How AI can help SE; or: Randomized Search Not Considered Harmful

Tim Menzies[1], Harhsinder Singh[2]

[1] Dept. Electrical & Computer Eng. University of British Columbia, 2356 Main Mall, Vancouver, B.C. Canada, V6T 1Z4 tim@menzies.com; http://tim.menzies.com
[2] Department of Statistics, West Virginia University, Morgantown, West Virginia, hsingh@stat.wvu.edu

Abstract. In fast-paced software projects, engineers don't have the time or the resources to build heavyweight complete descriptions of their software. The best they can do is lightweight incomplete descriptions which may contain missing and contradictory information. Reasoning about incomplete and contradictory knowledge is notoriously difficult. However, recent results from the empirical AI community suggest that *randomized search* can tame this difficult problem. In this article we demonstrate the the *relevance* and the *predictability* of randomized search for reasoning about lightweight models.

1 Introduction

Software engineering (SE) faces a dilemma which might be resolved by artificial intelligence (AI) research. However, before SE practitioners accept AI methods, they must be satisfied as to the *relevance* and the *predictability* of AI solutions.

The dilemma of current SE research is that much of that research is out of step with much of current industrial practice. At the recent International Symposium on Software Predictability (San Jose, California, 2000), a keynote address from Sun Microsystems shocked the researchers in the audience: few of the techniques endorsed by the SE research community are being used in fast-moving dot-com software companies. For such projects, developers and managers lack the resources to conduct *heavyweight software modeling*; e.g. the construction of complete descriptions of the business model[1] or the user requirements. Yet such heavyweight software modeling is very useful. Complete models of (e.g.) specifications can be used for a variety of tasks. For example, test cases could be auto-generated from the specification. Also, the consequences of conflicts between the requirements of different stakeholders could be studied. Further, we can automatically test that important temporal constraints hold over the lifetime of the execution of the specification. Lastly, model-based diagnosis could be used to localize errors.

To better support the fast pace of modern software, we need a new generation of *lightweight software modeling* tools. Lightweight software models can be built in

[1] For the the purposes of explaining this work to an SE audience, we will adopt widely used terminology. Hence, we will say business "model" when, strictly speaking, we should say business "theory".

a hurry and so are more suitable for the fast-moving software companies. However, software models built in a hurry can contain incomplete and contradictory knowledge. The presence of contradictions in the lightweight theories complicates the above useful tasks. Suppose some inference engine is trying to build a proof tree across a lightweight software model containing contradictions. Gabow et.al. [4] showed that building pathways across programs with contradictions is *NP-complete* for all but the simplest software models (a software model is very simple if it is very small, or it is a simple tree, or it has a dependency networks with out-degree ≤ 1). No fast and complete algorithm for NP-complete tasks has been discovered, despite decades of research.

Empirical results from AI offers new hope for the practicality of NP-complete tasks such as reasoning about lightweight software models. A repeated and robust empirical result (e.g. [1, 14]) is that theoretically slow NP-complete tasks are only truly slow in a narrow *phase transition* zone between under-constrained and over-constrained problems. Further, it has been shown empirically that in both the under/over-constrained zones, seemingly naive *randomized search algorithms* execute faster than, and nearly as completely, as traditional, slower, complete algorithms. Much of that research is based on conjunctive normal forms (e.g. [14]) but some evidence exists that the result holds also for horn-clause representations [9, 10]. These empirical results suggest that we might be able to implement the processing of lightweight software models using randomized search.

SE practitioners may well rebel at the prospect of applying randomized search to their applications. One issue is the *relevance problem*. With the exception of database programmers, it is not usual practice to view a (e.g.) "C" program as a declarative search space that can be explored this way or that way. Another issue is the *predictability problem*. Nondeterministic programs are usually not acceptable to an SE audience. For example, the SE guru Nancy Leveson clearly states "Nondeterminism is the enemy of reliability" [6]. If random search algorithms generate significantly different conclusions each time they run, then they would be unpredictable, uncertifiable, and unacceptable to the general SE community.

The goal of this article is to solve the relevance and predictability problems. §2 discusses the relevance problem and argues that declarative representations are common in SE, even when dealing with procedural programs. We will further argue that these declarative representations are compatible with *NAY0 graphs*: a directed, possibly cycle graph containing No-edges, And-nodes, Yes-edges, and Or-nodes. §3 discusses the predictability problem in the context of NAYO graphs. That discussion formalizes the predictability problem in terms of multiple world reasoning. If very different conclusions are found in the worlds of belief extracted from NAYO graphs, then we cannot predictably assert what conclusions hold. §4 builds and explores a mathematical model that predicts the likelihood of multiple worlds. This section concludes that randomized set-covering abduction, the odds of multiple worlds are very small. Hence, predictability is not a major concern.

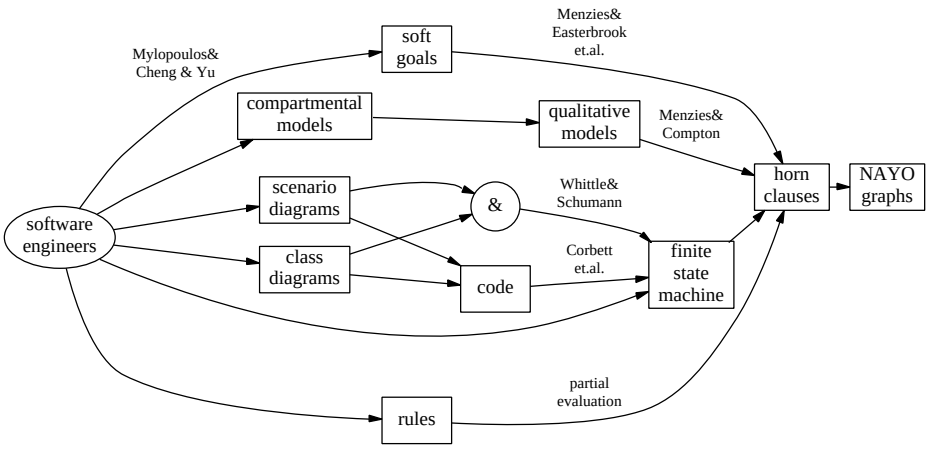

Fig. 1. Methods of generating NAYO graphs

2 The Relevance Problem

Figure 1 shows a variety of commonly used representations in SE. AI search is relevant to SE if these representations can be mapped into declarative representations. There are many examples of such a mapping in the literature, a sample of which is offered in this section.

Before beginning, we note that each of the mappings described potentially confound the predictability problem. Some information is lost when mapping down into low-level declarative representations. Typically, the information lose removes certain constraints which means that more inferences are possible in the abstracted form than in the non-abstracted form. Fortunately, in the next section, we show that the predictability problem is less of an issue that we might expect.

Common representations used in SE are object-oriented specification documents and procedural code. Whittle and Schumann have shown that specifications containing class diagrams and scenario diagrams can be automatically converted to finite state machines [15]. Also, Corbett et.al have shown that code written in some languages can be converted into finite state machines [3]. For example, the BANDERA system automatically extracts (slices) the minimum portions of a JAVA program's bytecodes which are relevant to proving particular properties models. These minimal portions are then converted into the finite state machine required for automatic formal analysis.

Before generating procedural code, software engineers may build requirement documents. Mylopoulos, Chung, and Yu express system requirements using an and-or structure called "soft goals" [12]. "Soft goals" have been mapped into horn-clause form by Menzies, Easterbrook, et.al. [9]. Horn clauses are a declarative representation that take the form

$$Goal\ if\ SubGoa11\ and\ SubGoal2\ and\ ...$$

which, in a Prolog notation, we would write as `goal :- subGoa11, subGoal2,...` If there exists more than one method of demonstrating some $Goal$, then each method is a separate clause.

Sometimes software engineers describe business rules in some rule-based language. These rules can be mapped into horn-clauses using standard partial evaluation techniques [13]. At other times, software engineers build discrete event simulations of their systems in some sort of compartmental modeling framework[2]. Menzies and Compton offered an declarative (abductive) semantics for executing incomplete compartmental models [8].

Finite state machines are a commonly used representation, particularly for real-time systems. Finite-state diagrams contain transitions between states. Transitions may be conditional on some guard. States may contain nested states. To translate state machines to horn-clauses, we create one variable for each state, then create one clause for each transition from state $S1$ to $S2$. Each clause will take the form `s2 :- s1, guard` where $guard$ comes from the conditional tests that activate that transition. If a state $S1$ contains sub-states $S1.a$, $S1.b$,... then create clauses of the form `s1a :- s1` and `s1b :- s1`, etc.

Horn-clauses can be easily reduced to NAYO graphs. A NAYO graph is a finite directed graph containing two types of edges and two types of nodes. *Or-nodes* store assignments of a single value to a variable. Only one of the parents of an or-node need be reached before we visit the or-node. *And-nodes* model multiple pre-conditions. All the parents of an and-node must be reached before this node is visited. *No-edges* represent illegal pairs of inferences; i.e. things we can't believe at the same time. For example, we would connect `diet(light)` and `diet(fatty)` with a no-edge. *Yes-edges* represent legal inferences between or-nodes and and-nodes. Figure 2 shows some sample horn clauses and its associated NAYO graph.

We focus on NAYOs for three reasons. Firstly, it is merely a graphical form a common representation: negation-free horn clauses. Secondly, and related to the first point, a range of representations can be expressed as NAYOs. Thirdly, there exist average case search results for NAYO graphs (see below). Any other representation with these three properties might be a suitable alternative framework for our analysis.

At first glance, it might appear that we can simply emulate the execution of a program by building proof trees across the NAYO graph. For example, we could mark some nodes as "inputs" then grow trees across the NAYO graph whose leaves are the inputs and whose root is some reached part of the program. However, we can't reckless grow proof trees across a NAYO: as a proof tree grows it should remain consistent (i.e. must not contain two nodes connected by a no-edge).

Proving a goal in a NAYO graph means recursively exploring all edges that arrive at that node. A randomized search would explore these edges in an order chosen randomly. HT0 is such a random search algorithm for NAYO graphs [10]. When HT0 reaches a literal, it retracts all other literals that might contradict this literal. The algorithm is

[2] Compartmental models utilize the principal of conservation of mass and assume that the sum of flows of substance in and out of a compartment must equal zero. Flows are typically modeled using a time-dependant exponential function since the rate of flow is often proportional to the amount of stuff in the compartment [7].

```
diet(fatty).
diet(light).
happy :- tranquillity(hi),
         ; rich,healthy.
healthy  :- diet(light).
satiated :- diet(fatty).
tranquillity(hi) :-
    satiated
    ; conscience(clear).
```

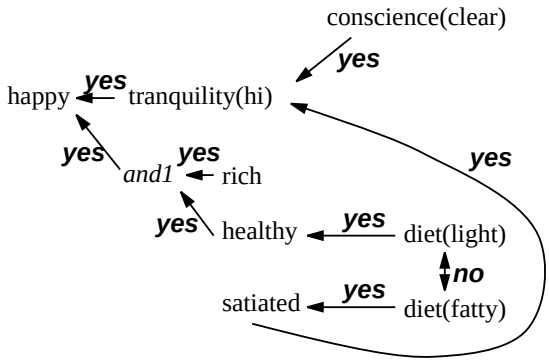

Fig. 2. Some ground horn clauses (left) converted to a NAYO graph (right).

very fast since it removes the Gabow et.al precondition for NP-completeness (any node that contradicts the nodes already in the proof tree). The random order in which HT0 explores the NAYO graphs selects which literals will be explored. Hence, HT0 repeats its processing several times. After trying to prove all it's goals in this random way, HT0 re-asserts the retracted literals and executes another "try" to prove all its goals. This process terminates when algorithm detects a plateau in the largest percentage of reachable goals found in any "try".

3 NAYO Graphs and Predictability

NAYO graphs offer a common declarative reading for a range of representations (e.g. those shown in Figure 1). At the NAYO level it is easy to show that the heuristic inferences made by random search may not be repeatable and hence not predictable.

Consider the three proofs HT0 might generate to prove `happy` in Figure 2.

$$Proof_1 \ : \ \texttt{happy} \leftarrow \texttt{tranquility(hi)} \leftarrow \texttt{conscience(clear)}$$
$$Proof_2 \ : \ \texttt{happy} \leftarrow \texttt{tranquility(hi)} \leftarrow \texttt{satieted} \leftarrow \texttt{diet(fatty)}$$
$$Proof_3 \ : \ \texttt{happy} \leftarrow \texttt{and1} \begin{cases} \leftarrow \texttt{rich} \\ \leftarrow \texttt{healthy} \leftarrow \texttt{diet(light)} \end{cases}$$

Some of these conclusions made by these proofs are not categorical conclusions. For example, our belief in `healthy` is contingent on accepting $Proof_3$ and not $Proof_2$ ($Proof_3$ is incompatible with $Proof_2$ since these two proofs require different diets). In the general case, a random search engine like HT0 will find only some subset of the possible proofs, particularly if it is run for a heuristically selected time interval. That is, a random search engine may not repeatedly realize that (e.g.) `healthy` is an uncertain conclusion.

Clearly for tiny systems like Figure 2 generating only a handful of proofs, the conclusions from random search are unpredictable and our SE colleagues are wise to reject

it. However, for such tiny systems, manual analysis will suffice. The automatic processing of NAYO graphs only gets interesting for larger systems. In such large systems, the goal nodes are a small subset of the total nodes. Further, as we show below, there emerges average case properties relating to our ability to quickly probe all the possible contingencies from a system. The sequel present these average case properties using the terminology of Menzies' prior work on set-covering abduction [8] (for notes on other abductive frameworks, see [2, 5]).

Given a model such as Figure 2 and a goal such as `happy`, HT0 builds proof trees to those goals; e.g. $Proof_1 \ldots Proof_3$. Anything that has not been asserted as a fact is an assumption. No proof can contain mutually exclusive assumptions or contradict the goal; i.e. assuming ¬`happy` is illegal. The generated proofs should be grouped together into maximal consistent subsets called *worlds*. Our example generates two worlds: $World_1 = \{Proof_1, Proof_3\}$ and $World_2 = \{Proof_1, Proof_2\}$.

A world contains what we can conclude from NAYO inference. A goal is proved if it can be found in a world. In terms of multiple world reasoning, the predictability problem can be formalized as follows:

> *Random search is unpredictable when it does not generate enough worlds to cover the range of possible conclusions.*

Note that this is a weak objection if it can be shown that the number of generated worlds is not large. This will be our argument below.

4 Average Number of Generated Worlds

Assumptions can be categorized into three important groups, only one of which determines how many worlds are generated. Some assumptions are *dependant* on other assumptions. For example, in $Proof_3$, the `healthy` assumptions depends fully on `diet(light)`. In terms of exploring all the effects of different assumptions, we can ignore the dependant assumptions. Another important category of assumptions are the assumptions that contradict no other assumptions. These *non-controversial* assumptions are never at odds with other assumptions and so do not effect the number of worlds generated. In our example, the non-controversial assumptions are everything except `diet(light)` and `diet(healthy)`. Hence, like the dependant assumptions, we will ignore these non-controversial assumptions. The remaining assumptions are the *controversial, non-dependant* assumptions or *funnel* assumptions. These funnel assumptions control how all the other assumptions are grouped into worlds of belief. DeKleer's key insight in the ATMS research was that a multi-world reasoning device need only focus on the funnel[3] When switching between worlds, all we need to resolve is which funnel assumptions we endorse. Continuing our example, if we endorse `diet(light)` then all the conclusions in $World_2$ follow and if we endorse `diet(healthy)` then all the conclusions in $World_1$ follow.

[3] DeKleer called the funnel assumptions the *minimal environments*. We do not adopt that terminology here since DeKleer used consistency-based abduction while we are exploring set-covering abduction here. For an excellent discussion that defines and distinguishes set-covering from consistency-based methods, see [2].

Proofs meet and clash in the funnel. If the size of the funnel is very small, then the number of possible clashes is very small and the number of possible resolutions to those clashes is very small. When the number of possible resolutions is very small, the number of possible worlds is very small and random search can quickly probe the different worlds of beliefs (since there are so few of them). Hence, if we can show that the average size of the funnel is small, then we can quickly poll the range of possible conclusions from our NAYO graphs.

There are numerous case studies suggesting that generating a few worlds (picked at random) adequately samples the space of possibilities that would be found after sampling a much larger number of worlds. Williams and Nayak found that a locally guides conflict resolution algorithm performed as well as the best available ATMS algorithm [16]. Menzies, Easterbrook et.al. report experiments comparing random world generation with full world generation. After millions of runs, they concluded that the random world generator found almost as many goals in less time as full world generation [9]. In other work, Menzies and Michael showed that the maximum percentage of reachable goals found by HT0 plateaus after a small number of tries [10]. These case studies are consistent with the claim that (1) the total number of worlds is usually very small, hence (2) average funnel size is not large. In order to test if this claim generalizes beyond these isolated case studies, we developed the following mathematical model [11]. Suppose some goal can be reached by a narrow funnel M or a wide funnel N as follows:

$$\left.\begin{array}{r} \xrightarrow{a_1} M_1 \\ \xrightarrow{a_2} M_2 \\ \cdots \\ \xrightarrow{a_m} M_m \end{array}\right\} \xrightarrow{c} goal_i \xleftarrow{d} \left\{\begin{array}{l} N_1 \xleftarrow{b_1} \\ N_2 \xleftarrow{b_2} \\ N_3 \xleftarrow{b_2} \\ N_4 \xleftarrow{b_2} \\ \cdots \\ N_n \xleftarrow{b_n} \end{array}\right.$$

Under what circumstances will the narrow funnel be favored over the wide funnel? More precisely, when are the odds of reaching $goal_i$ via the narrow funnel much greater than the odds of reaching $goal_i$ via the wide funnel? To answer this question, we begin with the following definitions. Let the M funnel use m variables and the N funnel use n variables. For comparison purposes, we express the size of the wider funnel as a ratio α of the narrower funnel; i.e. $n = \alpha m$. Each member of M is reached via a path with probability a_i while each member of N is reached via a path with probability b_i. Two paths exist from the funnels to this goal: one from the narrow neck with probability c and one from the wide neck with probability d. The probability of reaching the goal via the narrow pathway is $narrow = c\prod_{i=1}^{m} a_i$ while the probability of reaching the goal via the wide pathway is $wide = d\prod_{i=1}^{n} b_i$.

Assuming that the goal is reached, then there are three ways to do so. Firstly, we can reach the goal using both funnels with probability $narrow.wide$. Secondly, we can reach the goal using the narrow funnel and not the wider funnel with probability $narrow(1 - wide)$. Thirdly, we can reach the goal using the wider funnel and not the narrow funnel with probability $(1 - narrow)wide$. Let g be probability of reaching $goal_i$ which is the sum of the three probabilities; i.e. $g = narrow + wide - narrow.wide$.

Given the goal is reached, then the conditional probabilities of reaching the $goal_i$ via two our funnels is:

$$P\left(narrow|g\right) = \frac{narrow}{narrow + wide - narrow.wide}$$
$$P\left(wide|g\right) = \frac{wide}{narrow + wide - narrow.wide}$$

Let R be the ratio of the odds[4] of these conditional probabilities. Our pre-condition for use of the narrow funnel is $R > 1$. More generally, using the narrow funnel is much more likely if R is bigger than some threshold value t:

$$\left(R = \frac{\left(narrow\right)^2\left(1 - wide\right)}{\left(wide\right)^2\left(1 - narrow\right)}\right) > t \tag{1}$$

4.1 Assuming Uniform Distributions

Assuming that a_i and b_i come from uniform probability distributions, then $\sum_{i=1}^{m} a_i = 1$ and $a_i = \frac{1}{m}$, so $narrow = c\left(\frac{1}{m}\right)^m$. Similarly, under the same assumptions, $wide = d\left(\frac{1}{n}\right)^n$. Thus, by Equation 1 when $t = 1$, narrow funnels are more likely when:

$$narrow^2(1 - wide) > wide^2(1 - narrow)$$

which we can rearrange to: $(narrow - wide)(narrow + wide - narrow.wide) > 0$. This expression contains two terms, the second of which is always positive. Hence, this expression is positive when $\frac{narrow}{wide} > 1$. We can expand this expression to:

$$\frac{narrow}{wide} = \frac{c\left(\frac{1}{m}\right)^m}{d\left(\frac{1}{n}\right)^n}$$

Recalling that $n = \alpha m$, this expression becomes $(\alpha m)^{\alpha m} m^{-m} > \frac{d}{c}$

Consider the case of two funnels, one twice as big as the other; i.e. $\alpha = 2$. This expression can then be rearranged to show that $\frac{narrow}{wide} > 1$ is true when

$$(4m)^m > \frac{d}{c} \tag{2}$$

At $m = 2$, Equation 2 becomes $d < 64c$. That is, to access $goal_i$ from the wider funnel, the pathway d must be 64 times more likely than the pathway c. This is not highly likely and this becomes less likely as the narrower funnel grows. By the same reasoning, at $m = 3$, to access $goal_i$ from the wider funnel, the pathway d must be 1728 times more likely than the narrower pathway c. That is, under the assumptions of this uniform case, as the wide funnel gets wider, it becomes less and less likely that it will be used.

[4] The odds of an event with probability $P(x)$ is the ratio of the probability that the event does/does not happen; i.e. $\frac{P(X)}{1-P(X)}$

4.2 Assuming Non-Uniform Distributions

To explore the case where $\sum_{i=1}^{m} a_i \neq 1$ and $\sum_{i=1}^{m} b_i \neq 1$ (i.e. the non-uniform probability distribution case), we created and executed a small simulator many times. The mean μ and standard deviation σ^2 of the logarithm of the variables a_i, b_i, c, d were picked at random from the following ranges: $\mu \in \{1, 2, \ldots 10\}$;
$spread \in \{0.05, 0.1, 0.2, 0.4, 0.8\}$. μ and $spread$ where then converted into probability as follows: $\sigma^2 = spread * \mu$; $probability = 10^{-1 * normDist(\mu, \sigma^2)}$. Next, m and α were picked at random from the ranges: $m \in \{1, 2, \ldots 10\}$; $\alpha \in \{1, 1.25, 1.5, \ldots 10\}$. R was then calculated and the number of times R exceeded different values for t is shown in Figure 3. As might be expected, at $t = 1, \alpha = 1$ the funnels are the same size and the odds of using one of them is 50%. As α increases, then increasingly Equation 1 is satisfied and the narrower funnel will be preferred to the wider funnel. The effect is quite pronounced. For example, at $\alpha = 3$, 82% of our simulated runs, random search will be 10,000,000,000 times more likely to favor narrow funnels $\frac{1}{3}$ the size of alternative funnels.

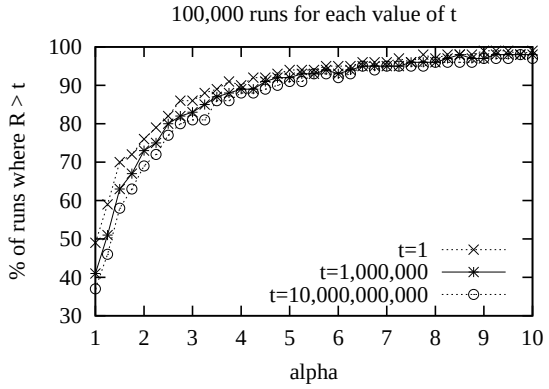

Fig. 3. 100000 runs of the funnel simulator. Y-axis shows what percentage of the runs satisfies Equation 1.

In summary, in both the uniform and non-uniform case, random search engines such as HT0 will favor worlds with narrow funnels. Since narrow funnels mean fewer worlds, we can now assure our SE colleagues that it is highly likely that random search will sample the entire space of possible conclusions.

5 Conclusion

Modern SE research urgently needs to address the issue of lightweight modeling in order to support current industrial practices. A central problem with the processing of lightweight models is that they are incomplete and contain contradictions. AI research has been exploring theories containing contradictions for decades. Random search is an AI technique that can explore very large models, even when they contain contradictions.

Before the SE community accepts random search, it must be shown that these techniques are relevant and predictable. We have shown that a wide range of SE artifacts can be mapped into a declarative representation called NAYO graphs. We have also shown that after the randomized generation of a small number of worlds from the NAYO

graphs, it is unlikely that very different goals will be reachable if we randomly generated many more worlds. Hence, we assert that (1) random search is both relevant and surprisingly predictable; and (2) SE can use random search to support the lightweight modeling tools needed for the current fast pace of software development.

References

1. P. Cheeseman, B. Kanefsky, and W. Taylor. Where the really hard problems are. In *Proceedings of IJCAI-91*, pages 331–337, 1991.
2. L. Console and P. Torasso. A Spectrum of Definitions of Model-Based Diagnosis. *Computational Intelligence*, 7:133–141, 3 1991.
3. J. Corbett, M. Dwyer, J. Hatcliff, S. Laubach, C. Pasarenu, Robby, and H. Zheng. Bandera: Extracting finite-state models from java source code. In *Proceedings ICSE2000, Limerick, Ireland*, pages 439–448, 2000.
4. H. Gabow, S. Maheshwari, and L. Osterweil. On two problems in the generation of program test paths. *IEEE Trans. Software Engrg*, SE-2:227–231, 1976.
5. A. Kakas, R. Kowalski, and F. Toni. The role of abduction in logic programming. In C. H. D.M. Gabbay and J. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming 5*, pages 235–324. Oxford University Press, 1998.
6. N. Leveson. *Safeware System Safety And Computers*. Addison-Wesley, 1995.
7. J. McIntosh and R. McIntosh. *Mathematical Modeling and Computers in Endocrinology*. Springer-Verlag, 1980.
8. T. Menzies and P. Compton. Applications of abduction: Hypothesis testing of neuroendocrinological qualitative compartmental models. *Artificial Intelligence in Medicine*, 10:145–175, 1997. Available from `http://tim.menzies.com/pdf/96aim.pdf`.
9. T. Menzies, S. Easterbrook, B. Nuseibeh, and S. Waugh. An empirical investigation of multiple viewpoint reasoning in requirements engineering. In *RE '99*, 1999. Available from `http://tim.menzies.com/pdf/99re.pdf`.
10. T. Menzies and C. Michael. Fewer slices of pie: Optimising mutation testing via abduction. In *SEKE '99, June 17-19, Kaiserslautern, Germany. Available from `http://tim.menzies.com/pdf/99seke.pdf`*, 1999.
11. T. Menzies and H. Singh. Many maybes mean (mostly) the same thing. In *2nd International Workshop on Soft Computing applied to Software Engineering (Netherlands), February*, 2001. Available from `http://tim.menzies.com/pdf/00maybe.pdf`.
12. J. Mylopoulos, L. Cheng, and E. Yu. From object-oriented to goal-oriented requirements analysis. *Communications of the ACM*, 42(1):31–37, January 1999.
13. D. Sahlin. *An Automatic Partial Evaluator for Full Prolog*. PhD thesis, The Royal Institute of Technology (KTH), Stockholm, Sweden, May 1991. Available from `file://sics.se/pub/isl/papers/dan-sahlin-thesis.ps.gz`.
14. B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *AAAI '92*, pages 440–446, 1992.
15. J. Whittle and J. Schumann. Generating statechart designs from scenarios. In *Proceedings of the 22nd International Conference on Software Engineering (ICSE). Limerick, Ireland*, June 2000. Available from `http://www.riacs.edu/research/detail/ase/icse2000.ps.gz`.
16. B. Williams and P. Nayak. A model-based approach to reactive self-configuring systems. In *Proceedings, AAAI '96*, pages 971–978, 1996.