A framework for automated verification of workflow using timed temporal logic in a distributed memory environment

Md Zahidul Islam and Wendy MacCaull

Department of Mathematics, Statistics and Computer Science, St. Francis Xavier University, Antigonish, Nova Scotia

Abstract

Workflow management systems (WfMS) provide convenient ways of visualization, analysis and automation of work processes. We wish to improve the quality of workflow systems by applying formal verification techniques. We present an automatic translator for workflow models designed as Petri net models to a state space, and we develop a tableau based model checking algorithm to verify properties of that model. Tableau based theorem provers are both computationally and resource intensive. Introducing a timed and temporal logic increases the complexity. To cope with this problem we discuss a mechanism to distribute the verification procedure among multiple processors.

Background

- **Workflow**

  A workflow can be depicted as a sequence of tasks. It gives a virtual representation of an actual work.

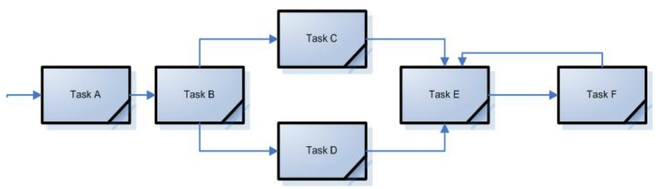

Figure: A simple workflow

- **Workflow Management Systems (WfMS)**

  WfMS provide convenient ways of visualization, analysis and automation of business processes to identify and correct potential errors in a system.

  - Properly verified workflows ensure better systems.
  - Safety critical systems like health care must ensure an error free workflow execution to ensure the safety of the patients.
  - Although time constraints plays important role in workflow execution, present WfMS cannot model time.

- **Petri Net**

  Having a strong mathematical foundation, Petri nets are a popular tool to model workflow systems in the industry. A Petri net consists of the following:

  1. A finite set of places
  2. A finite set of transitions
  3. A finite set of directed arcs. An arc can only connect a place and a transition.
  4. A finite number of tokens (representing resources) in places.

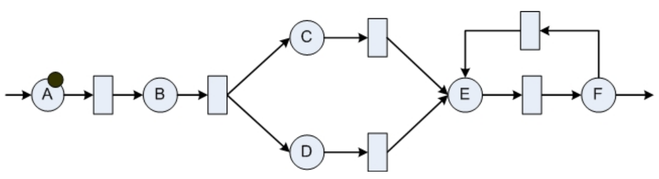

Figure: A simple Perti net workflow

A transition can only be enabled when all the input places have tokens. An enabled transition may fire (if there is no other contraints) by consuming a token from each of its input places and produces a token in each output place.

Extended formalisms of Petri nets support modelling time. However Petri net tools provide an implicit representation of the system and have limited verification capability.

Formal Verification

The process of showing that a formal design specification (model) satisfies its formal requirement specifications (properties).

Formal verification uses mathematical techniques to ensure the correctness of a system design. Various formal verification approaches include:

- **Interactive theorem proving:** It is the most powerful verification method. But it needs logicians to guide the proof procedure.
- **Automated theorem proving:** It does not require human interactions. The implementation of ATP is very complex. It provides better assurance than model checking.
- **Model checking:** In principle it is automatic. However, the application is limited to finite state transition systems because of the state explosion problem.

High Performance Computing

- Divides computation tasks into smaller pieces and distributes them to multiple processors to process concurrently.
- Reduces time to completion.
- Increases available memory.

What We Need

- An explicit model of the system showing all possible states.
- A system specification language to express timed properties.
- A strong proof procedure to verify properties of the system.

Timed temporal logics permit the explicit reference to time in order to specify quantitative properties, needed by safety critical systems. Our property specification language is timed CTL logic, which in Backus-Naur form is:
$\phi ::= \top \mid \bot \mid p \mid \neg\, \phi \mid \phi \vee \psi \mid \phi \wedge \psi \mid \mathsf{AX}\ \phi \mid \mathsf{AG}_n\ \phi \mid \mathsf{AF}_n\ \phi \mid \mathsf{A}_n(\phi\ \mathsf{U}\ \psi) \mid \mathsf{EX}\ \phi \mid \mathsf{EG}_n\ \phi \mid \mathsf{EF}_n\ \phi \mid \mathsf{E}_n(\phi\ \mathsf{U}\ \psi)$
where p is an atomic formula, n is an integer that specifies an upper bound on the time, AG, AF, EG, EF, AX, EX are the CTL operators.

Problems

- Manual translation of a large Petri net model to a state space is very time consuming.
- Timed temporal logic increases the computational complexity.
- Model checking suffers from the state explosion problem.
- Fully automatic theorem proving is quite tedious and impractical for large workflow systems.
- Even a simple workflow can have many possible execution paths and may results in a very large state space, which is hard to manipulate in a reasonable amount of time by a single processor.

Proposed Framework

Our framework provides an efficient combination of model checking and theorem proving to achieve the automatic style of model checking as well as the logical foundation of theorem proving. It also applies distributed computing to manage the state space explosion problem and to make the verification process more efficient.

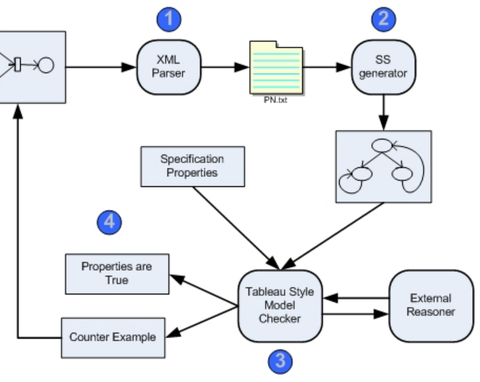

Figure: Proposed Framework

Verification Process

- **Step 1** The XML parser generates a simple text file containing the places, transitions and arcs extracted from the XML file generated by the CPNTool.
- **Step 2** The state space generator program takes the text file and generates the state space by executing the PN model.
- **Step 3** The tableau model checker then verifies the properties on that state space.
- **Step 4** The output of the model checker shows either the specification properties are true or generates a counter model.
- The framework also incorporates an ontology reasoner to guide the verification process.

Tableau MC For Bounded Time CTL

(1) $\dfrac{\neg\neg\phi^s}{\phi^s}$ (2) $\dfrac{\neg\top^s}{\bot^s}$ (3) $\dfrac{\neg\bot^s}{\top^s}$ (4) $\dfrac{(\phi \vee \psi)^s}{\phi^s \mid \psi^s}$

(5) $\dfrac{\neg(\phi \vee \psi)^s}{\neg\phi^s\ ,\ \neg\psi^s}$ (6) $\dfrac{AX\phi^s}{\wedge\phi^{s'}\ \ \forall s' : s \to s'}$ (7) $\dfrac{EX\phi^s}{\vee\phi^{s'}\ \ \forall s' : s \to s'}$

(8) $\dfrac{AG_n\phi^s}{\phi^s \wedge time(s) \leq n\ ,\ \wedge AG_m\phi^{s'}\ \ \forall s' : s \to s'}$

(9) $\dfrac{AF_n\phi^s}{\phi^s \wedge time(s) \leq n \mid \wedge AF_m\phi^{s'}\ \ \forall s' : s \to s'}$

(10) $\dfrac{A_n(\phi U \psi)^s}{\psi^s \wedge time(s) \leq n \mid (\phi^s \wedge A_m(\phi U \psi))^{s'}\ \ \forall s' : s \to s'}$

(11) $\dfrac{EG_n\phi^s}{\phi^s \wedge time(s) \leq n\ ,\ \vee EG_m\phi^{s'}\ \ \forall s' : s \to s'}$

(12) $\dfrac{EF_n\phi^s}{\phi^s \wedge time(s) \leq n \mid \vee EF_m\phi^{s'}\ \ \forall s' : s \to s'}$

(13) $\dfrac{E_n(\phi U \psi)^s}{\psi^s \wedge time(s) \leq n \mid (\phi^s \wedge E_m(\phi U \psi))^{s'}\ \ \forall s' : s \to s'}$

Here rule 6 states that for a Kripke model M, the formula $AX\ \phi$ is true in state s iff for all states s' adjacent to s, ϕ is also true in s'. An example

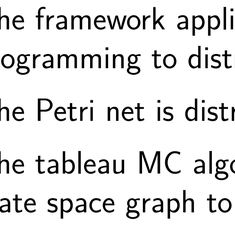

Figure: A simple state space

$AX(\phi \vee \psi)^s$

1. $(\phi \vee \psi)^{s'} \wedge (\phi \vee \psi)^{s"}$ – by 6
2. $(\phi \vee \psi)^{s'}, (\phi \vee \psi)^{s"}$ – by 5
3. $\phi^{s'}, (\phi \vee \psi)^{s"} \mid \psi^{s'}, (\phi \vee \psi)^{s"}$ – by 4
4. Now the algorithm searches the state space to see whether ϕ and ψ are true or not, in states s' and $s"$ respectively.

Distributed Memory Computing

Our framework uses distributed computing as follows:

1. The Petri net model is distributed over processors to generate the state space.
2. Each processor will search only its local states.

Distribution of the PN model can be, by distributing the places or the transitions, or according to a cutting set. Distribution by places increases inter-process communication. Computing the cutting set involves extra effort, can be problematic, and is application dependent. Our framework represents the Petri net in memory, as a list of transitions containing the input and output places.

- The framework applies the SPMD paradigm of parallel programming to distribute the computation task.
- The Petri net is distributed by transitions.
- The tableau MC algorithm will search the distributed state space graph to find out which propositions are true in which state.

Related Works

K. Miller (in 2009) developed a model checker for ontology driven workflow verification. It involved a manual translation of a PN model to its state space, the specification language was Timed BDI$_{CTL}$ and the implementation used Prolog in a shared memory architecture.
J. Dallien (in 2007) developed a model checker for health care workflow verification that involved a manual translation of a PN model to its state space, the specification language was $ACTL$ and a serial Prolog implementation was used.

Discussion

One major difference in the proposed framework is the implementation language is now MPI C++, which makes the system architecture easier to extend than the unstructured Prolog implementation. Also C++ has been successfully used and proved to be efficient in various theorem provers. The distributed memory computation makes the system capable of handling very large model efficiently.
This project is the first author's M.Sc. thesis, the completed project will have important implications to safety critical system design.

This research is part of a research and development project, Building Decision-support through Dynamic Workflow Systems for Health Care, including researchers from the StFX Centre for Logic and Information, their industry partner and the local health authority.

For more details about this project please contact Md Zahidul Islam at x2010mce@stfx.ca. Some related work can be found at http://logic.stfx.ca.

This research has been supported by the Natural Sciences and Engineering Research Council of Canada, Atlantic Computational Excellence network (ACEnet) and by the Atlantic Canada Opportunities Agency through an Atlantic Innovation Fund Award.