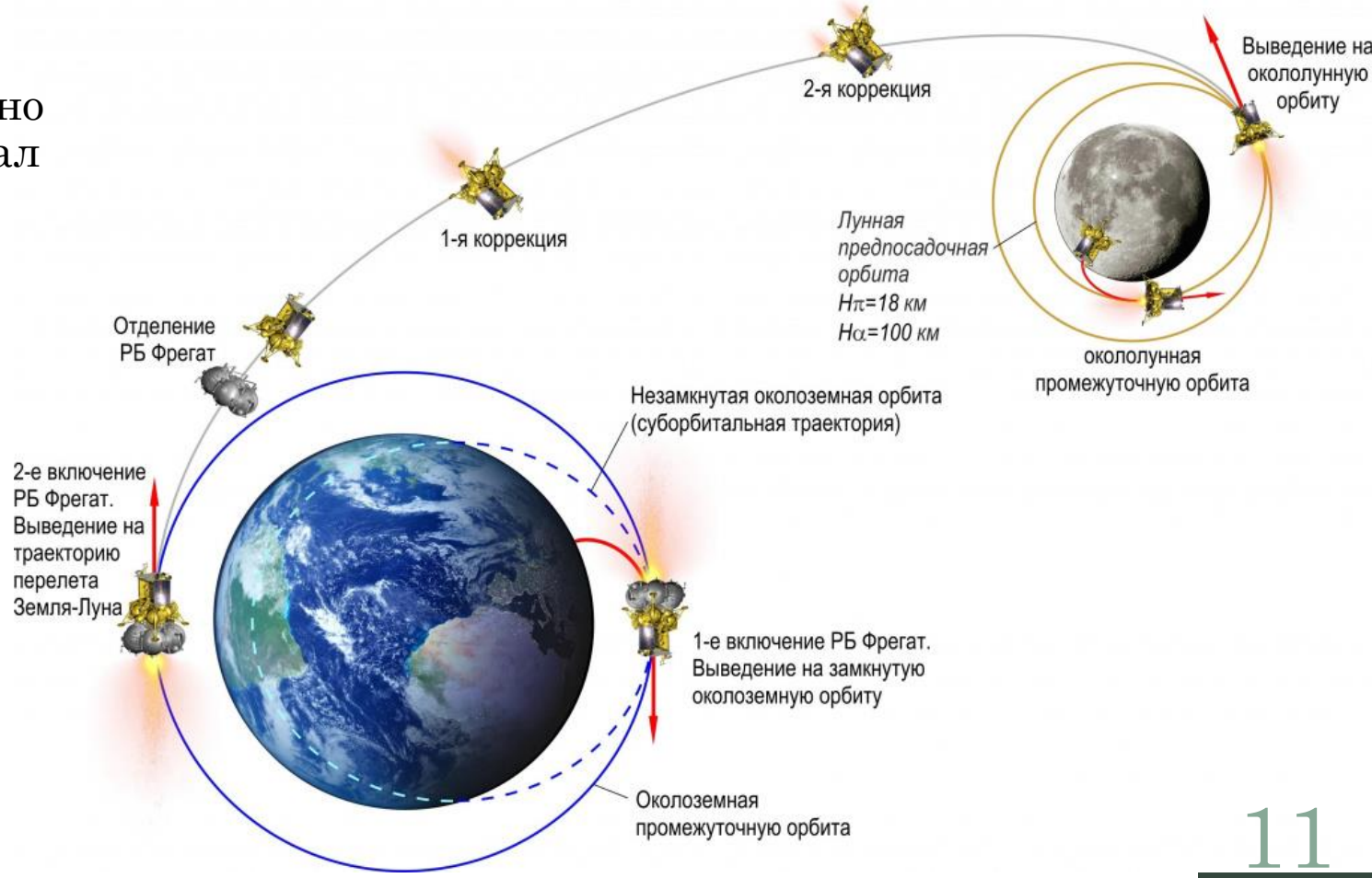
Автономное транспортное средство Alice

- Проектирование надежных встроенных систем управления наследует трудности
  - систем управления
  - распределенных (параллельных) вычислительных систем.
- Ошибка проектирования Alice
  - нежелательное поведение при определенном наборе условий
    - планировщик, контроллер и ROA взаимодействуют очень специфическим образом.
  - не обнаружена за тысячи часов интенсивных симуляций и более трехсот миль полевых испытаний.
- Формальные методы предоставляют инструменты и методы для выявления таких тонких ошибок дизайна и математически доказывают правильность проектирования,
  - автоматическое конструирование контроллеров, корректных по построению.

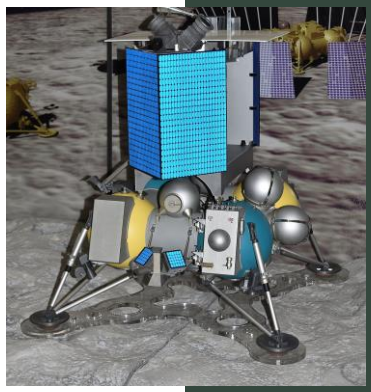
Космос



- **Луна 25, Роскосмос** (19.08.2023)
- Модуль не смог выйти на промежуточную окололунную орбиту, промахнулся, т.к. излишне ускорился, т.к
  - двигатель аппарата не отключился штатно во время последнего маневра и проработал 127 секунд вместо 84.
    - несработавший датчик скоростей.



Космос



- **Луна 25, Роскосмос** (19.08.2023)

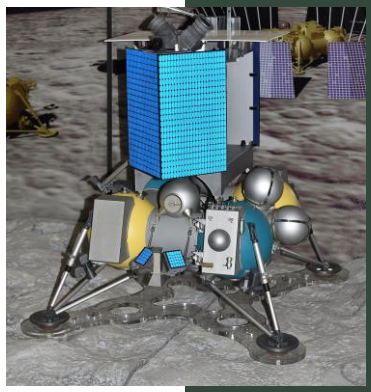
Версия Андрея Емельянова (заслуженный испытатель космической техники)

- Выключение двигателей происходит по показаниям датчиков, которые
  - определяют изменение скорости и дают команду на выключение двигателей.
- Если такого не происходит, двигатель выключается не по датчику скорости, а по часам:
  - рассчитывается гарантированное время, в течение которого двигатель может набрать определенную скорость.

Пример:

- По датчику скорости должно отработать 10 м/с, после чего произойдет отключение.
- Если выключение не происходит, то мы понимаем, что наш набор скорости 10 м/с может произойти при условно штатной работе двигателей гарантированно в пять секунд.
- Как только пять секунд проходит, поступает команда на выключение двигателей от датчика скоростей.

Космос



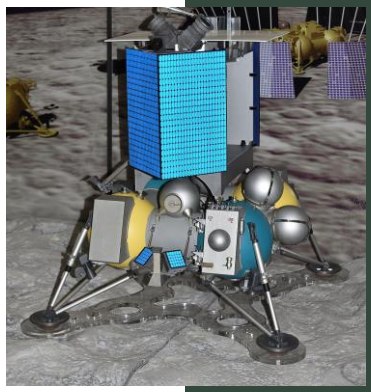
- **Луна 25, Роскосмос** (19.08.2023)

Почему не сработал датчик скорости.

- Согласно протоколу, при подлете «Луны-25» к земному спутнику *система блока измерения угловых скоростей* БИУСа-Л ждала команду от подсистемы на включение акселерометров.
- Команда была подана, но анализ ответной реакции не был запрограммирован.
  - Отсутствие информации о том, что блок измерения угловых скоростей запущен в работу.
  - «БИУС-Л», не получив команды,
    - не включился в правильный режим,
    - не передавал правильную информацию в систему, реализующую общий алгоритм управления,
      - а она не передала сигнал двигателю на своевременное отключение.

https://www.mk.ru/science/2023/09/13/poyavilas-novaya-versiya-krusheniya-luny25-programmisty-proglyadeli-elementarnuyu-oshibku.html

Космос



- **Луна 25, Роскосмос** (19.08.2023)

Емельянов отметил, что теперь потребуется проверить все системы, где устанавливаются аналогичные датчики скорости.

«Проверят партию, проверят производство этих датчиков, обязательно найдут причину, почему не сработал этот датчик, либо организационную причину, либо техническую причину.

Будут сделаны выводы: доработана документация, будет перестроено производство, может быть, какие-то административные выводы будут сделаны. …»

https://www.rbc.ru/technology_and_media/22/08/2023/64e473189a79473d2e6638cf

History

- 1962 – the theory of Petri nets.
- 1970 – three main styles of formal reasoning about computer programs.

1. *Operational semantics*. John McCarthy.
   - A computer program is modeled as an execution of an abstract machine.
   - A state of such a machine is a valuation of variables, a transition between states is an elementary program instruction.
2. *Denotational semantics*. Dana Scott and Christopher Strachey
   - Computer programs are modeled by a function transforming input into output.
3. *Axiomatic semantics*. Robert Floyd and Tony Hoare.
   - Proof methods proving programs correct.
   - Central notions are program assertions, proof triples consisting of precondition, program statement and postcondition, and invariants.

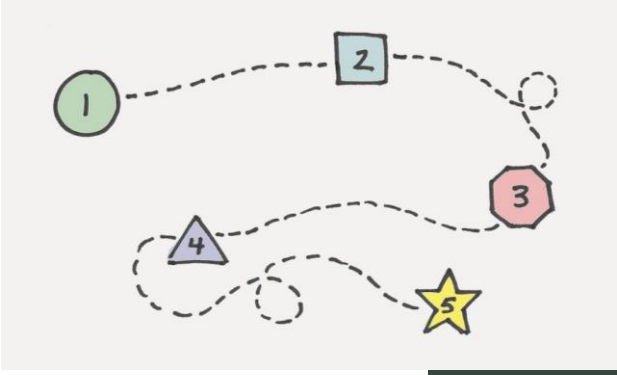
History

- How to give semantics to programs with a parallel operator?
- The methods of denotational, operational or axiomatic semantics is not appropriate:
  - a behaviour as an input/output function.
    - a program can be modeled as an automaton
      - but the notion of language equivalence does not work.
        - the interaction a process between input and output influences the outcome.
  - the notion of global variables
    - a state of a modeling automaton is given as a valuation of the program variables.
      - The independent execution of parallel processes makes it difficult or impossible to determine the values of global variables at a given moment.
        - to let each process have its own local variables, and
        - to denote exchange of information explicitly.

Introduction

- Concurrency = processes + communication.
- Communication via message passing vs shared variables.
- Different ways of specification, formalization and analysis of concurrent systems
  - Formalization
    - Process algebras, Petri nets, transition systems: automata, Kripke structures, …
  - Analysis
    - Property verification, checking equivalence

What are Processes?



- Process
  - behaviour of a system
    - the execution of a software system, the actions of a machine, the actions of a human being.
    - the total of events or actions that a system can perform,
    - the order in which they can be executed
    - timing or probabilities.
  - certain aspects of behavior
    - an abstraction or idealization of the ‘real’ behaviour.
  - an observation of behavior
    - an action is the chosen unit of observation.
      - the discrete actions:
        - occurrence is at some moment in time,
          - different actions are separated in time.
- A process is sometimes also called a discrete event system.

A Process Algebra

- An algebraic/axiomatic approach in talking about behaviour
  - the methods and techniques of universal algebra

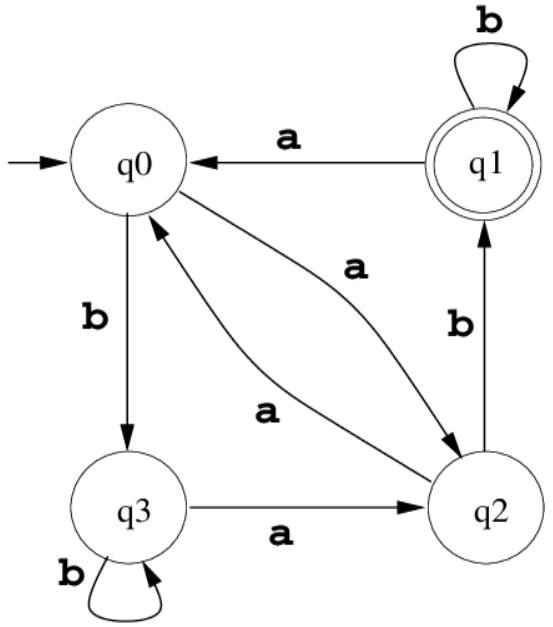
Definition 1. A group has a signature (G, *, u, −1) with the laws or axioms

- a * (b * c) = (a * b) * c
- u * a = a = a * u
- a * a−1 = a−1 * a = u

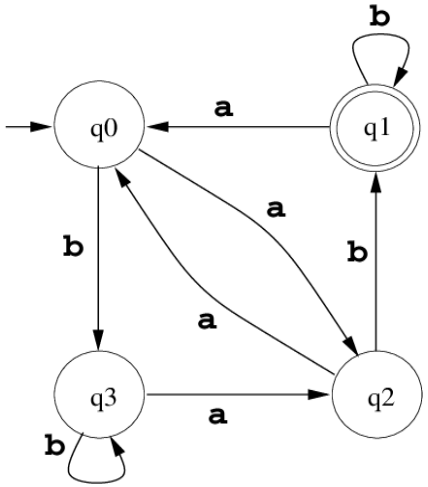
- A group is
  - any mathematical structure with operators satisfying the group axioms.
  - any model of the equational theory of groups.
- A process algebra is any
  - mathematical structure satisfying the axioms given for the basic operators.
- A process is an element of a process algebra.
- By using the axioms, we can perform calculations with processes.

A Process Algebra

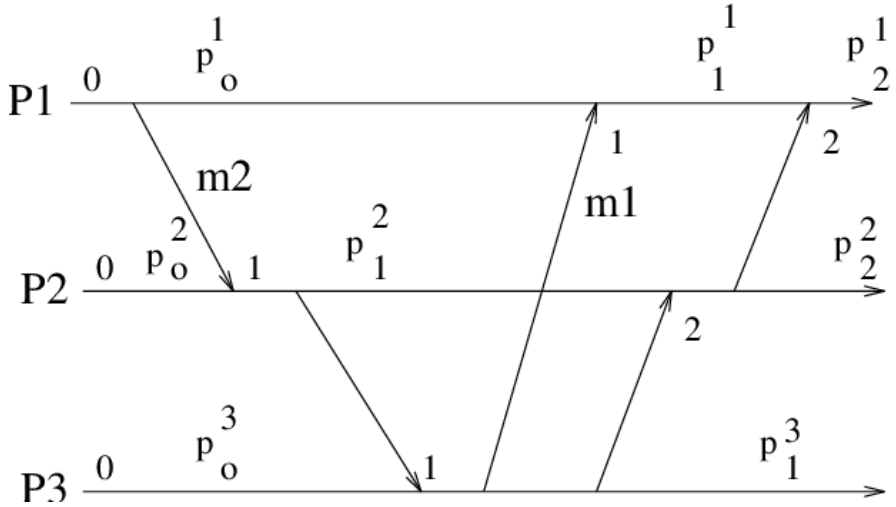
- The simplest model of behaviour is
  - input/output function.
- The finite state automata theory.
  - a process is modeled as an automaton.
    - has a number of states and
    - a number of transitions between states.
      - the execution of an elementary action.
    - initial states
    - final states.
  - A behaviour is a run, i.e. a path from initial state to final state.
  - A notion of automata equivalence.
    - language equivalence.
    - the algebra of regular expressions for equational reasoning about automata.



A Process Algebra



- Missing is the notion of interaction
  - for parallel or distributed systems, or reactive systems.
- Concurrency theory is the theory of interacting, parallel and/or distributed systems.
  - process algebra is an approach to concurrency theory.
    - the study of the behaviour of parallel or distributed systems by algebraic means.
      - parallel composition.
      - alternative composition (choice)
      - sequential composition (sequencing).
    - equational reasoning.
      - verification
        - a system satisfies a certain property.



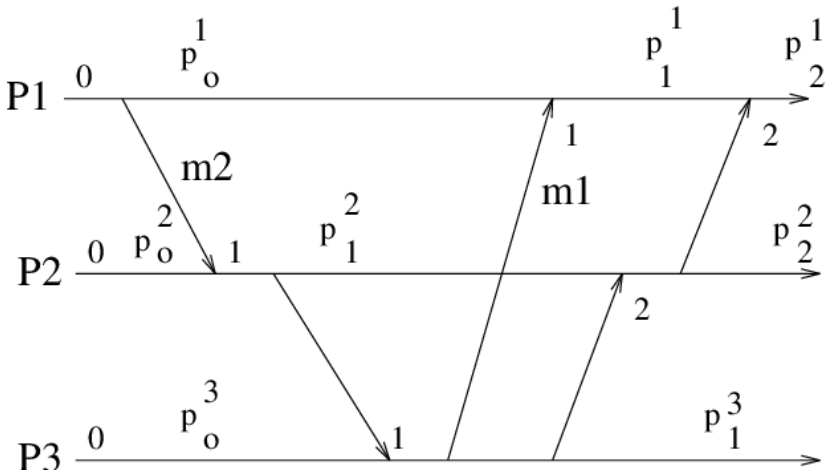
A Process Algebra

- Basic laws of process algebra
  - A given set of atomic actions
- The basic operators to compose the actions into more complicated processes.
  - + – alternative composition
  - ; – sequential composition
  - ‖ – parallel composition.

1. $x + y = y + x$ (commutativity of alternative composition)
2. $x + (y + z) = (x + y) + z$ (associativity of alternative composition)
3. $x + x = x$ (idempotency of alternative composition)
4. $(x + y); z = x; z + y; z$ (right distributivity of + over ;)
5. $(x; y); z = x; (y; z)$ (associativity of sequential composition)
6. $x \| y = y \| x$ (commutativity of parallel composition)
7. $(x \| y) \| z = x \| (y \| z)$ (associativity of parallel composition)

A Process Algebra

- Connection parallel composition to the other operators.
  - an expansion theorem
    - law for expressing parallel composition in terms of the other operators
    - dynamic law
      - involves action execution explicitly
- Process algebras
  - with an expansion theorem
    - interleaving process algebras
  - without
    - partial order or true concurrency.

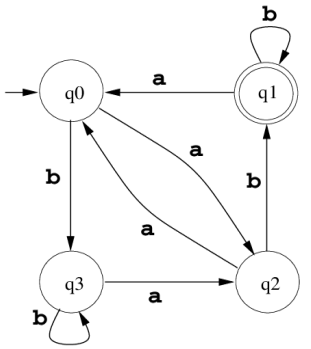


A Process Algebra

1. $x + y = y + x$
2. $x + (y + z) = (x + y) + z$
3. $x + x = x$
4. $(x + y); z = x; z + y; z$
5. $(x; y); z = x; (y; z)$
6. $x \parallel y = y \parallel x$
7. $(x \parallel y) \parallel z = x \parallel (y \parallel z)$

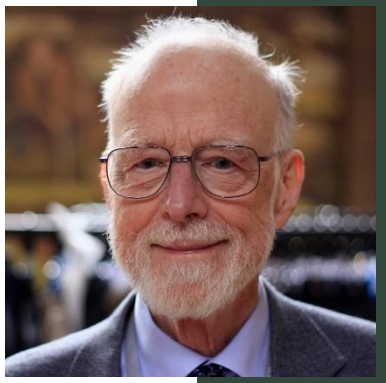
- A process algebra is
  - any mathematical structure with three binary operations satisfying the 7 laws.

- Transition systems are
  - these structures formulated in terms of automata.
    - states and transitions between them, initial states and final states.
  - equivalence as bisimulation.



- Process algebra is
  - the study of equational theories with their models,

- Process theory is also
  - the study of transition systems and related structures, ways to define them and equivalences on them.



Communicating Sequential Processes

- CSP is a formal language for describing patterns of interaction in concurrent systems.
- 1978 by **Tony Hoare**.
- The description of systems in terms of processes
  - operate independently
  - interact with each other through message-passing communication.
- Process algebraic operators describe
  - the relationships between different processes, and
  - the way each process communicates with its environment
- $(a \rightarrow a \rightarrow STOP) \parallel (a \rightarrow b \rightarrow P)$
- Influence:
  - occam, Limbo, JCSP, Go, PyCSP, Crystal, and Clojure's core.async.
- Practical application in industry
  - as a tool for specifying and verifying the concurrent aspects
  - …T9000 Transputer, a secure ecommerce system.

Calculus of Communicating Systems

- CCS is a process calculus introduced by Robin Milner around 1980.
- The formal language includes primitives for describing parallel composition, choice between actions and scope restriction.
  - indivisible communications between exactly two participants.
- CCS is useful for evaluating the qualitative correctness of properties of a system such as deadlock or livelock.
- The expressions of the language are interpreted as a labelled transition system.
- Bisimilarity is used as a semantic equivalence between these models.
- $a.b.P1 + c.d.P2$
- Influence:
  - The π-calculus, Language Of Temporal Ordering Specification (LOTOS), Java Orchestration Language Interpreter Engine (Jolie)

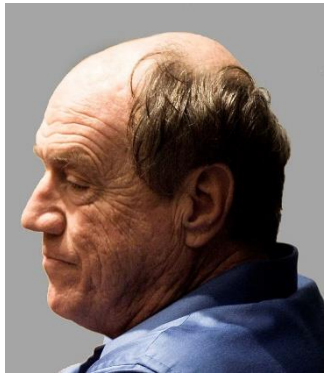
π–calculus

- 1992 by Robin Milner et al.
- Mobile processes:
  - channel names to be communicated along the channels themselves
    - network configuration may change during the computation.
- *!(c(x).P | Q)*
- Extensions
  - Control structures: recursion, loops and sequential composition.
  - Distribution or public-key cryptography.
- Influence:
  - spi–calculus and applied π
    - reasoning about cryptographic protocols
  - Business Process Modeling Language (BPML)
  - JoCaml (based on the Join–calculus)
  - RhoLang

Before the Actor Model

- CSP and CCS are process algebras
  - Interprocess communication via a static structure of channels between processes.
  - Mobility is not easily representable
    - new objects can be created at run time and/or moved in different locations.
- The π–calculus is a calculus for mobile processes
  - for processes with a dynamically changing linkage structure.
  - taking into account a synchronous handshake communication mechanism.
  - the asynchronous communication mechanism.
  - do not provide the concept of an object as a first class entity.
- The actor model directly deals with
  - object identity, asynchronous message passing, an implicit receive mechanism, and support for object creation;
  - an actor has the same structural and interaction properties as an object.

Actor Model



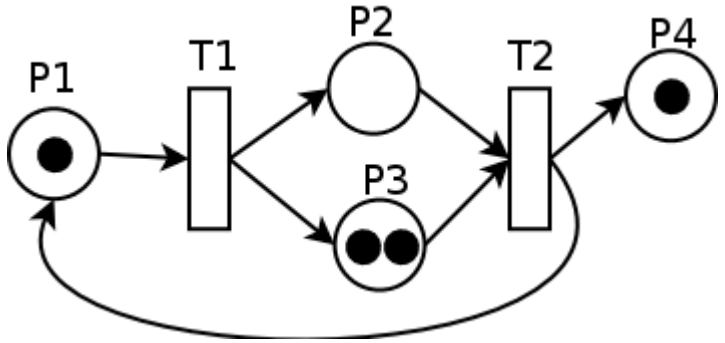
- 1977 by Carl Hewitt.
- Actors are self-contained agents with
  - a state and a behaviour which is
    - a function of incoming communications.
  - has a unique name (mail address) determined at the time of its creation.
    - supporting object identity.
      - Not in CCS or asynchronous π–calculus
        - message dispatching is performed by means of channels.
        - the association address–process is not unique
          - a process may have several ports (channels) from which it receives messages and the same channel can be accessed by different processes.

Actor Model

- Communication by asynchronous and reliable message passing
  - whenever a message is sent it must eventually be received by the target actor.
- An implicit receive mechanism.
  - objects can be seen as passive entities which react to messages or to method invocation.
- Three basic primitives for actors
  - *create*
    - to create new actors;
  - *send*
    - to send messages to other actors; and
  - *become*
    - to change the behaviour of an actor
  - asynchronous and non–blocking.

Petri Net

- 1939 by Carl Adam Petri
  - at the age of 13—for the purpose of describing chemical processes.
- A Petri net is a directed bipartite graph in which
  - the nodes represent transitions and places.
  - The directed arcs describe which places are pre– and/or postconditions for which transitions.
- Petri nets offer a graphical notation for stepwise processes that include
  - choice, iteration, and concurrent execution.



The Course

- We will study
  - Model checking
  - Verification method
  - Model representation
  - Property representation
  - Verification with SPIN
- Communicating Sequential Processes (CSP) by Tony Hoar
  - Classical syntax manipulation
- Classical problems and features of concurrency
  - Abstractions: atomicity, locks, semaphores, monitors, threads, rendezvous…
  - Classical problems: mutual exclusion, dining philosophers, sleeping barber, termination detection….

The Course

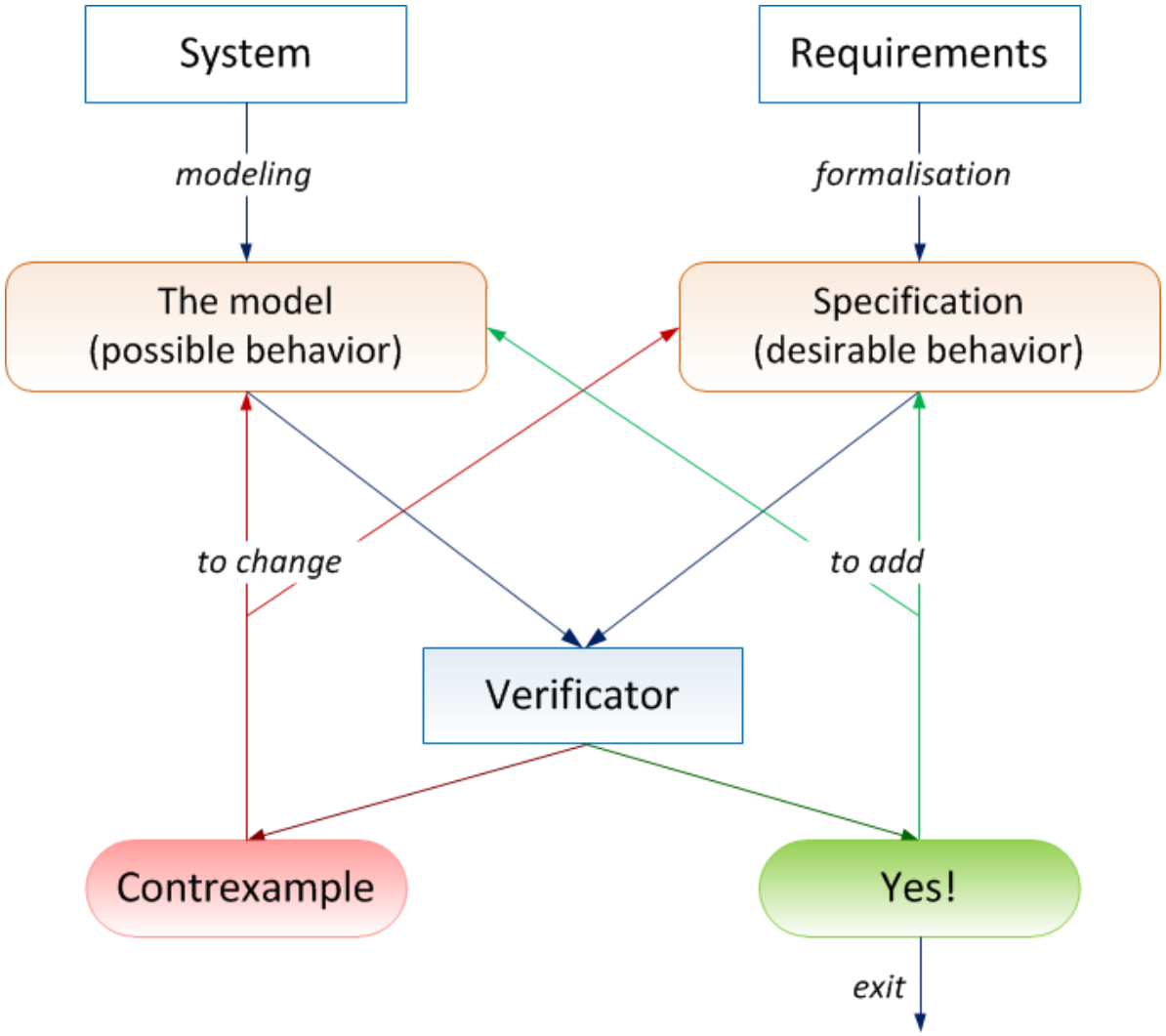
- Outline
  1. Model checking
     1. Model representation – Kripke structures
     2. Property representation – temporal logic LTL
     3. The verifier SPIN
  2. Communicating Sequential Processes
     1. Everything

Model checking

Model Checking

- An *exhaustive search* in the set of all states of the system model:
  - for each state of the system, it is checked whether it satisfies the required property;
  - terminates due to the finiteness of the model.
- *Reachability analysis*.
  - *Blocking*
    - Can the system get into a state in which the calculation cannot continue?
      - identify all reachable states and
        - check if there is a block state among them.
  - *Invariant properties*
    - The proof of properties are satisfied throughout the calculation.
      - no blocking, progress, etc.
- *Insufficiently*, for example, for communication protocols,
  - Property: if a message is sent, then it will certainly be received.

Model Checking Scheme



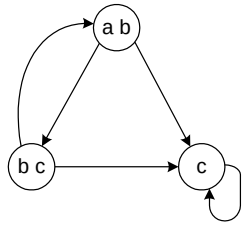
Model Checking Approachs

- The difference in the description of the desirable behavior.

1. Logical or heterogeneous approach.
2. Behavioral or homogeneous approach.

- *Logical or heterogeneous approach*.
  - The specification of properties (requirements) with
    - some logic (temporal or modal).
  - System model
    - The finite Kripke structure (automata, etc.)
      - states – values of variables and control positions,
      - transitions – change of system states.
  - A system is considered correct with respect to requirements
    - if in a given set of initial states these requirements are satisfied.

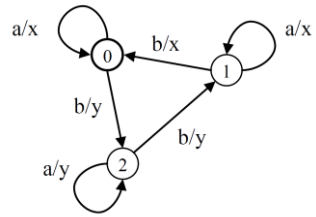
$$\neg a \, \mathbf{W} \, (\, b \wedge \neg a \wedge \mathbf{F} a \,)$$



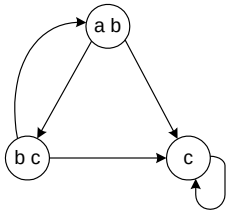
Model Checking Approachs

- The difference in the description of the desirable behavior.

1. Logical or heterogeneous approach.
2. Behavioral or homogeneous approach.



- *Behavioral or homogeneous approach*.
  - The desired and possible behavior are specified in the same notation
    - finite state machines, Petri nets, process algebras etc.
  - The correctness is due to equivalence relations (or preorders).
    - Simulation, bisimulation, step by step, input-output, language inclusion, etc.



- A system is considered correct
  - if the desired and possible behaviors are equivalent (or ordered) with respect to the equivalence (or preorder).

Model Checking Advantages

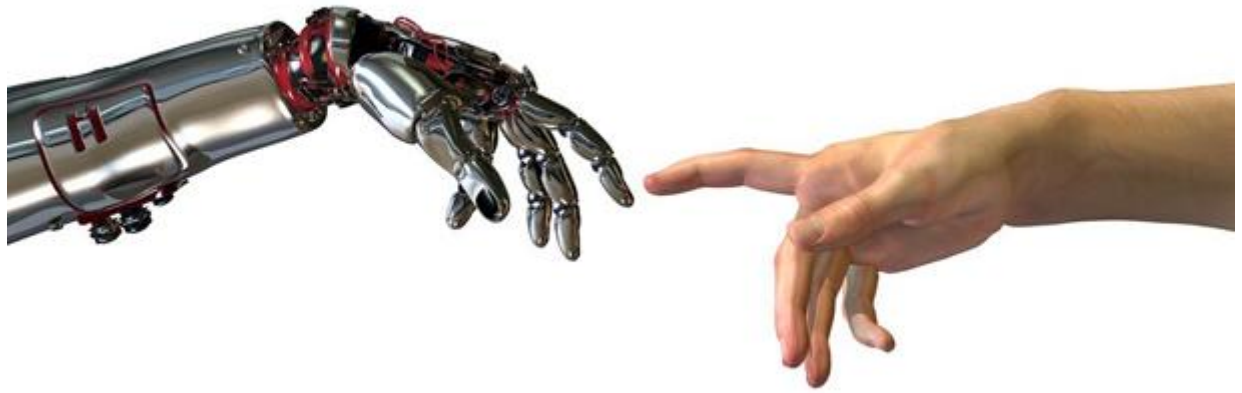
- This is a general approach with applications for verification of
  - hardware, programs, communication protocols, multi-agent systems, embedded systems, etc.
- Completeness of verification.
- The approach supports partial verification.
  - verification of the most important properties, ignoring the verification of less important, but computationally expensive requirements.
- Model checking tools do not require a high degree of user interaction.
- A reliable mathematical foundation:
  - modeling, semantics, logic and theory of automata, data structures, graph algorithms, etc.

Model checking disadvantages

- Not for data processing applications
  - infinite sets of states.
- Verification of the system model, not the system itself.
  - There is no guarantee that the final implementation has the same properties.
- The formalization of the “correct” model of the system and the “correct” formulation of requirements requires appropriate qualifications and efforts.
- Insufficient verification of verifiers.
- Limited generalization of verification results
  - The verified protocol for three processes is not necessarily correct for four.
  - There are generalization methods.
- The main problem of model checking is *a state explosion*.
  - concurrent systems with a large number of states of individual components.

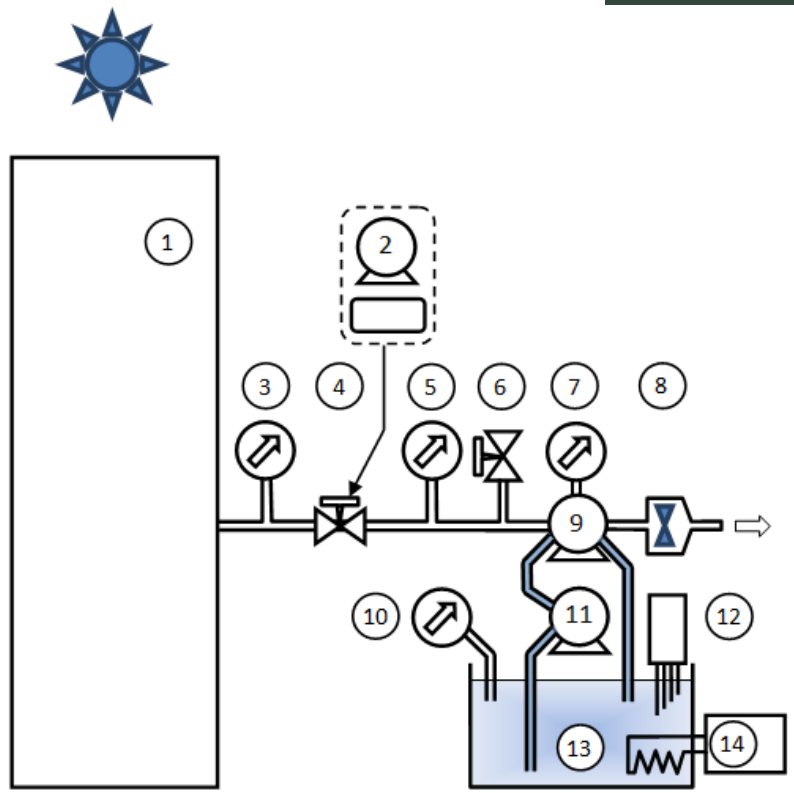
Formal Verification

- Absolutely guaranteed system correctness is not possible.
- *Just formulated is verified*.
- But model checking significantly *increases confidence* in systems.

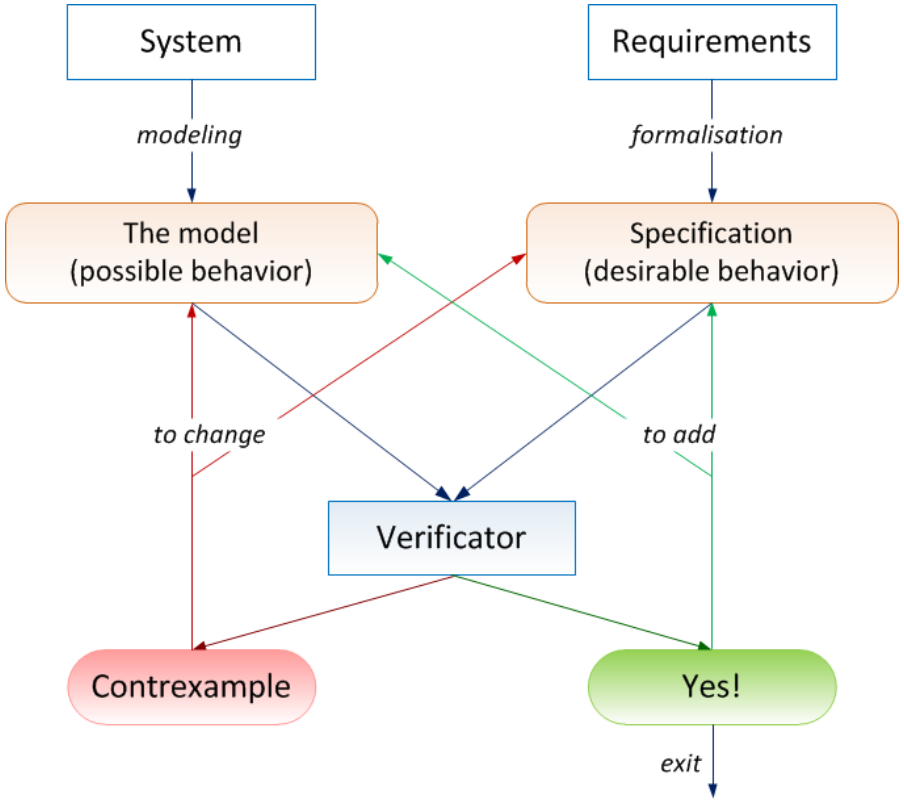


Reactive systems

- Continuous interaction with the environment
  - take the values of inputs from their environment and respond to them.
- Operating Systems
- Communication protocols
  - cryptographic, communications, etc.
- Control systems
  - airplanes, automobiles, plants and ships, technological processes, etc.



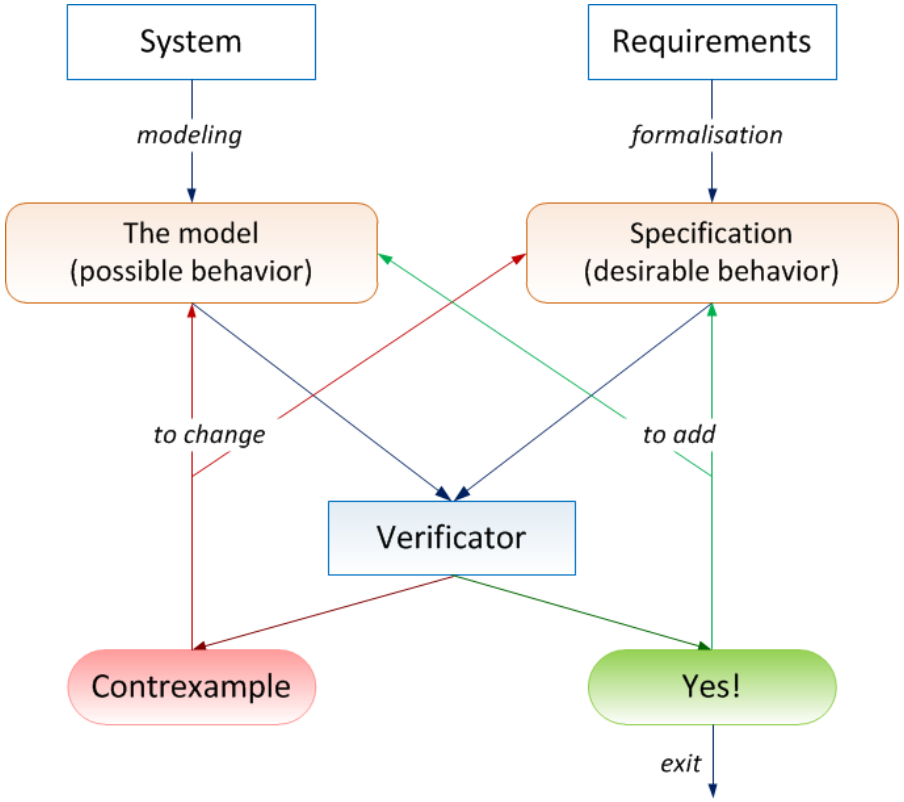
Model checking process



- Modeling
  - Translating a description of
    - a software system to the formal description
      - suitable for automatic verification.
  - Compilation, translation.
  - Abstraction.
- Specification
  - Formulating software properties systems for verification
    - Logic: temporal, dynamic, epistemic, deontic and many others.
    - Completeness of the specification.
      - Safety: nothing bad will ever happen.
      - Liveliness: something good is definitely happen.
- Verification

Model checking process

- Modeling
- Specification
- Verification
  - Checking consistency of the model and its specification.
  - Fully automatic.
  - Results Analysis
    - Counterexample
    - False counterexample
  - Large size model.
    - Abstraction
    - Model refinement



Summary

- Parallel systems and Critical software
  - Correctness
  - Formal verification (math)
    - Quality assurance
- Concurrency = processes + communication
- Formalization
  - Process algebras
  - Transition systems: Kripke structures, automata, Petri nets, hyperprocesses…
- Analysis
  - Property verification, checking equivalence

Summary

CSP

- $a \rightarrow P \mid (a \rightarrow P) \sqcap (b \rightarrow Q) \mid (a \rightarrow P) \Box (b \rightarrow Q) \mid P \,|||\, Q \mid P \,||\, \{a\} \,||\, Q \mid (a \rightarrow P) \setminus \{a\}$

SSC

- $P ::= \emptyset \mid a.P_1 \mid A \mid P_1 + P_2 \mid P_1 | P_2 \mid P_1[b/a] \mid P_1 \setminus a$

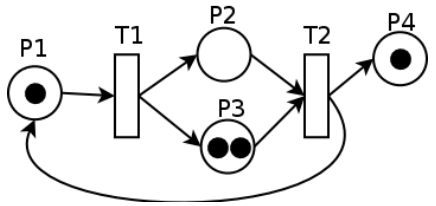
π–calculus

- $P \mid Q - c(x).P - c\langle y\rangle.P - !P - (\nu x)P - 0$

Actor model

- $P ::= become(C, e).P \mid send(e_1, e_2).P \mid create(b, C, e).P \mid e_1 : P_1 + \ldots + e_n : P_n \mid \emptyset$

Petri net



Summary

- Model Checking (formal verification)
  - Exhaustive search in parallel system states
    - States, transitions
      - Kripke structure
        - Formal model for programs and systems in general
          - Translation
  - Checks if a specification is satisfiable
    - Specification logics
      - Temporal logics
        - Something happens in time
      - Other logics