Vol 8. No. 3 Issue 2 – October, 2015
African Journal of Computing & ICT

www.ajocict.net

An Industrial Integrated Tool Support

Babatunde A. O
Department of Computer Science
University of Ilorin
Ilorin, Nigeria
babatunde.ao@unilorin.edu.ng

Gbadeyan J.A
Department of Mathematic
University of Ilorin
Ilorin, Nigeria
Gbadeyan.ja@unilorin.edu.ng

Olabiyisi S. O
Department of Computer Science and Engineering
Lautech, Ogbomoso
soolabiyisi@lautech.edu.ng

ABSTRACT

A Theorem Prover is a Computer Program that automates logical reasoning of finding proofs for some mathematical theorems. Examples of such tools are A Computational Logic for Applicative Common Lisp (ACL2) and Prototype Verification System (PVS). The motivation for this paper was the observation that ACL2 tool can prove theorems in first order logic only while PVS tool proves theorems in higher order logic only. The above twotools which are application programs are neither flexible, nor scalable and therefore cannot prove some theorems within their domains. It was also observed that certain theorems exist for which ACL2 and PVS tools could only generate partial proofs. The aim of this paper was therefore to design a single tool that has the ability to generate proofs of some theorems of the two tools. The method used involved carrying out evaluation on the response of each of the tools to theorem problems. In the process, set notations were used. In particular, the tools were defined and represented as sets with their attributes representing members of the set. Integration was then carried out based on direct mapping of the two sets to obtain members of the set of the new tool. Furthermore, an algorithm was developed, and a Delphi Pascal programming language was used to implement the integration of the two tools. The findings showed that the developed tool is able to prove some theorems in set theory, e.g., equivalent set and Cartesian product set and also support proof of some real numbers analysis e.g. Cartesian product and relation equivalent among others . The new tool designed called BT tool is also flexible and scalable.

Keywords: Industrial Tool, Integration, Support, Computational Logic, Application Common Lisps (ACL2) & Prototype Verification System (PVS)

African Journal of Computing & ICT Reference Format:
A.O Babatunde, J.A. Gbadeyan & S.O. Olabiyisi (2015): An Industrial Integrated Tool Support . Afri J Comp & ICTs Vol 8, No.3 Issue 2 Pp 1-12.

INTRODUCTION

Integration describe a link between two tools that enables the strengths of properties of each tools to be deployed smoothly within a single formal development. Integration of formal methods can happen on many levels, including tools, languages, models, notations, methods, and techniques. Tool integration can happen on many levels. Possible choices and related issues are language extension (i.e. embedding): cumbersome, slow and inconvenient. , hard translation: (error prone) ,Point-to-point translation (i.e. features of one tool are added to another): does not produce elegant results, but could be acceptable.

Generic frameworks: seems like the best approach to find some commonality between the tools and "glue" them together using this commonality. The information shared as the "glue" must exist in both tools. There are many possible commonalities between the tools, including: tools sharing common language (e.g. VHDL); tools sharing common data; tools having underlying inference systems that can be specified in rewriting logic. Integration of tools ought to share common interface. One approach is to have one tool produce output to be fed into another tool; another would be to be able to call one tool without exciting the other (i.e. have shared data).

The first approach is easier to implement but works more slowly in practice. Formal methods tools can be integrated into the existing "non-formal" toolkits, or can be integrated into separate formal toolkits such as Telelogic Tau toolkit. "Non-formal" toolkits are widely accepted in industry. Theorem provers on the other hand are computer programs that automate the reasoning of finding proofs within a mathematical theory [**Woodraw *et al*, 1985**]. Examples of theorem Provers are Prototype Verification System (PVS) and A computational Logic for Application Common Lisp (ACL2).

To understand what *automated reasoning is,* we must first understand what reasoning is. Reasoning is the process of drawing conclusions from facts. These conclusions must follow inevitably from the facts from which they are drawn. In other words, reasoning is not concerned with some conclusions that has a good chance of being true when the facts are true. Indeed, reasoning refers to logical reasoning, not of common-sense reasoning or probabilistic reasoning. The only conclusions that are acceptable are those that follow logically from the supplied facts. Automated reasoning is concerned with the study of using the computer to assist in the part of problem solving that requires reasoning [**Wos, 1985**] We can easily see that *automated theorem provers* are t he product of the automated reasoning field. The idea of automated in reasoning is not new. Many of the greatest mathematicians and computer scientists of the century had thought of automated reasoning. All the historical information about the development of logic can be found in Martin Davis' survey article [**Davies, 1983**].

Leibniz recognized the necessity of three basic elements for automated reasoning: 1) A formal language, 2) Formal rules of inferences, and 3) Knowledge. In the nineteenth century, **George Boole** developed the propositional calculus which provided a language and a set of inference rules in which much ordinary common-sense reasoning can be expressed. The advantage of this language was that there was a procedure that would determine whether any sentence in the language was true or false in a finite amount of time. Unfortunately, the language of propositional logic is not expressive enough. In 1879, Gottlob Frege expanded the propositional language to full *first-order logic* which allows much more complex statements to be expressed and verified. It was **David Hilbert** in the early 1920's who initiated a research program in which one of its goals was to discover a systematic procedure that would decide the truth or falsity of any statement in **Frege**'s first-order logic. Unfortunately, in the 1930's, Church and Turning, based in Godel's work, independently discovered that there is no procedure that will decide whether any given statement in first-order logic is true or false [**Alonzo, 1995 and Harry *et al*, 1981].** The decidability of the satisfiability of the first-order logic can be obtained by applying a reduction method to translate the first-order formula to any of the special classes of first-order formulas known for which there exists a procedure to determine the truth value [**Alonzo, 1995].** One of these translation or reduce methods is the *resolution method* proposed by Robinson [**Robinson, 1965**] .

A *proof* is a structure or sequence of well-formed formulas that can be built using a procedure in a finite amount of time, if each of the well-formed formulas in the sequence is either an axiom or is immediately derived from preceding well-formed formulas in the sequence by means of one of the *rules of inference2* [**Alonzo, 1995].** If the procedure to build a proof is sound then the existence of a proof implies that the sentence is true. If it is complete then for every true sentence there must be a proof. There are several sound and complete procedures to determine the proof of a given first-order formula, such as *sequent calculus and tableaur calculus etc* [**Melvin, 1983**].

1.1 Industrial Uses of Theorem Provers

Commercial use of theorem proving is mostly concentrated in integrated circuit design and verification,
e.g since the Pentium FDIV bug, the complicated floating point units of modern microprocessors have been designed with extra scrutiny for removing bugs, and in the latest processors from AMD, Intel, and others, automated theorem proving has been used to verify that division and other operations are correct.

1.2 Theorem Provers of ACL2 and PVS tools

ACL2; (A Computational Logic for Applicative Common Lsp): is the name of a functional programming language based on Common Lisp, a first-order mathematical logic and a mechanical theorem provers. The theorem prover is used to prove theorems in the logic i.e theorems about functions defined in the programming language. ACL2, is sometimes called an "industrial strength version of the Boyer-Moore system," a product of Kaufmann and Moore, with many early design contributions by Boyer. The ACL2 theorem prover is interactive in the sense that the user is responsible for the strategy used in proofs. But it is automatic in the sense that once started on a problem, it proceeds without human assistance. In the hands of an experienced user, the theorem prover can produce proofs of complicated theorems.

The ACL2 theorem prover is a computer program that takes formulas as input and tries to find mathematical proofs. It uses rewriting, decision procedures, mathematical induction and many other proof techniques to prove theorems in a first-order mathematical theory of recursively defined functions and inductively constructed objects [Alonzo, 1995]. It has been used for a variety of important formal methods projects of industrial and commercial interest, including: Verification that the register-transfer level description of the AMD Athlon processor's elementary floating point arithmetic circuitry implements the IEEE floating point standard **[Russinoff, 1998 and Russinoff et al, 2000]**. Similar work has also been done for components of the AMD k5 processor **[Moore et al, 1998],** the IBM Power 4 **[Sawada, 2002]** and the AMD Opteron processor, verification that a micro architectural model of a Motorola digital signal processor (DSP) implements a given microcode engine **[Brock et al, 1999]** and verification that specific microcode extracted from the ROM implements certain DSP algorithms **[Brock et al, 1999],** verification that microcode for the Rockwell Collins AAMP7 implements a given security policy.

This has to do with process separation **[David et al, 2003],** Verification that the JVM bytecode produced by the Sun compiler javac on certain simple Java classes implements the claimed functionality **[Moore, 2003]** and the verification of properties of importance to the Sun bytecode verifier as described in JSR-139 for J2ME JVMs **[Liu et al, 2003],** Verification of the soundness and completeness of a Lisp implementation of a BDD package that has achieved runtime speeds of about 60% those of the CUDD package (however, unlike CUDD, the verified package does not support dynamic variable reordering and is thus more limited in scope) [Sumner, 2000], Verification of the soundness of a lisp program that checks the proofs produced by the Ivy theorem prover from Argonne National Labs; Ivy proofs may thus be generated by unverified code but confirmed to be proofs by a verified Lisp function [McCune, 2000].

Prototype Verification System (PVS) on the other hand consists of a specification language, a number of predefined theories, a type checker, an interactive theorem prover that supports the use of several decision procedures and a symbolic model checker, various utilities, including a code generator and a random tester, documentation, formalized libraries, and examples that illustrates different methods of using the system in several application areas. It consists of specification language; a number of predefined theories, a theorem prover, of various utilities, documentation and have various examples illustrating deferent methods of using the system in several application areas. **(Owrel et al, 1996).** Typical applications of PVS include the formalization of mathematical concepts and proofs in areas such as analysis, graph theory, and number theory, the embedding of formalisms such as I/O automata, modal and temporal logics, the verification of hardware, sequential and distributed algorithms, and as a back-end verification tool for computer algebra as well as code verification systems.

1.3 Related Works

The claim that the lack of tools is one of the major reasons for the difficulties of incorporating formal methods in industry is a misconception and a myth (**Bowen and Hinchey, 1995).** The actual reason is the lack of adequate, powerful, user friendly strength tools that will aid the application of formal methods to industry and allow them to be fully integrated with existing methods (**Butler, 2001**). The importance of providing means for connecting with external tools has been widely recognized in the theorem proving community. Some early ideas for connecting different theorem provers are discussed in a proposal for the so-called "interface logics" [**Guttman, 1991**], with the goal to connect automated reasoning tools by defining a single logic *L* such that the logics of the individual tools can be viewed as sub-logics of *L*. More recently, with the success of model checkers and Boolean satisfiability solvers, there has been significant work connecting such tools with interactive theorem provers. The PVS theorem prover provides connections with several decision procedures such as model checkers and SAT solvers [**Rajan *et al,* 1995 and Shanker, 2001**].

The Isabelle theorem prover [**Nipkow *et al*, 2002**] uses unverified external tools as *oracles* for checking formulas as theorems during a proof search; this mechanism has been used to integrate model checkers and arithmetic decision procedures with Isabelle [**Muller *et al*, 1995 and Basin *et al*, 2000**]. Oracles are also used in the HOL family of higher order logic theorem provers [**Gordon *et al*, 1993**], for instance, the PROSPER project [**Dannis *et al*, 2000**], uses the HOL98 theorem prover-as a uniform and logically-based coordination mechanism, between several verification tools. The most recent incarnation of this family of theorem provers, HOL4, uses an external oracle interface to decide large Boolean formulas through connections to state-of-the-art of Binary Decision Diary (BDD) tool and SAT-solving libraries tool [**Gordon, 2002**], and also uses oracle interface to connect HOL4 with ACL2. **(Meng and Paulson , 2004**], interface Isabelle with a resolution theorem prover.

In 1991, Fink et al, described a proof manager called PM [**George *et al*, 1991**], that enabled HOL input to be transformed into "first-order assertions suited to the Boyer-Moore prover." In 1999 **Mark Staples** implemented a tool called ACL2PII for linking ACL2 and HOL98 [**Mark, 1991**]. ACL2PII was used by Susanto and Melham **[Kong, 2003]**. Both PM and ACL2PII provided ways of translating between higher-order logic and first-order logic., when translating from untyped Boyer-Moore logic to typed higher-order logic it can be hard to figure out which types to assign. Staples points out that the ACL2 S-expression NIL might need to be translated to F (Boolean type), or [ ] (list type) or NONE (option type), depending on context. The ACL2PII user has to set up "translation specifications" that are pattern-matching rewrite rules to perform the ACL2-to-HOL translation. These are encoded in ML and are thus not supported by any formal validation. In 2006, Mike Gordon, Warren A. and Matt Kaufmann also integrated HOL and ACL2.

1.4 Motivation for Integrating ACL2 and PVS Tools

Research showed that ACL2 tool can only prove theorems in first order logic while PVS tool prove theorems in higher order logic only. The above two tools which are also application programs designed to prove some selected theorems are not flexible, not scalable and therefore cannot prove some theorems within their domains. It was also observed that certain theorems exist for which ACL2 and PVS tools could only generate partial solutions, hence a motivation for their integration.

2. MATERIAL AND METHOD

The study worked on existing tools and carried out performance evaluation of the existing theorem provers, case study of ACL2 and PVS tools. The evaluation was carried out based on the response of each of the tools to theorem problems for them to prove. And as a result, a list of attributes of each of the tools was obtained, based on the responses and performance of the provers on selected theorems (problems). Using set notations, the tools are defined and represented as set with their attributes representing members of the set. Integration was carried out based on the direct mapping of the two sets (ACL2 and PVS) to obtain the members of the set of the new tool.

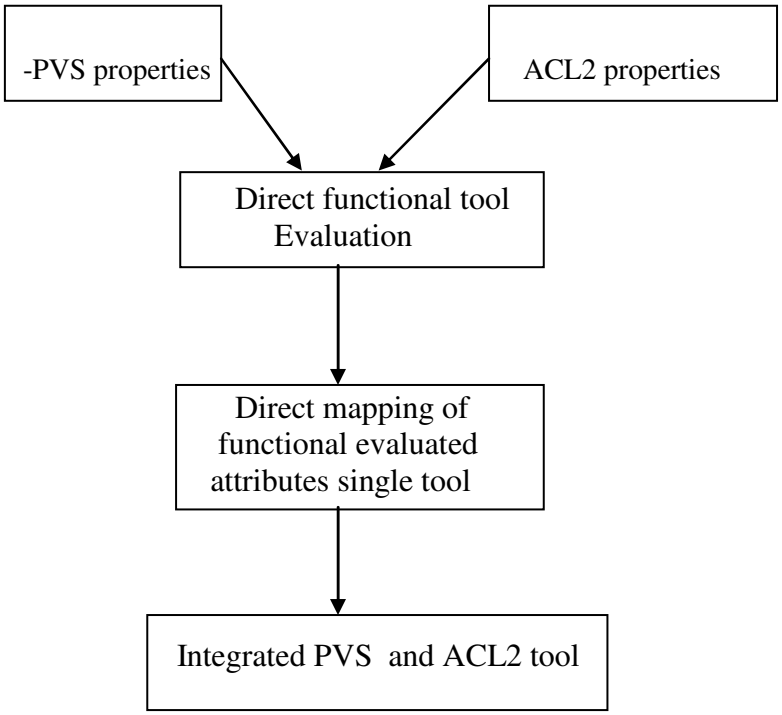
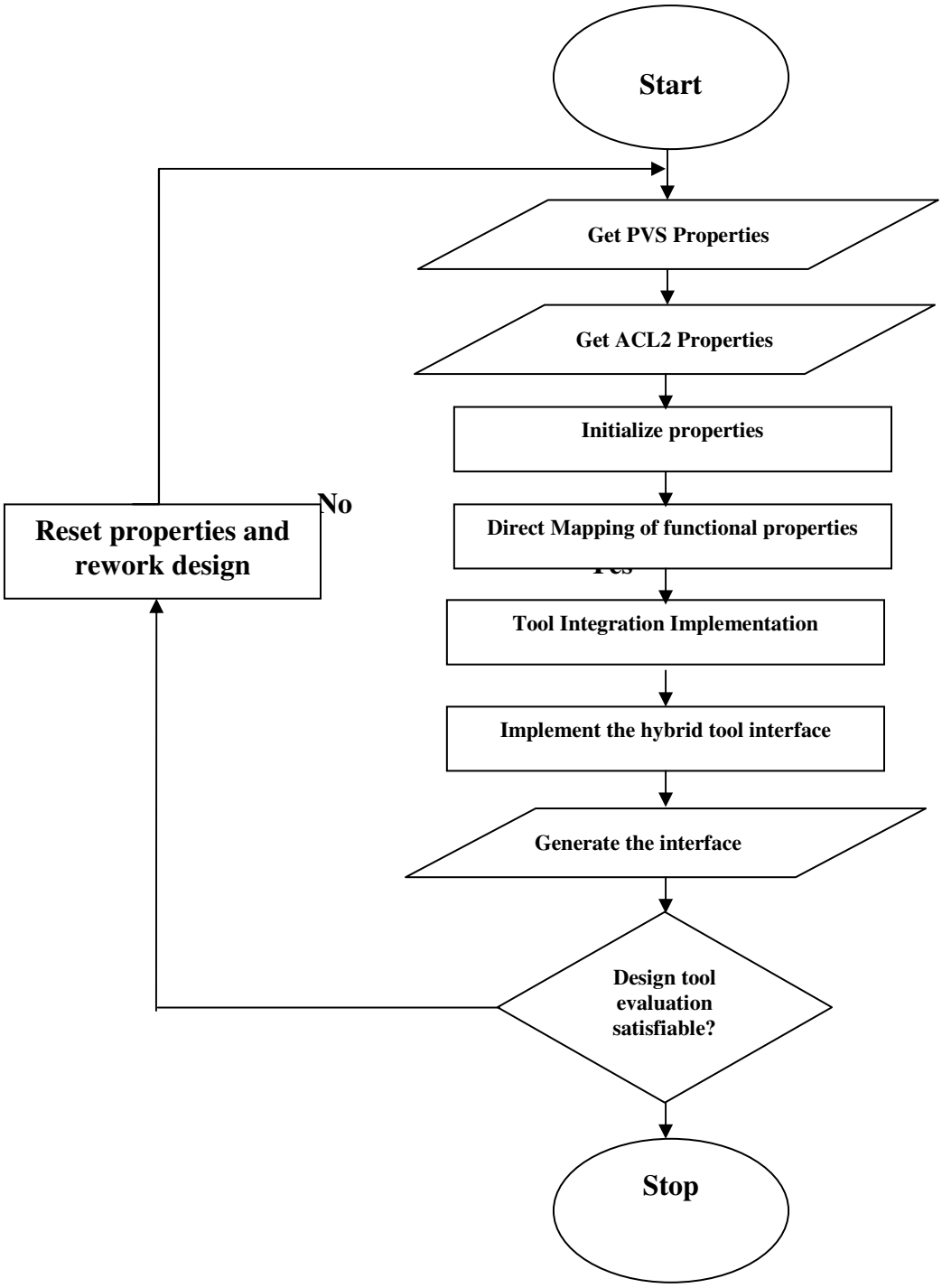


Fig 3.1Framework for the integration of ACL2 and PVS tools



3.2 Flowchart showing the integration of ACL2 and PVS tools

3.2.1 Functional system design representation

Let x represent a logic theorem (problem). Note that a logic theorem represents the computational problem solvable or provable by theorem provers (ACL2, PVS) under consideration.

Let $x_p(1)$, $x_p(2)$, $x_p(3)$, $x_p(4)$, $x_p(5)$ represent the intrinsic (characteristics) properties of X.

Such that, X= { $x_p(1)$, $x_p(2)$, $x_p(3)$, $x_p(4)$, $x_p(5)$}

X is made up of sub-problems

Challenge

Derive a single tool (theorem prover) efficient enough to prove $x_p(1)$, $x_p(2)$, $x_p(3)$, $x_p(4)$, $x_p(5)$ at a single evaluation

After evaluation of the problem (theorems) using ACL2 and PVS tools

Observation 1

With ACL2 theorem prover

X is partially solvable

Assumption

$x_p(1)$, $x_p(2)$ are solvable using ACL2

$x_p(5)$, $x_p(3)$, $x_p(4)$ remain unsolvable after evaluation using ACL2.

Observation 2

With PVS theorem prover

X is still partially solvable

Assumption

$x_p(1)$, $x_p(2)$ are unsolvable using PVS

$x_p(3)$, $x_p(4)$, $x_p(5)$ are solvable by PVS

Note that $x_p(5)$ is solvable by both i.e $x_p(5)$ is not a concern of this study.

TASK: Integration of functional properties of ACL2 and PVS tools efficient enough to prove all the sub-problem at once.

3.2.2 Mathematical representation of PVS tool using set notation

The attributes of PVS tool are:

i. Prove some theorem in set theory (ST) e.g. equivalent set and Cartesian product set.
ii. Supports prove of some real numbers analysis (RN) e.g. Cartesian product and relation equivalent.
iii. Supports prove of some quantifier reasoning (QR) e.g. expression translation from logical reasoning to English expressions.
iv. Some Mathematical concepts formalization proving(MCF) e.g. character case support and Fibonacci support
v. Support some inductive proof checking (IPC) e.g. principle of mathematical induction, well-typed functions and complex rational analysis.

Let the attributes of PVS tools be represented in terms of set notation

Thus: **PVS= {ST, RN, QR, MCF, IPC}**

3.2.2 Algorithmic Design of the proposed integrated tools

1. Start the design
2. Get PVS properties
3. Get ACL2 properties
4. Initialize the properties
5. Map the functional properties of the tools directly
6. Implement tool integration modules
7. Integrate PVS and ACL2 tool
8. Implement the Interface
9. Generate the interface
10. If design tool evaluation is satisfiable goto 11 else goto 2
11. Stop

3.2.3 Mathematical representation of ACL2 using set notation

The attributes of ACL2 tool are:

i. Some inductive proof checking (IPC) e.g. principle of mathematical induction, well typed functions and complex rational analysis.
ii. Some complex rationales support(RCR)
iii. Some well typed functions support (WTF)

Let the attributes of ACL2 tool be represented in terms of set notations

Thus: ACL2= {IPC, RCR, WTF}

3.2.4 Mathematical representation of Integrated Theorem Prover (ITP)

Let ITP tool represent the integration of ACL2 and PVS tools.

In terms of set relation, ACL2 and PVS tool are subset of ITP tool

ACL2 ϵ ITP

PVS ϵ ITP

ITP= {ACL2, PVS}

ITP= ACL2 U PVS

Thus:

ITP = {ST, RN, QR, MCF, IPC, RCR, WTF}

It can be said that, the union of the member of the ACL2 and PVS subset are members of the ITP set.

3.2.5 System (pseudocode) design

```
Class pvs_acl2 (pvs, acl2, bt, pvs_att, acl2_att, bt_att)

begin
        pvs_att:=pvs;
        acl2_att:=acl2;
        bt_att:= nil;
        bt_att:=pvs_att + bt_att;
        bt_att:=acl2_att + bt_att;
        bt:= bt_att;
end;
```

4. RESULTS AND DISCUSSION

4.1 Implementation of the Design

The figure 4.2 below explicitly analyses the attributes of ACL2 and PVS and then incorporated the attributes to a new tool defined as BT Tool. The attributes of the ACL2 tool is a shown below. The ACL2 tool takes formulae as input and find mathematical proofs. It uses rewriting decision procedures, mathematical induction and many other proof techniques to prove theorems in a first-order mathematical theory of recursively defined functions and inductively constructed objects in the integrated design BT tool (Alonzo, 1995). The PVS tool in the design support several decision procedures and various utilities, documentation and formalized libraries, that illustrates different methods of using the designed integrated BT tool for several application areas. It also support proofs theorems in set theory and prove of some complex rationals. It helps to translates quantifiers logical input of the designed BT tool to a more conceptual format. The inference rules can be supplied as inputs and the definitions will be generated as output by the integrated BT tool. It also takes hints or lemma as input, stimulates it and displays the closest related definition and the proof as output. It takes a concept definition as input, formalized it and displays the syntax equivalent as output in the BT tool. It also takes a set of theorems as input, stimulates the input and displays the closest related definition and the proof in the designed BT tool as output.

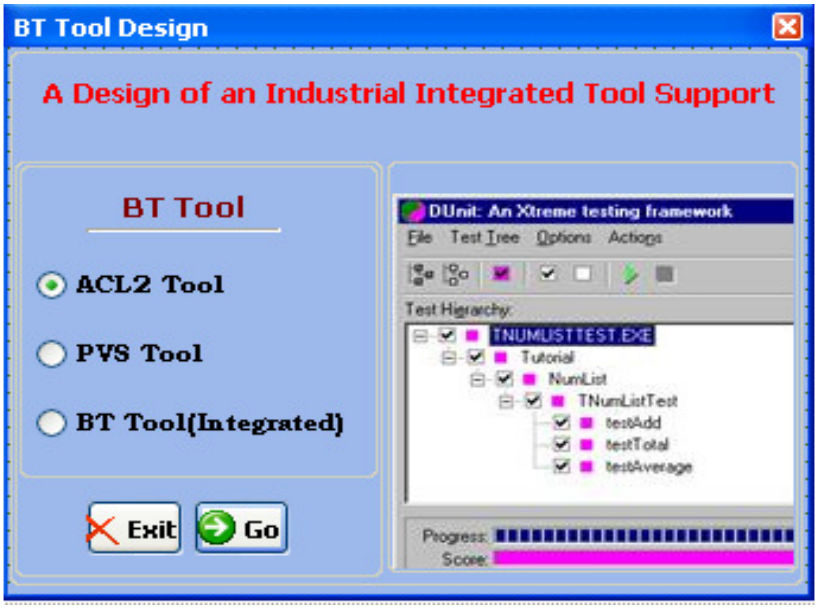


Figure 4.2: Start-Up Page for the BT Tool design

The attributes of ACL2 as shown in figure 4.3 include complex rationals support , well-typed functions and support Inductive Proof Checker. These attributes as incorporated in ACL2 makes its function independently as a tool before being incorporated in the integrated BT tool. The function include taking formulae as input and finding it mathematical proofs. It also takes decision procedures, mathematical induction and many other proof techniques to prove theorems. For examples it takes real value numerator, real value denumerator, operator, imaginary value numerator, and imaginary value denumerator in the complex support prove as input to generates the detail equation as output solution. Also Figure 4.4 shows an example of the implementation of operational behavior of the complex rational support attribute of ACL2. It takes RealVal Numerator, RealVar Denominator, Operator, Imaginary Value Numerator and Imaginary Value Denominator as input at two consecutive iterations and then generates the detailed equation and the overall solution to the problem as output.

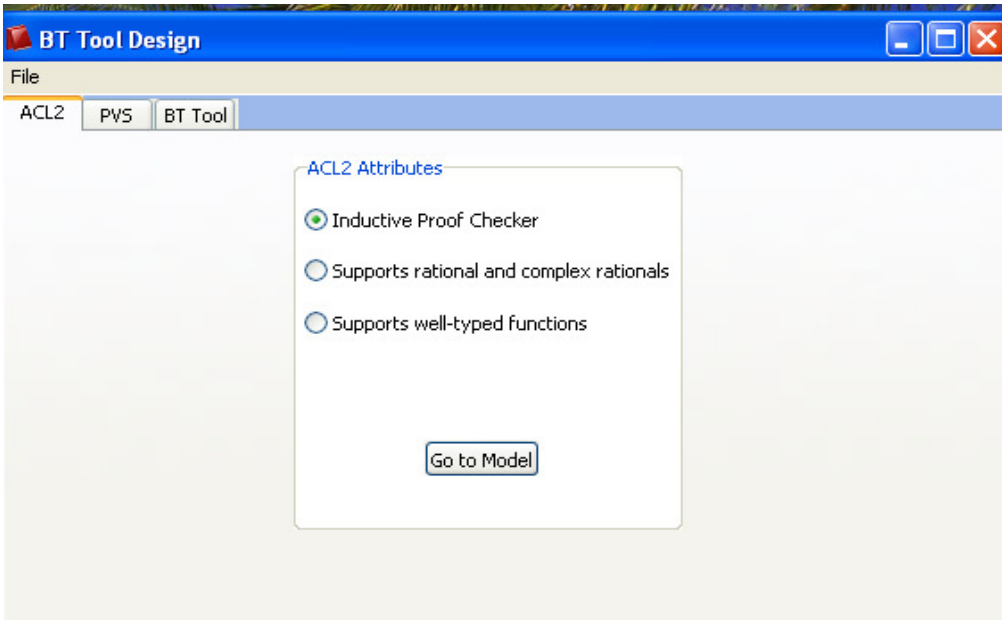


Figure 4.3: Testing of the Attributes of ACL2 tool in the Design

4.5 Testing of the Attributes of PVS tool in the Design

The figure 4.5 shows the attributes of PVS. The basic attributes include support for theorems in set theory, supports for real number analysis, support for quantifiers reasoning, supports for proof of analysis and support for formalization of mathematical concepts. These attributes are inherited in PVS tool before it is integrated in the designed BT tool. The above attributes helps to translate logical input to a more conceptual understanding. And also supplies inferences as input and generates the resultant definition as output. For example it takes a concept definition as input formalized the input and display the syntax equivalence as output. It also takes a set theorem as input, stimulates the input and displays the closest related definition as output.

.

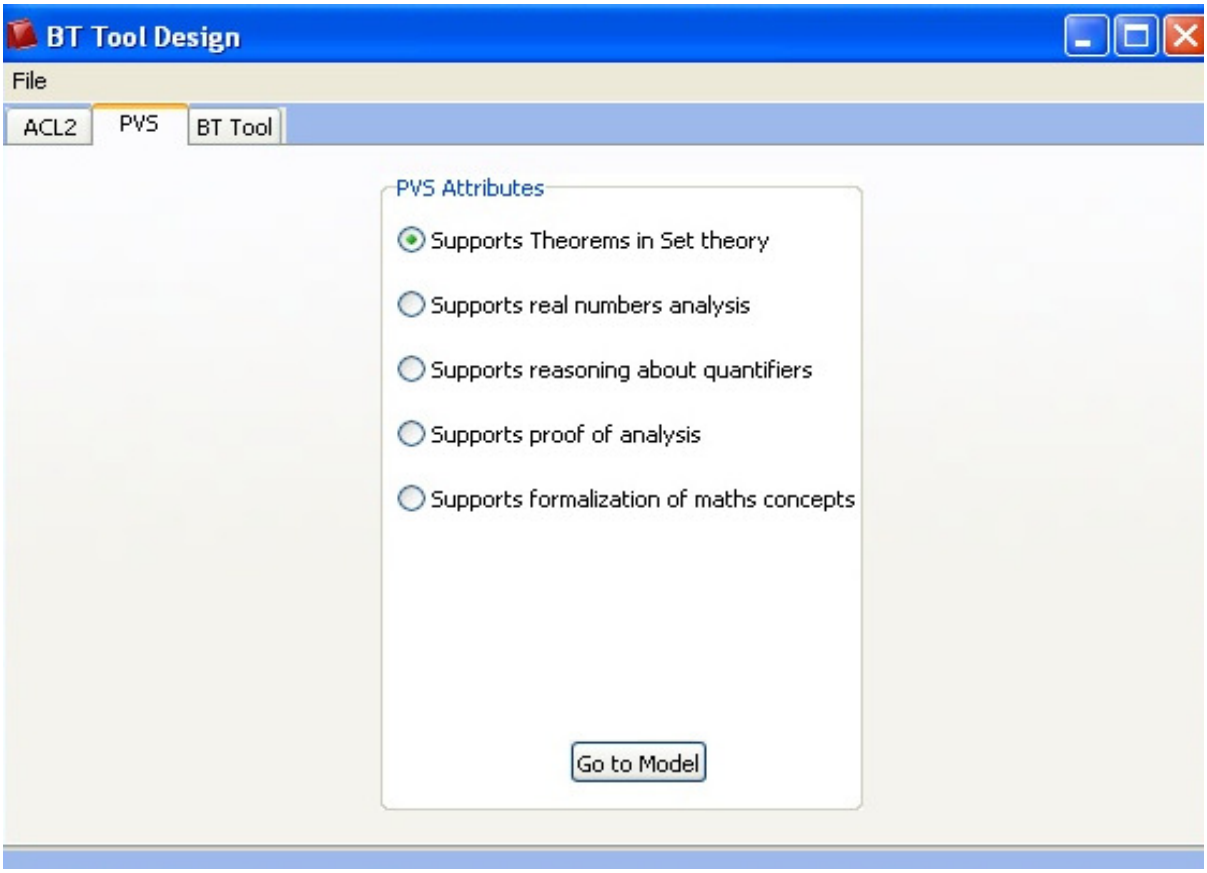


Figure 4.5: Attributes of PVS tool in the Design

.

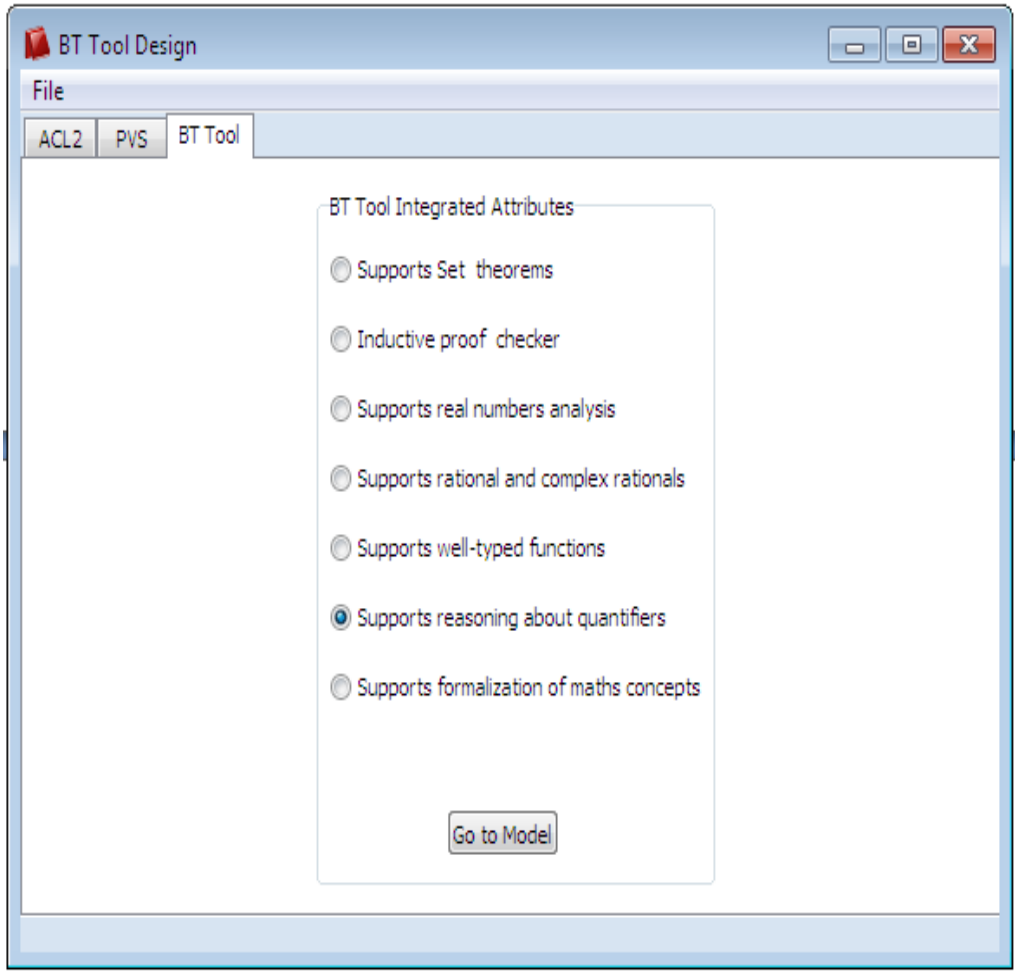


Figure 4.6: Testing of the Attributes of the New Design BT Tool

4.7 Performance Evaluation

Performance evaluation can be defined as assigning quantitative values to the indices of the performance of the system under study. Evaluating and analyzing integrated system is difficult due to the complex interaction between application characteristics and architectural features. To study the performance of the integrated BT tool designed and the existing PVS tool and ACL2 tool. The following parameters are used:

a. *Output statistics*-this parameter examines the capabilities of the technique towards providing the desirable integrated tool.
b. *Accuracy*-this factor evaluates the validity (ability of a tool to achieve its objective) and reliability (ability of a tool to meet its requirement specification) of the integrated tool.
c. *Cost/effort*-this parameter investigates the cost and effort invested in each performance evaluation strategy in context with computer and human resources.
d. *Resource consumption*-this parameter examines the amount of resources consumed/required by the performance of the new integrated tool.
e. *Time consumption*- this parameter examines the amount of time consumed/required by the performance of the designed tool.
f. *Trustability/Believability*- these parameters reveals how much one can trust on the results of performance of the integrated tools.
g. *Scalability complexity*-this parameter examines the ability of the integrated tool acceptance of other tools attributes or complexity involved in scaling during performance of the integrated designed tool.
h. *Flexibility*-this parameter examines the flexibility of performance towards adapting the modifications or inherited attributes made to the integrated tool and checking their effect.

TABLE1: Showing comparison of performance evaluation techniques

CHARACTERISTICS	ACL2	PVS	BT TOOL (INTEGRATED TOOL)
Output Statistics	Low	Medium	High
Accuracy	Medium	Medium	High
Cost/Effort	Low	Medium	High
Resource consumption	medium	High	Low
Time consumption	medium	High	Low
Trustability	Low	Medium	High
Scalability	None	None	High
Flexibility	None	None	High

4.8.1 Analysis of Tools Performance

From the table 1 above, it is seen that the integrated BT tool designed is very flexible and scalable compare to ACL2 and PVS tool which is neither scalable nor flexible . The PVS tool is also more Trustable than ACL2 tool in its performance. The cost and effort of getting the integrated BT tool designed is more than that of the PVS tool and ACL2 tool. Though the cost and effort of getting ACL2 tool is lower than that of the PVS tool. The time taking for the performance of the integrated BT tool design is however less compare to PVS tool. PVS tool also spend more time in its performance compare to ACL2 tool. The integrated BT tool design consume less resources for its performance compare to PVS tool (high) and ACL2 tool (medium). The PVS and ACL2 tools have lower performance output than the integrated BT tool. The above analysis of the three tools show that the integrated BT tool has overall best performance evaluation than the existing PVS and ACL2 tools.

5. CONCLUSION

A design and implementation of an Industrial Integrated Tool Support was carried out in this study. The new tool designed hereby called BT tool inherited the attributes of ACL2 and PVS tools. The new tool developed is able to prove theorem in set theory and prove of some complex rationals. The tool developed was also flexible (integrate well with existing tools) and scalable (accept attributes of existing tools and any other tools attributes that want to integrate with it in the future). The new tool is of economic advantage to industries because it saves time and money. The study has been able to increase the number of formal methods tools in industry.

REFERENCES

[1] Butler R.W. (2001) "What is Formal Methods? Retrieved on 2006.Michael Hollowing (2006) "Why Engineers should consider Formal Methods. 16th Digital System Conference.Clake E.M. 398- 401

[2] Craigen D. Gerhart S. (1995) "Formal Methods reality check, Industrial usage, IEEE Trans Software Engineering 21(2), 90-98.

[3] Formal Methods Europe http://www.fmeurope.org

[4] Formal Methods Virtual Library. http://www.afm.sbu.ac.uk/

[5] Gordon M. (2002), "Programming Combinations of Deductions and BDD-based Symbolic Calculation, Journal of Computation and Mathematics pp. 56-76.

[6] Guttman J.D. (1991) "a Proposal Interface Logic for Verification Environment, Tech Rep pp. 19-91.

[7] Kefas, P and Kapeti, E (2000) "A Design Language and Tool for X machine specification" World Scientist Publishing Company pp 134-135.

[8] Michael J.,Gordon C,Warren A.,and James R.(2006) "An Integration of HOL and ACL2" IEEE Computer Society Press,pp 153-160.

[9] Mike G. Warren A. and Kaufman, M. 2006 "An integration of HOL and ACL2 fifth International Conference on integrated formal methods tools" Dec 2005 Netherland.

[10] Moore J., Lynch T. and Kaufmann, M. (1998) "A Mechanically checked proof of the Correctness of the Kenel of the AMD5K86 floating - point division Algorithm, IEEE Transactions on Computer 47(9), pp. 913-926.

[11] Rojan, S. Shanker M. and Strivas K. (1995), "An Integration of Model Checking with Automated Proof Checking" Proceedings of the 8th International Conference on Computed Aided Verification Proving in Higher-order Logic (CAVAS), vol. 939, pp. 84-97.

[12] Russinoff, D. (1998) "A Mechanically checked proof of IEEE compliance of a register transfer - level specification of the AMD-K7 floating - point multiplication, division and square root instructions" London Mathematics Journal of Computation and Mathematics (1) pp. 448-200.

[13] Sawada J. (2002), "Formal Verification of Divide and Square root Algorithms using series calculation" Proceedings of the ACL2 Workshop, Grenoble.

[14] Shanker, N. (2001), "Using Decision Procedures with Higher Order Logics," Proceedings of the 14th International Conference on Theorem Proving in Higher-Order Logics Vol. 2152 pp. 5-26.

[15] Somerville I. (2001) "Software Engineering, 6th Edition, Addison-Wesle.

[16] Woodrow W. and Henshen L. (1985) "What is Automated Theorem Proving"? Journal of Automated Reasoning (1) 1, pp. 23-28.

[17] Wos, L. (1985) "What is Automated Reasoning? Journal of Automated Reasoning. 1 (1) pp. 6-8.

[18] World Wide Web Virtual Library on Formal Methods (WWWVLFM)