Minutes of PARDI's Kick-off

30th jan 2017 - 1st feb 2017

The purpose of this report is to keep track of the questions and discussions which occurred during the presentations. It does not summarize these presentations (refer to the slides).

Participants

Souheib Baarir, Sylvain Conchon, Marie Duflot-Kremer, Mamoun Filali, Lom Hillah, Aurélie Hurault, Philippe Mauran, Stephan Merz, Meriem Ouederni, Pascal Poizat, Philippe Quéinnec, Jean-Baptiste Raclet (partially), Xavier Thirioux (partially), Fatiha Zaïdi.

Presentations

- PARDI and the ANR: general information on the ANR (Philippe Quéinnec)
- Parameters and Distributed Systems
  - Introduction to Distributed Systems (Philippe Quéinnec)
  - Parameters in Distributed Systems (Philippe Quéinnec)
  - Communication Models (Aurélie Hurault)
- Mechanized Proofs of Parameterized Systems
  - Introduction to TLA+ and its Tooling (Stephan Merz)
  - Encoding TLA+ into Many-Sorted First-Order Logic (Stephan Merz)
- Automatic Verification of Parameterized Systems
  - Current Status of Cubicle (Sylvain Conchon)
  - Future of Cubicle (Sylvain Conchon)
- Parameterized Workflow Systems
  - Introduction to Workflows (Pascal Poizat)
  - Choreography Development Issues (Pascal Poizat)

Discussions

Parameters and Distributed Systems

- *Faults.* Faults have been discussed. What are they? How are they modeled? Equivalence between communication failure (e.g. message loss) and process failure (e.g. omission).
- *Synchronous/asynchronous models.* Synchronous and asynchronous models have been discussed. Especially, the overloaded word "synchronous" has several meanings: synchronous communication (e.g. in CCS or rendez-vous), synchronous execution model (e.g. Lustre), synchronous by round (e.g. synchronous distributed systems). In distributed systems, synchronous system/algorithm is often an alias for "by round". In an asynchronous

system, messages have unbounded transmission delay; in a synchronous system, the transmission delay is bounded and is known. In this world, algorithms are generally described by rounds, where all processes are in the same round, and a message sent in a round is received in this same round. The execution of a process in a round is: send messages, wait for messages from the current round to arrive, local action. Alternatively and equivalently, a round can be defined as: wait for messages from previous round, local action, send messages for the next round.
Rounds are weak synchronisation points, as processes are not necessarily stepping from one round to the next all at the same exact time. However, messages do not cross round barriers. This model is important in distributed algorithms as it makes it possible to reach consensus in presence of failure (whereas consensus is impossible in an asynchronous system with fault - see famous result by Fischer, Lynch & Paterson, 1985)

- *System/algorithm specifications.* Often, the high-level specification of a distributed algorithm is imprecise ("it does an election"), especially with regard to special cases or faults.
- *Specifications in TLA+.* TLA+ does not have an explicit notion of process, contrary to PlusCal (translated into TLA+) or Cubicle. However it is good practice to make the processes visible, for instance with variables indexed by 1..N. Making this good practice more explicit (e.g. having a standard word such as Proc) seems required to easily translate TLA to Cubicle.
- *Specifications in TLA+.* TLA+ is untyped. Again, it is good practice to include a type invariant which is verified on the specification. Having annotations (comments) to help translation seems reasonable.
- *Specifications in TLA+.* The transition relation of TLA+ is just a transition predicate, arbitrarily written using unprimed variables (denoting the values of the variables in the current state) and primed variables (denoting the values of the variables in the next state). However, the usage is to use a *Next* predicate built with a disjunction of actions. This naturally translates to Cubicle transitions.
- *Why TLA+.* Proponents of TLA+ (Philippe, Stephan) put forward its expressiveness and its ease of reading, being a purely mathematical and logical language. The current tools (TLC for finite model checking and TLAPS for assisted proof) are already adequate.
- *Why PlusCal.* It provides a standard translation from a finite concurrent shared memory model to TLA+.
- *Objectives.* Describe/specificy abstractions and models of representative problems with TLA+, Cubicle and a workflow language (this is the deliverable D1.1). This is necessary to identify the relevant schemas and data structures.

Mechanized Proofs of Parameterized Systems

- *Running example.* Dijkstra termination algorithm (EWD840)

- *Replacement of TLC by Cubicle.* The first step is to identify the fragment of TLA+ easily translatable to Cubicle. A second step is to enhance Cubicle with essential structures for TLA+ (e.g. sets).
- *Other useful enhancements.* Network topology, for instance next and previous to build a ring.
- *Use of Cubicle as a backend.* Cubicle can expose tentative invariants which could be used in TLAPS to build a machine-assisted proof.
- *Certification.* TLAPS depends on lots of internal and external tools (axiomatisation of TLA+ logic and SMT for instance). To get enough confidence, a proof built with TLAPS can be checked with Isabelle afterwards. Currently, SMT solvers do not expose enough information to certify their answers. The same question will be asked of Cubicle: if it states that no unsafe states are reachable, can it expose enough information so that this statement can be verified?
- *Assistance.* The user often knows (or thinks he knows) the number of processes which should be enough to consider when doing a proof (if the system is correct for any number of processes). For instance, with a mutual exclusion algorithm, it may be sufficient to consider two conflicting processes and another non-conflicting one. This last one abstracts any number of non-conflicting processes. Cubicle is able to help in this situation. By construction, it starts with one process and adds new ones when necessary. It could signal if ever it needs to go above the expected number provided by the user.

Automatic Verification of Parameterized Systems

- *Backward reachability.* TLC and Cubicle work in the opposite direction: TLC starts with the initial states and explores new states by computing the transition relation (forward); Cubicle starts with the unsafe states and computes a backward relation (preimage) to check if it can reach an initial state.
- *Invariant injection.* Cubicle allows for invariants to be specified, in addition to the unsafe states. These invariants can be taken as true or they can be checked on the fly. They help in reducing the explored space.
- *Objective.* An objective is to reach the same expressiveness as DVF (Deductive Verification Framework), a framework developed by Intel especially for the verification of shared memory algorithms with weak memory models. The language of DVF is also close to Murphi, and includes sets, records, tuples, integer operations, parameterized type... An important difference is that, in DVF, invariants must be given, whereas Cubicle is able to infer some by looking at finite instances.
- *Data structure.* By experience, trying to encode files and buffers in Cubicle with arrays and variables is bound to fail. Theses data structures should be native.

- *BRAB option.* Cubicle's brab option provides backward reachability with approximations and backtrack helped with a finite model of size n. Finite instances are built by forward analysis (from the initial states), and backward reachability over-approximates. The potential invariants discovered by forward analysis is used to control the over-approximation. Experimentally, this helps a lot and allows to study more complex systems.
- *Documentation.* Cubicle documentation is incomplete. Actually, Cubicle is able to do far more things than documented!
- *Modules.* TLA+ has modules to help structure and reuse a specification. Cubicle could benefit from them. However this does not have a high priority as there is little reuse in actual TLA+ specifications except for the basic data structures (finite sets, bags, sequences).
- *Multiple parameters.* Cubicle currently only handles one parameter (the processes). Adding more parameters is expected and seems reasonably easy to do (at least for two parameters). It is also useful that the parameters may be linked a constraint: typically, the fraction of processes that may fail.
- *Weak variables.* Weak variables are variables with weaker semantics. Events corresponding to start/end of memory access are made explicit and TSO (Total Store Order) is axiomatized in Alt-Ergo. Other memory models can be used as long as they are axiomatized based on these events. An important question is the usability of this approach for communication models in point-to-point message-oriented communication. In these communication models, events correspond to sending and receiving a message, and the order of delivery is defined by predicates on the ordering of the events. Both approaches seem close enough.

Parameterized Workflows

- A workflow specifies a procedure/process using a composition of activities/tasks. This specification can be executable and workflow engines are a key element. A workflow can be described as a choreography (centered on interactions from a global point of view) or a collaboration (processes are identified and interactions between these processes are explicitly specified). To be realized, a choreography is projected on processes. The realizability of a choreography is a non trivial question.
- *Objective.* The objective of the PARDI project is not to formalize BPMN or any other existing workflow DSL. The goal is to identify a core, sufficient to describe workflows extended to specify parameters, and expressive enough to specify significant examples. Actually, most workflow languages largely overlap with regard to the concepts.
- *Parameters.* Parameters are hidden in several constructs: data-related constructs (used collections), task markers. The parameters are not explicit and are not linked. For instance, multiple occurrences of a parallel operator (e.g. loops, parallel operator "triple bar") may refer to the same degree of

parallelism or not. Human interpretation of the workflow is required to discover these hidden links.

- *Current state of verification.* Currently, the majority of verification is done by translation to Petri nets. Parameters such as number of instances are often defined as 1. In any case, they must be fixed by the verification engineer.
- *Infinity.* An observation is made that workflows are often human processes. As such, they have terminal states (a goal), and rarely exhibit infinite executions (except for a cycling behavior) or infinite states (except for unbound number of instances, which is sometimes used in specifications but never used for verification).
- *Messages.* Interactions between processes can be specified by databases or by messages. Nevertheless, properties of the communication model are generally unclear (e.g. ordering? message loss?)
- *Workflow/TLA/Cubicle.* It seems reasonable to translate high level workflows into TLA+ or Cubicle input language. However, whereas describing distributed algorithms with TLA+ or Cubicle is natural, such algorithms seem unclear as workflow specifications. Thus, it is unclear if the PARDI project should seek uniformity of examples (all specifications as a workflow, as a TLA+ specification and as a Cubicle specification) or should consider two classes of systems (distributed algorithms on the one hand, business processes on the other hand).
- *Example.* The "university" (example by Chevrou-Hurault-Quéinnec) is used as an inspiration.

Deliverables

- D0.1 Minutes of the kick-off meeting – **done**
- D0.2 Web site http://pardi.enseeiht.fr – **done**
- D0.3 Consortium agreement: unnecessary (only academic partners, open source code) – **done**

Incoming Actions

- Describe/specificy abstractions and models of representative problems with TLA+, Cubicle and a workflow language (deliverable D1.1)
- Considered parameters (deliverable D1.2)