Automatic Generation of Test Oracles - From Pilot Studies to Application

Martin S. Feather
Jet Propulsion Laboratory,
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109, USA
+1 818 354 1194
Martin.S.Feather@Jpl.Nasa.Gov

Ben Smith
Jet Propulsion Laboratory,
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109, USA
+1 818 353 5371
Ben.D.Smith@Jpl.Nasa.Gov

ABSTRACT

There is a trend towards the increased use of automation in V&V. Automation can yield savings in time and effort. For critical systems, where thorough V&V is required, these savings can be substantial.

We describe a progression from pilot studies to development and use of V&V automation. We used pilot studies to ascertain opportunities for, and suitability of, automating various analyses whose results would contribute to V&V. These studies culminated in the development of an automatic generator of automated test oracles. This was then applied and extended in the course of testing an AI planning system that is a key component of an autonomous spacecraft.

Keywords
Test Oracles, Verification and Validation, Analysis, Planning, NASA

1 INTRODUCTION

Cost, performance and functionality concerns are driving a trend towards use of self-sufficient autonomous systems in place of human-controlled mechanisms. Verification and validation (V&V) of such systems is particularly crucial given that they will operate for long periods with little or no human supervision. Furthermore, V&V must itself be done at low cost, rapidly and effectively, even as the systems to which it is applied grow in complexity and sophistication.

Spacecraft – especially deep space probes – exemplify these concerns. We have been involved in V&V of an AI planner that is a key component of a spacecraft's autonomous control system. In [Feather & Smith 1997] we report our use of an automated generator of automated test oracles to support these V&V activities. The paper is organized to show the progression of steps we followed leading up to this application, and the lessons we have learnt by reflecting upon our experience:

- First pilot study: rapid automated analysis (Section 2). In this study we determined the viability of a rapid analysis approach. We did case studies of two kinds of traditional design information, yielding confirmation of the viability of the analysis method for this kind of information.
- Second pilot study: application to an autonomous planner (Section 3). We needed this second study to determine suitability of the rapid analysis approach to, specifically, checking plans generated by an AI planner. Particular concerns were scalability of the approach, and investment of domain experts' time. The pilot study produced instances of automatic test oracles.
- Development of automated generator of planner test oracles (Section 4). Based on the lessons learned from the second pilot study, we committed to developing a tool to be used in actual spacecraft testing. The tool would go beyond the capabilities of the second pilot study by both extending aspects of the analyses performed, and automating the generation of the test oracles themselves.
- Application to V&V of spacecraft planner (Section 5). We applied the tool during spacecraft planner testing. Using it, we checked thousands of test cases for adherence to hundreds of flight rules. Additionally, we extended it to perform additional validation checks of particularly complex rules.
- Lessons learned (Section 6). We describe lessons learned for both software engineering:
  - *Our experience re-iterates several well-understood virtues of pilot studies as a precursor to actual development.*
  - *When domain experts' time is a critical resource, follow an "on-demand" policy of knowledge acquisition.*

  and V&V:
  - *V&V can make good use of redundancy and*

The architecture of the system developed in this phase is shown in Figure 3. For the remainder of this paper we will refer to this system as the "planchecker". It has the same stages as the second pilot study, but with some additional capabilities:

- **Additional analyses:** the planner experts asked for further analyses beyond temporal constraints, notably typechecking of plan elements, and cross-checking of plan activities against their rationale (information on which is included in the generated plans). These required loading additional information from plans into the database, and development of additional database queries.
- **Automatic translation:** there were over 200 temporal planner constraints (counting each lowest-level clause as one constraint). Based on the observations of the second pilot study, we recognized that manual translation of the whole set would be a tedious task. Worse yet, we expected the set of planner constraints to grow and change over time. In keeping with our overall goal of judicious use of automation, it was decided build an automatic translator that would take *any* constraint expressible in the planner language and generate the equivalent database query.
- **Extended output:** the planner experts wanted the query results to report more than simply "OK" when a plan passed the checks. In essence, they wanted a justification for *why* a temporal constraint was satisfied. For example, a constraint that says every SEP-thrusting interval is followed by an SEP-idle interval would be justified by listing, for each SEP-thrusting interval, the specific SEP-idle interval found to satisfy the constraint.
- **Coverage analysis:** the planner experts also wanted to know *which* of the planner constraints had been exercised in the plan. For example, only plans that involved intervals of SEP thrusting would exercise a constraint of the form "every thrusting interval must ...".

Insights gained from development experience

The development effort did indeed culminate in the planchecker tool (use of which is discussed in the next section). We therefore confirmed the validity of the conclusions drawn from the second pilot study. We also gained some further insights. These fell into two key areas:

- The second pilot study had suggested that the translation from planner constraints to database queries would be straightforward. In practice, automating the translation of the full planner language turned out to be more complex than the pilot study had indicated (see Appendix B for examples). While a procedural approach to programming the planchecker's translator sufficed to meet the development goals, we concluded that translation warrants further attention. We will return to this in Section 6, Lessons Learned.
- In practice, testers need analysis results with more content and structure than simply "pass" or "fail". Again, details can be found in Appendix B, and discussion is deferred to Section 6. Lessons Learned.

5 USE OF ANALYSIS TOOL

The planchecker was used by the second author (a planning expert) during testing. Interaction with the V&V expert was not required during this phase.

The planchecker was applied to check each plan generated. Its results were accumulated alongside other statistics about the plan generation, e.g., how long it took to generate the plan, how much memory was required to do so. It was easy to apply in "batch mode" to a whole series of plans. It was tolerably efficient, taking on the order of 2 minutes to complete the checking of a typical plan.

Over the course of use, several sets of changes were made

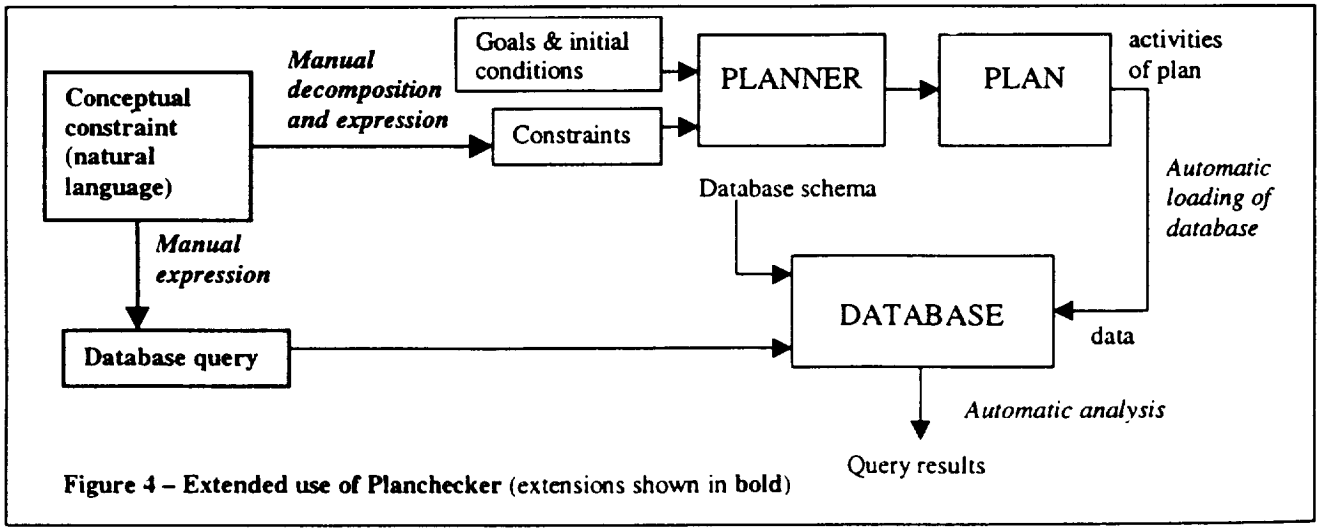


Figure 4 – Extended use of Planchecker (extensions shown in **bold**)

out to be a driving concern. Our database-based approach to analysis sufficed. More important to us was the investment of effort that would be required of our domain experts, whose time was in short supply. This led us to automate the generation of test oracles from a domain-specific representation. Thus the domain experts' effort it would take to construct that generator became out dominant concern. Approaches that could reduce this kind of effort include the parameterized tableaus of [Dillon & Ramakrishna 1996], or the algebraic-signature based mappings of [Reyes & Richardson 1998]. We found, however, the need to yield needed test results with finer distinctions than simply "passed" or "failed." Information about "passed" cases was useful to for test coverage analysis, and for ascertaining that the test had been passed "for the right reasons". Information about "failed" cases was useful to locate the relevant portions of the plan contributing to those failures, and so speed the domain expert in debugging what was going wrong in the planner.

We are not aware of work on automatic generation of test oracles that supports this capability. Based on our practical experience of application of test oracle generation, we see the need for further investigation of this area.

ACKNOWLEDGEMENTS

The research described in this paper was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space administration. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government or the Jet Propulsion Laboratory, California Institute of Technology.

The authors thank the other members of the DS-1 planner team, Nicola Muscettola and Kanna Rajan, for their help.

REFERENCES

[Allen 1983] J.F. Allen. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26(11):832-843, 1983.

[Andrews 1998] J.H. Andrews. Testing using Log File Analysis: Tools, Methods, and Issues. *Proceedings of the 13th IEEE International Conference on Automated Software Engineering* (Honolulu, Hawaii, October 1998), IEEE Computer Society, 157-166.

[Cohen 1989] D. Cohen. Compiling Complex Database Transition Triggers. *Proceedings of the ACM SIGMOD International Conference on the Management of Data* (Portland, Oregon, 1989), ACM Press, 225-234.

[Dillon & Ramakrishna 1996] L.K. Dillon & Y.S. Ramakrishna. Generating Oracles from Your Favorite Temporal Logic Specifications. *Proceedings 4th ACM SIGSOFT Symposium Foundations of Software Engineering* (San Francisco, October 1996), ACM Press, 106-117.

[Dillon & Yu 1994] L. Dillon & Q. Yu. Oracles for checking temporal properties of concurrent systems. *Proceedings 2nd ACM SIGSOFT Symposium Foundations of Software Engineering* (New Orleans, December 1994), ACM Press, 140-153.

[DS1 1998] http://nmp.jpl.nasa.gov/ds1/

[Feather1998] M.S. Feather. Rapid Application of Lightweight Formal Methods for Consistency Analyses. *IEEE Transactions on Software Engineering*, 24(11): 949-959, Nov 1998.

[Feather & Smith 1998]. M.S. Feather & B. Smith. V&V of a Spacecraft's Autonomous Planner through Extended Automation. *Proceedings of the 23rd Annual Software Engineering Workshop* (NASA Goddard, MD, December 1998).

[Jagadeesan et al 1997] L.J. Jagadeesan, A. Proter, C, Puchol, J.C. Ramming & L.G.Votta. Specification-based Testing of Reactive Software: Tools and Experiments. *Proceedings of the 19th International Conference on Software Engineering* (Boston, MA, May 1997), 525-535.

[Pell 1996] B. Pell, D.E. Bernard, S.A. Chien, E. Gat, N. Muscettola, P.P. Nayak, M.D. Wagner & B.C. Williams. A Remote Agent Protoype for Spacecraft Autonomy. *Proceedings of the SPIE conference on Optical Science, Engineering and Instrumentation*, 1996.

[Pell 1997] B. Pell, D.E. Bernard, S.A. Chien, E. Gat, N. Muscettola, P.P. Nayak, M.D. Wagner & B.C. Williams. An Autonomous Spacecraft Agent Prototype. *Proceedings First International Conference on Autonomous Agents*. ACM Press, 1997.

[RAX 1998] http://nmp.jpl.nasa.gov/ds1/tech/autora.html

[Reyes & Richardson 1998] A.A. Reyes & D.J. Richardson. Specification-Based Testing of Ada Units with Low Encapsulation. *Proceedings of the 13th IEEE International Conference on Automated Software Engineering* (Honolulu, Hawaii, October 1998), IEEE Computer Society, 22-31.

[Richardson, Aha & O'Malley 1992] D.J. Richardson, S.L. Aha & T.). O'Malley. Specification-based Test Oracles for Reactive Systems. *Proceedings of the 14th International Conference on Software Engineering* (Melbourne, Australia, May 1992), 105-118.

[SOHO 1998] *SOHO Mission Interruption Preliminary Status and Background Report – July 15, 1998* http://umbra.nascom.nasa.gov/soho/prelim_and_backgr

ound_rept.html

[Wasserman & Blum 1997] H. Wasserman & M. Blum. Software Reliability via Run-Time Result-Checking. JACM 44(6): 826-845, 1997.

[Wile 1997] D. Wile. Abstract Syntax from Concrete Syntax. *Proceedings of the 19th International Conference on Software Engineering* (Boston, MA, May 1997), 472-480.

APPENDIX A - DETAILS OF THE SECOND PILOT STUDY

Example of planner constraint

The following example of one of the simpler plan constraints, as expressed in the planner's special purpose language, will convey a feel for the challenges faced in this pilot study:

```
(Define_Compatibility
  ;;
  ;; Idle_Segment
  ;;
  (SINGLE ((SEP_Schedule SEP_Schedule_SV))
          (Idle_Segment))
    :duration_bounds [1 _plus_infinity_]
  :compatibility_spec
  (AND
   ;; Thrust and Idle segments must all
meet--no gaps
    (meets
     (SINGLE
       ((SEP_Schedule SEP_Schedule_SV))
       ((Thrust_Segment (?_any_value_
                         ?_any_value_)))))
    (met_by
     (SINGLE
       ((SEP_Schedule SEP_Schedule_SV))
       ((Thrust_Segment (?_any_value_
                         ?_any_value_)))))))
```

This illustrates several areas where knowledge held by the planner experts had to be acquired by the V&V expert:

- **Overall application domain knowledge:** "SEP" is an acronym for "Solar Electric Propulsion," the innovative engine that provides thrust to DS-1. "Thrust" and "Idle" are the two main states this engine can be in.

  Knowledge such as this of the spacecraft domain provided useful intuition to the V&V expert, and this second pilot study warranted a deeper level of understanding than had been necessary for the first pilot study.

- **Problem-specific terminology:** "SINGLE" has a connotation specific to DS-1's planner. It introduces a description that matches a single interval. (One alternatives is "MULTIPLE," introducing a description that matches a contiguous sequence of intervals).

- **Terminological variants:** The overall definition is of a "compatibility." The V&V expert preferred to think of this as a "constraint," in keeping with the terminology of the database tool. Another example is the "?_any_value" term, which serves as a wildcard, indicating any acceptable parameter value may occur in the corresponding parameter position. Again, the V&V expert had the exact same concept, but preferred a different syntax.

- **Confirmation of shared understanding:** there were some areas of shared understanding, but these had to be confirmed, not taken for granted. A trivial example is "AND", which in the above is used to indicate that the constraint [compatibility] holds if all of the clauses of this AND hold. More interesting are the terms "meets" and "met-by," which are binary temporal relations between intervals, drawn from the work by Allen [Allen 1983].

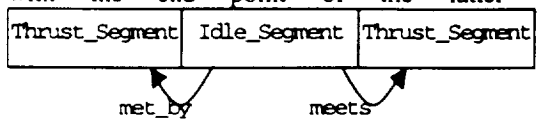
The net result was that the V&V expert required an intensive session of coaching on the meaning of the planner notations (plans and constraint language) at the start of this pilot study, and incremental assistance at various points throughout. Overall this did not amount to an undue consumption of planner experts' time.

Example of Translation from Planner Constraint to Database Query

Consider the Idle_Segment constraint given earlier. Its essential core is the following:

```
(SINGLE ((SEP_Schedule ...(Idle_Segment))
:compatibility_spec
(AND
 (meets (SINGLE ((SEP_Schedule ...
                 (Thrust_Segment (?,?)))
 (met_by (SINGLE ((SEP_Schedule ...
                 (Thrust_Segment (?,?))))
```

The fragments (SINGLE ((SEP_Schedule ... introduce descriptions that are to match to activities of the SEP scheduled in the plan. The first such description is of an Idle_Segment activity. For every instance of an activity in the plan matching that description, the constraint requires that the logical condition (AND ...) is true. The logical condition is the conjunct of two clauses. The first says that the matching instance meets a Thrust_Segment activity, i.e., the end-point of the Idle_Segment activity exactly coincides with the start point of some Thrust_Segment also in the plan. The second says that the matching instance is met_by a Thrust_Segment activity, i.e., the start point of the former exactly coincides with the end point of the latter Pictorially,



For translation, this is split into two separate constraints,

- All the DS-1 planner constraints take the overall form: for every activity-1 that matches description-1 there exists an activity-2 that matches description-2. A constraint of this form is *trivially* satisfied if the plan contains no activities matching description-1. The planchecker separates trivial and non-trivial cases in its reports of constraint satisfaction.

- The DS-1 planner generates plans for a segment of the entire mission (e.g., one week). Thus a plan is bounded within some "horizon"– it has a start and an end. Yet, the constraints may extend across this planning horizon. Such an instance is reported as a special kind of constraint satisfaction in which the plan satisfies the constraint within its horizon, but defers some residual checking for the next plan. The details of all such deferred checks are included within the planchecker's report.

- In an early version of the planner, a few of the constraints referenced information that is not stored in plans. In essence, this external information directed which one of several constraints is to apply. The planchecker's constraint translations handle these circumstances by checking each alternative. If all fail, it is an anomaly. If the plan is found to satisfy one of the alternatives, again, a special kind of constraint satisfaction is reported, which included the deduction of what the external information must be to direct the choice of the satisfied constraint.

The details are domain-specific, but we see a recurring need to make distinctions among classes of "pass" reports, and structure the analysis results accordingly.