SNePS Efficiency Report*
SNeRG Technical Note 43

A. Patrice Seyed, Michael Kandefer, and Stuart C. Shapiro
Department of Computer Science and Engineering
and Center for Cognitive Science
and National Center for Multisource Information Fusion
State University of New York at Buffalo
{aliseyed|mwk3|shapiro}@cse.buffalo.edu

1 Introduction

This paper contains the results of various timing tests on the SNePS 2.7.0 knowledge representation and reasoning system (Shapiro & The SNePS Implementation Group 2008) when various queries are performed on different knowledge bases (KBs). Then, in §4, modifications are discussed that were made to SNePS 2.7.0, resulting in SNePS 2.7.1. All tests were then run on SNePS 2.7.1, with the results reported in §5. Other modifications motivated by studying these tests are being scheduled for inclusions in SNePS 3, as reported in §4.4.

SNePS is implemented in Common Lisp. All tests were performed using the SNePSLOG interface to SNePS and a fresh Lisp image between runs. In each test, a SNePSLOG query was issued for varying numbers of propositions in the KB. The times required to answer each query was obtained by the Lisp function `time`. The tests were performed on `nickelback.cse.buffalo.edu`, a Sun Sunfire X4200 running RedHat Enterprise Linux 4 (64-bit) with 8.0 GB of main memory and two Dual Core AMD Opteron^{TM} Processor 285s, clocked at 2592 MHz, running RedHat Enterprise Linux 4 (64-bit). Allegro ACL 8.1 Enterprise Edition was used for the tests with optimization settings: safety 1, speed 3, space 0, and debug 0.

A query is expressed as a SNePSLOG well-formed formula (wff) followed by a question mark. The wff may contain free variables (expressed as a symbol preceded by a question mark, such as `?x`) or may be ground. The following steps are taken to answer the query.

1. An internal representation is created for the wff. We will refer to the internal representation of a wff as a "node." SNePS guarantees that there is at most one node for any wff. So if there is already a node for the wff, it is found. (This process of finding the node for a wff, or, if necessary, creating a new node is performed by the function `find-or-build`).

2. If the wff was ground, its node might already be asserted or negated in the KB. If so, that asserted or negated node is returned, and printed as a SNePSLOG wff.

3. If the wff was ground, but its node was neither asserted nor negated, or if the wff contained free variables, the wff's node is **match**ed against other nodes in the KB using a version of unification.[1]

4. The result of a match of a node n is a set of triples $\langle m, \sigma, \tau \rangle$, such that m is a node, σ and τ are substitutions, and $m\sigma \vdash n\tau$. If $m\sigma$ is asserted in the KB, $n\tau\sigma$ is an answer to the query.

*This work was supported in part by CUBRC under prime contract FA8750-06-C-0184 between CUBRC and U.S. Air Force Research Laboratory, Rome, NY.

[1] An implementation of (Escalada-Imaz & Ghallab 1988) modified by the Unique Variable Binding Rule (Shapiro 1986) is used.

5. If $m\sigma$ is not asserted, but is in consequent position of an asserted rule whose antecedent is p, then $p\sigma$ is treated as a subquery in a manner similar to the original query.

2 Methods

Examples of this process, and of some more complicated variations will be given in the following sections. In this paper the term 'test' denotes the type of test, numbered 1 through 11, the term 'trial' denotes a test with a specific number of extraneous propositions (discussed below), and the term 'run' denotes individual trials executed for a test, which is particularly relevant to tests with trial times averaged. Each trial for a test includes assertions of `n` extraneous propositions, to determine its effect on a respective rule and query. The value of `n` across the trials includes 0, 5, 10, 50, 100, 500, 1,000, 2,500, 4,000, 5,500, 7,000, and 8,500 extraneous propositions. Previously, the set of trials did not include runs with extraneous propositions between 5,000 and 10,000, but was incorporated to provide more conclusive data for tests that failed due to memory limitations after a trial with 5,000 extraneous propositions. Note, the upper limit on SNePS propositions is dependent on the Lisp image's heap size, since a Lisp image allocated a one gigabyte Lisp heap was able assert up to 20,000 arity two predicates successfully into SNePS, while a Lisp image allocated the default Lisp heap size (17.5 megabytes) could not assert 10,000.

In addition, time calculations were averaged over three trials for Test 1, 2, 3, 4, 6, 8, and 10. This strategy was added to normalize the data, and was particularly motivated by the observation of drastic fluctuations in Garbage Collection (GC) time in test 6 and 8. Test 5, 7, and 9 included only one trial due to the extensive time for running these tests' query with at least $2,500$ extraneous propositions.

3 Tests and Results

3.1 Test 1

Test 1 was performed to see if the size of the KB has any effect on answering a fully ground query which is already asserted in the KB. In this case the KB was populated with a single proposition `Isa(n1,Network)`, and n extraneous propositions of the form `Isa(a,b)`, where `a` and `b` were random, unique individual constants. n varied between 0 and $8,500$.The query was `Q = Isa(n1,Network)?`. The results of this test are shown in Figure 1.

In this case, all the work was done by `find-or-build`—as soon as the node for the query was found, it was found to be asserted, and the answer was returned. Figure 1 reflects that the size of the KB has minimal effect on `find-or-build`; only the trial with $8,500$ extraneous propositions required at least 1 ms.

3.2 Test 2

Test 2 was performed to see if the size of the KB had any effect on answering a query with one free variable. In this case the KB was populated with the single assertion,[2] `Isa(n1,Network)`, and n extraneous assertions of the form `Isa(a,b)`, where `a` and `b` were random, unique individual constants. n varied between 0 and $8,500$. The query was `Q = Isa(n1,?y)?`, where `?y` is a free variable. The results of this test are shown in Figure 2.

In this test, `find-or-build` did not find an extant node for `Q`, and so created one with `Isa` as the predicate, `n1` as the first argument, and a free variable `?y` as the second argument. When matched against all the nodes in the KB except base nodes (individual constants), n nodes were found with predicate `Isa`, n nodes with a second argument that can unify with `?y`, but only one node with `n1` as first argument. Since that node was asserted, `Q`$\{$`Network/?y`$\}$ was returned as the answer. The graph illustrates incremental increase in query time, following the trial with 100 extraneous propositions, which is the effect of a simple match.

[2] We will use the term "assertion" for an asserted proposition.

3.3 Test 3

Test 3 was performed to see if the size of the KB had any effect on answering a query with two variables. In this case the KB was populated with *n* assertions of the form `Isa(a,b)`, where `a` and `b` are random, unique terms, and n varied between 0 and $8,500$. The query was `Q = Isa(?x,?y)?`, where `?x` and `?y` are free variables. The results of this test are shown in Figure 3.

In this test, the query node was matched against the same number of assertions as in the second test, but succeeded with all n of them instead of with just one.

The additional computational expense of variable subsitution in unification with two variables as opposed to one variable is reflected in the results. The supporting graph indicates increase in query time in trials with over 500 extraneous propositions, which is more significant in trials with over $1,000$ extraneous propositions.

At $4,000$, $5,500$, $7,000$, and $8,500$ extraneous propositions test 3's total time was two, three, four, and five times the total time of test 2, respectively. These results indicate the additional cost of unifying two variables with the KB versus just one variable, and successfully matching with n propositions versus just one. Printing n propositions to standard output is a factor in the time cost as well. Results like these and the previous suggest an improvement to unification could increase performance in large scale KBs, being a primary mechanism of the inference engine.

3.4 Test 4

Test 4 and 5 were performed to see the effect of the increase in the number of assertions on a simple one-step transitivity rule. In test 4 the KB was populated with the assertions `Isa(n1,Network)`, `Ako(Network,Entity)`, and `all(x,y)(Ako(x,y) => all(z)(Isa(z,x) => Isa(z,y)))`. The KB also contained n assertions of the form `Isa(a,b)`, where `a` and `b` are random, unique terms, and n varied between 0 and $8,500$. The query was `Q = Isa(n1,?w)?`, where `?w` is a free variable. The results of this test are shown in Figure 4.

In this test, the query `Isa(n1,?w)?` matches the assertion `Isa(n1,Network)`, producing the answer `Isa(n1,?w)`$\{$`Network/?w`$\} =$ `Isa(n1,Network)`, and the antecedent and consequent propositions `Isa(z,x)` and `Isa(z,y)`. The latter is in the consequent position, so the rule instance `(Isa(z,x) => Isa(z,y))`$\{$`n1/z`$\}$ would be relevant for answering the query if it were asserted. However it isn't, but it is in the consequent of the rule instance `all(x,y)(Ako(x,y) => (Isa(n1,x) => Isa(n1,y)))`. Therefore `Ako(x,y)` is generated as a subgoal, where `x` and `y` are free variables.

`Ako(x,y)`, a two-variable query as in the test 3, is matched against all the non-base nodes in the KB and matches `Ako(Network,Entity)` with $\tau =$ $\{$`Network/x, Entity/y`$\}$. SNePS now asserts the rule `all(z)(Isa(z,Network) => Isa(z,Entity))` in the KB as a lemma (Choi & Shapiro 1991; Choi 1993), and generates the subgoal `Isa(n1,Network)`, a fully ground query as was used in the first test. That proposition is found already to be asserted, and so the rule fires, producing as a second answer to the original query `Isa(n1,?w)`$\{$`Entity/?w`$\} =$ `Isa(n1,Entity)`.

In summary, this test required: one one-variable query that matched one assertion; one two-variable query that matched one assertion (Remember that in the test 3, the two-variable query matched n assertions.); one ground query that was found to be asserted; and the assertion of one new rule as a lemma.

The results indicate an increase of 25 to 30 msec between tests with $2,500$ and $4,000$, $4,000$ and $5,500$, and $7,000$ and $8,500$ extraneous propositions, respectively, with a minimal increase of approximately 14 msec between tests with $5,500$ and $7,000$ extraneous propositions. Later, this test will be compared with test 6 and 8 which have the same `Isa` predicate for extraneous propositions but with a different logical form that accomplishes the same rule firing goal.

3.5 Test 5

In test 5 the KB was populated with the assertions `Isa(n1,Network)`, `Ako(Network,Entity)`, and `all(x,y)(Ako(x,y) => all(z)(Isa(z,x) => Isa(z,y)))`. The KB also contained n assertions of the form form `Ako(a,b)`, where `a` and `b` are random, unique terms, and n varied between 0 and 8,500. The query was `Q = Isa(n1,?w)?`, where `?w` is a free variable. The results of this test are shown in Figure 5.

The results indicate a more than exponential increase in time it takes SNePS to answer the same query as in test 4 for the trial including 50 extraneous propositons forward. It was also discovered that during this test the underlying

Lisp process reached an out-of-memory condition beginning with the trial with $5,500$ propositions forward.

The difference is that in test 4, the extraneous propositions matched the antecedent of the inner implication, whereas in the current test, the extra assertions matched the antecedent of the outer implication. Thus, when `Ako(x,y)` is generated as a subgoal, it matches only one assertion in test 4, resulting in one new rule, but matches $n+1$ assertions in test 5, resulting in $n+1$ new rules, `all(z)(Isa(z,Network) => Isa(z,Entity))` plus n rules of the form `all(z)(Isa(z,a) => Isa(z,b))`. Then, whereas in test 4, one additional query (`Isa(n1,Network)`) was generated, in test 5, $n+1$ additional queries were generated, (`Isa(n1,Network)` plus n queries of the form `Isa(n1,a)`).

Although the rules `(A => (B => C))` and `(B => (A => C))` are logically equivalent, the more efficient rule is the one whose outer-most antecedent matches the fewer assertions.

3.6 Test 6

Test 6 and 7 were done to see the effect of making the scope of `z` the entire rule instead of only the inner rule. In the sixth test the KB was populated with the assertions `Isa(n1,Network)`, `Ako(Network,Entity)`, and `all(x,y,z)(Ako(x,y) => (Isa(z,x) => Isa(z,y)))`. The KB also contained n assertions of the form `Isa(a,b)` where `a` and `b` are random unique terms, and n varied between 0 and $8,500$. The query was `Q = Isa(n1,?w)?` where `?w` is a free variable. The results of this test are shown in Figure 6.

The results show that the performance of test 6 is very similiar to the performance of test 4, except there is a spike in GC time at trials for 1,000, 4,000 and 7,000 extraneous propositions. This trend is illustrated in Figure 6, which skrews the time range of the overall chart. (Note: the GC time is 0 ms for all runs in test 4.) These findings reveal that during this test GC time is sporatically necessary for operations of the query given this form of the logic rule in question.

The primary difference of the current test and the test 4 is that for the current test when `Ako(x,y)` is generated as a subgoal and matches `Ako(Network,Entity)` with the unifier $\{$`Network/x,Entity/y`$\}$, SNePS asserts the rule `Isa(n1,Network) => Isa(n1,Entity)`. In the fourth test the rule `all(z)(Isa(z,Network) => Isa(z,Entity))` is generated because `z` is bound to innermost rule, whereas in the current test it is bound in the outermost rule. Rules in the form of this innermost rule are generated for each `Ako` predicated proposition for test 4 and 5. Aside from the GC time difference, there is only slight time difference for test 4 and 6, and this is attributable to there being only one `Ako` predicated proposition.

3.7 Test 7

In test 7 the KB was populated with the assertions `Isa(n1,Network)` and `Ako(Network,Entity)`, and `all(x,y,z)(Ako(x,y) => (Isa(z,x) => Isa(z,y)))`. `n` assertions of the form `Ako(a,b)` where `a` and `b` are random unique terms also populated the KB, where `n` varied between 0 and 8,500. The query was `Q = Isa(n1,?w)?` where `?w` is a free variable. The results of this test are shown in Figure 7.

In comparison to test 5, the current test had significantly less total time, which became evident for the trials with 100 extraneous propositions forward. In comparing the two tests, over the trials including $2,500$, $4,000$, $5,500$, $7,000$, and $8,500$ extraneous propositions, the test 7 was approximately two times, four times, five times, and twelve times as fast as the fifth test, respectively. Additionally, whereas test 5 reached an out-of-memory condition during the trial with $5,500$ extraneous propositions, the current test reached the condition during the trial with $7,000$ extraneous propositions.

The reason for this performance improvement is clear. Since in the current test the *z* variable is bound to the outermost antecedent, when `Ako(x,y)` becomes a subgoal and is unified with all non-base nodes, a rule is generated where *x* and *y* are the ground terms from the match, and *z* is a ground term for all matches of `Isa(z,x)`. Therefore, the newly generated rules are fully ground and require less computation time when the subgoal of the antecedent of each newly generated rule becomes a subgoal to satisfy the original query. When a query contains a fully ground proposition SNePS only needs to call `find-or-build` instead of embarking on the more computationally expensive process of unifying with all non-base nodes.

Still, in comparision to the test 6, significant degradation in performance was observed. This was because in the current test for every proposition `Ako(a,b)`, a rule was generated in the form `Ako(a,b) => (Isa(n1,a) =>`

Isa(n1,b)). In this test this includes every extraneous proposition and the proposition Ako(Network,Entity). For test 6 this was only necessary for the one proposition, Ako(Network,Entity).

3.8 Test 8

Test 8 and 9 were done to see the effect of the previously tested rule without rule nesting by including Ako(y,z) and Isa(x,y) as a conjunction in the antecedent and Isa(x,z) as the following: all(x,y,z) ({Ako(y,z),Isa(x,y)} &=> Isa(x,z)).

In test 8 the KB was populated with the assertions Isa(n1,Network), Ako(Network,Entity), and all(x,y,z)({Isa(x,y),Ako(y,z)} &=> Isa(x,z)). *n* assertions of the form Isa(a,b), where a and b are random, unique terms also populated the KB. *n* was given the values between the ranges of 0 and 8,500. The query issued was Q = Isa(Network,?y)?, where ?y is a variable in SNePSLOG. The results of this test are shown in Figure 8.

The graph indicates the current test performed worse through all trials than test 4. In comparision to test 6 the current test performed worse through all trials except those with $1,000$ and $7,000$ extraneous propositions. The performance comparison between test 6 and test 8 is inconclusive, and test 4 performs best of all three. The reason for this is because the rule of the form all(z)(Isa(z,a) => Isa(z,b)) is generated for each Ako(a,b) proposition. This type of intermediate rule with one free variable allows for less expense of matching, where in test 6 and 8 all of the variables are bound to the outermost antecedent, losing the gains of the intermediate rule and adding the expense of matching 3 free variables over 2 propositions in the antecedent of the given test's rule.

3.9 Test 9

In test 9 the KB was populated with the assertions Isa(n1,Network), Ako(Network,Entity), and all(x,y,z)({Isa(x,y),Ako(y,z)} =&> Isa(x,z)). *n* assertions of the form Ako(a,b), where a and b are random, unique terms also populated the KB. *n* was given the values between the ranges of 0 and 8,500. The query issued was Q = Isa(Network,?y)?, where ?y is a variable in SNePSLOG. The results of this test are shown in Figure 9.

This test was on par with the test 8 in terms of speed, with a significant increase in computation time after the trial with 500 extraneous propositions. Since Isa(n1,?y) is the query proposition in both tests unification is performed initially on this proposition. Since there is only one assertion in the KB with Isa as the predicate and n1 as the term in the first argument position, the cost of this test is equal in both tests. Once the substitution is returned by SNePS, the subgoal of Ako(x,y) is initiated as a query proposition. In the current test this step is repeated for every Ako proposition given that both arguments are free, and this is the primary reason for the additional time cost after 500 propositions.

The current test performs better than test 7, and better than test 5 to a greater degree. It also does not reach the out-of-memory condition until the trial with $8,500$ propositions. The reason for performance improvement when compared to test 7, which performs better than test 5 overall, is that in that test for every extraneous Ako(a,b) proposition generated, given the assertion Isa(n1,Network) in the KB, the intermediate rule Isa(n1,a) => Isa(n1,b) is generated, where a and b are ground terms. The current test avoids the overhead of generating these extraneous intermediate rules that are unified with and will never fire. There is no singular Ako predicated outer antecedent proposition, but rather a conjunction of Ako(y,z) and Isa(x,y), which fires when the variables x, y, and z are successfully substituted for during unification.

Table 1 contains a summary of tests 4–9. The columns show: the test number; the form of the rule; the form of the extraneous propositions; how many milliseconds it took to run with $n = 8,500$ extraneous propositions, and if 8,500 extraneous propositions could not fit in memory the largest value of n that was successful.

3.10 Test 10

Test 10 is similar to the second, and done to see if placing the extraneous assertions in a set, rather than as individual assertions would have an effect on query speed. In this case the KB was populated with a single proposition Isa(n1,Network), and one proposition of the form Isa(x,y) was asserted, where *x* was a random, unique term,

Table 1: Summary of tests involving nested rules.

Test	Rule	n Assertions	$n = 8,500$/error cond.
4.	`all(x,y)(Ako(x,y) => all(z)(Isa(z,x) => Isa(z,y)))`	`Isa(a,b)`	136.6
5.	`all(x,y)(Ako(x,y) => all(z)(Isa(z,x) => Isa(z,y)))`	`Ako(a,b)`	1,102,380/n=5500
6.	`all(x,y,z)(Ako(x,y) => (Isa(z,x) => Isa(z,y)))`	`Isa(a,b)`	110
7.	`all(x,y,z)(Ako(x,y) => (Isa(z,x) => Isa(z,y)))`	`Ako(a,b)`	2,203,990/n=7000
8.	`all(x,y,z)({Isa(x,y),Ako(y,z)} &=> Isa(x,z))`	`Isa(a,b)`	166.6
9.	`all(x,y,z)({Isa(x,y),Ako(y,z)} &=> Isa(x,z))`	`Ako(a,b)`	75870/n=8500

and `y` was a set consisting of *n* random, unique terms. *n* was given the values between the ranges of 0 and 100,000. The query issued was `Isa(n1,?y)?`. The results of this test are shown in Figure 10.

Unlike in the second test, the size of the set had no observable effect on the query's time. Having the terms for the second argument in a set rather than have a separate proposition for each term in that set streamlines unification by requiring one matching attempt as opposed to *n* matching attempts. Additionally, placing the extraneous terms in a set allowed for more extraneous terms to be asserted into the KB than in other tests, at around 90,000, . The likely reason for this is that fewer arcs and nodes were needed to represent this KB in SNePS than in previous tests. In addition, fewer assertions were made (only two), resulting in the creation of fewer extensional contexts.

3.11 Firewall Rules Inference Test

We have used SNePS to detect false positives for potential cyber attacks (Kandefer *et al.* 2007). The crucial information regarding an attack is collected from a notification system, and includes the source ip address, target ip address, signature of the vulnerability, the signature ID from the snort database, the target port number, and the protocol (e.g. tcp, udp, icmp). Five cases of false positives are tested for, which include (in order of case checking): "no target host found", "exposure signature does not reference an exposure in the KB", "exposure does not correspond to a vulnerability on the target ip address", "the signature ID does not correspond to a vulnerability on the target ip address", and "source ip address and target ip address are not connected". In service of the last case, the following rule was developed to infer instances of direct connection of two network devices, using a particular port and protocol, from the data provided in the input files.

```
all(npl,sid,did)(NeighborPermissionList(npl,sid,did)
          => all(n1,ip1)(NetworkDevice(n1,sid,ip1)
              => all(n2,ip2)(NetworkDevice(n2,did,ip2)
                   => all(po,pn,pp)(Port(po,pn,pp)
                        => (PartOf(po,npl)
                             => ConnectedByPort(n1,n2,pn,pp)))))).
```

The query performed to test the efficency of this rule is

```
ConnectedByPort("192.168.1.2","100.10.20.9",?x,?y)?.
```

This query is typically made with the first two arguments as ground terms representing the source and target ip address.

It was observed in a prior tests with `Ako` predicated extraneous propositions that scoping all variables in the outermost antecedent reduced the number of intermediate rules and overall time of computation. Therefore, the first attempt to make the firewall inference rule more efficient is to apply this technique. The rewrite of the rule takes the following form:

```
all(npl,sid,did,n1,ip1,n2,ip2,po,pn,pp)(NeighborPermissionList(npl,sid,did)
      => (NetworkDevice(n1,sid,ip1)
           => (NetworkDevice(n2,did,ip2)
                => (Port(po,pn,pp)
                     => (PartOf(po,npl)
                          => ConnectedByPort(n1,n2,pn,pp))))))).
```

As observable in Figure 11, where 1 along the x-axis represents testing the original rule and 2 represents testing the first rewrite above, the change in variable scoping provided a drastic improvement in efficiency of the query.

It was also observed that for the original rule during the query `NetworkDevice("192.168.1.2",?x,?y)` all three variable are free for the outermost antecedent `NeighborPermissionList(?x,?y,?z)`. Therefore, the number of intermediate rules is initially based on the number of `NeighborPermissionList` propositions in the KB, which in this case is 48. If the two `NetworkDevice` propositions were the outermost antecedents the number of intermediate rules would be initially based on the number of propositions that unify with `NetworkDevice("192.168.1.2",?x,?y)` and `NetworkDevice("100.10.20.9",?x,?y)` given the query `ConnectedByPort("192.168.1.2","100.10.20.9",?x,?y)?`, which in this case is 2. Thus, for the second attempt to make the firewall inference rule more efficient, the `NeighborPermissionList` proposition is moved as the innermost antecedent and the two `NetworkDevice` propositions become the outermost as follows:

```
all(n1,sid,ip1)(NetworkDevice(n1,sid,ip1)
      => all(n2,did,ip2)(NetworkDevice(n2,did,ip2)
           => all(po,pn,pp)(Port(po,pn,pp)
                => all(npl)(PartOf(po,npl)
                     => (NeighborPermissionList(npl,sid,did)
                          => ConnectedByPort(n1,n2,pn,pp))))))).
```

As observable in Figure 11, where 3 along the x-axis represents the second rewrite above, this rule and the respective query performs drastically better than the original rule and incrementally better than the first rewrite in terms of efficiency improvement having used outermost antecedent variable scoping for all variables.

The third rewrite includes a combination of both strategies to create the following inference rule:

```
all(npl,sid,did,n1,ip1,n2,ip2,po,pn,pp)(NetworkDevice(n1,sid,ip1)
      => (NetworkDevice(n2,did,ip2)
           => (Port(po,pn,pp)
                => (PartOf(po,npl)
                     => (NeighborPermissionList(npl,sid,did)
                          => ConnectedByPort(n1,n2,pn,pp))))))).
```

represented as 4 along the x-axis and outperforms all other forms inference rule. It is clear from this set of tests that outermost antecedent variable scoping and choosing the most constrained propositions as the outer antecedent are two choices that can make the inference engine more efficient. Overall, awareness of the KB content, query predicate and arguments are all important in inference rule planning.

4 Modifications To Improve SNePS Efficiency

4.1 Introduction

As a result of studying the tests reported above, several modifications were made to SNePS 2.7.0. The modified version of SNePS is SNePS 2.7.1. The following sections describe the modifications in two groups:

1. modifications to reduce the space requirements of SNePS in order to:
   - lessen the impact of the garbage collector;
   - increase the size of knowledge bases that might be stored without running out of memory;
   - and incidentally reduce the running time;
2. modifications primarily to reduce the running time of SNePS.

Other major improvements to the design of SNePS have been scheduled for the implementation of SNePS 3. Those improvements are discussed in the third section below.

4.2 Modifications to Reduce Space

Several of the data structures used in SNePS are composed of a fixed number of components, and are implemented with straight-forward constructors and selectors. Some of them that were implemented in SNePS 2.7.0 as lists have been changed to vectors. A vector with n elements uses half the space of a list with n elements, and moreover its elements are faster to access.

A SNePS assertion has a set of supports, which are used by SNeBR, the SNePS Belief Revision System (Shapiro & The SNePS Implementation Group 2008, Chapter 8). Each support includes an origin set, a set of hypotheses which were used to derive the assertion, and a restriction set, a set of sets of hypotheses, each of which, when unioned with the origin set, forms a minimal nogood. A minimal nogood is a set of hypotheses known to be inconsistent such that no subset of it is known to be inconsistent (Forbus & de Kleer 1993). Restriction sets have now been eliminated in favor of a global set of minimal nogoods. Since SNeBR was not invoked in any of the tests discussed in this paper, this modification is mainly relevant for the space it saves in the support set of each assertion.

4.3 Modifications to Reduce Running Time

As noted in §3.2, in SNePS 2.7.0, a query or subquery is matched against "all the nodes in the KB except base nodes (individual constants)". In SNePS, every functional term or predicate of arity greater than 0 is represented by an assertional frame. The set of slots of the frame, called a caseframe, constitutes a higher order function (or predicate). Comparing the caseframe of the query with the caseframe of a potential match could eliminate the potential match without actually performing the match routine. This is complicated by the fact that slots themselves can be inferred via path-based inference (Shapiro 1991). Nevertheless, we implemented a filter in SNePS 2.7.1 that considers the caseframes and the possibility of path-based inference so that the match routine is not even tried on nodes for which it is relatively easy to tell that it would not possibly succeed. In SNePS 3, nodes will be indexed by caseframes (considering path-based inference) to further reduce the number of nodes on which the match routine will be tried.

4.4 Modifications Planned for SNePS 3

In SNePS 2.7.0, sets of nodes are represented by ordered lists, and set operations, such as intersection and union, are implemented using hashsets. It would be more efficient to represent the sets as hashsets also, but knowledge of the implementation as lists is too pervasive in the code to make that change. In SNePS 3, sets of nodes are being represented by hashsets.

As discussed in §3.11, which rule is more efficient between

```
all(x,y,z)(A(x,y) => (B(y,z) => C(x,z)))
```

and

```
all(x,y,z)(B(y,z) => (A(x,y) => C(x,z)))
```

depends on whether `x` or `z` is bound by the query. In SNePS 3, the rule will be expressed either as

```
(&=> (setof (A x y) (B y z)) (C x z))
```

or as

```
(C (any x (A x (any y))) (any z (B y z)))
```

(*see* (Shapiro 2004)), neither of which specify which antecedent should be tried first. We intend to implement SNePS 3 to consider which variables are free or bound, and to order antecedents accordingly.

5 Post-Modification Reruns

Figures 12–22 show the results of the improvements made to SNePS. Each of these figures show the total times, in milliseconds, of the test for SNePS 2.7.0 and for SNePS 2.7.1. Note, especially that

- In test 1, SNePS 2.7.1 actually ran slower for 7,000 extraneous propositions than SNePS 2.7.0, but still took less than 4 ms. For 8,500 extraneous propositions, SNePS 2.7.1 took about $1/3$ the time of SNePS 2.7.0.
- In tests 4, 6, and 8, SNePS 2.7.0 suffered from garbage collection spikes, but these spikes have been eliminated in SNePS 2.7.1.
- In tests 5, 7, and 9, SNePS 2.7.0 reached out of memory conditions (after 4,000, 5,500, and 7,000 extraneous propositions, respectively), but SNePS 2.7.1 was able to run with 8,500 extraneous propositions.
- In tests 2 and 3, SNePS 2.7.1 performed significantly faster than SNePS 2.7.0 after 7,000 (in test 2) or 2,500 (in test 3) extraneous propositions.
- In test 10, SNePS 2.7.1's performance was exactly the same as SNePS 2.7.0's, both of which took only 10 ms to run with 8,500 extraneous propositions.
- In the Firewall Rules Inference test, SNePS 2.7.1 performed better than SNePS 2.7.0 regardless of the form of the rule.

6 Conclusion

Various timing tests were performed on the SNePS 2.7.0 knowledge representation and reasoning system (Shapiro & The SNePS Implementation Group 2008). The tests varied on the format of some simple reasoning rules, and on the size of the knowledge base when the tests were run. The size of the knowledge base was varied by introducing various numbers of propositions that, although extraneous to the results of the reasoning rules, were involved in the matching of goals and subgoals.

For most tests, the number of extraneous propositions was increased in steps up to 8,500. However, in three tests the available heap was exceeded before 8,500 extraneous propositions could be entered. In one case this happened between 4,000 and 5,000 extraneous propositions. In three tests, the run times were affected by garbage collection spikes. These six tests showed the need for improvements in the amount of space used to store SNePS knowledge bases.

The time to perform the tests when the maximal number of extraneous propositions were present varied from 10 ms to 2,204 secs, showing the need for improvements in the speed of various SNePS procedures.

After profiling SNePS performance, several basic software engineering improvements were made to SNePS data structures and procedures. The results were that all tests could be run with 8,500 extraneous propositions without incurring a heap overflow, garbage collection spikes were eliminated, and running times were improved. Additional modifications in the design of the data structures and procedures of SNePS are planned for SNePS 3.

References

Choi, J., and Shapiro, S. C. 1991. Experience-based deductive learning. In *Third International Conference on Tools for Artificial Intelligence TAI '91*. Los Alamitos, CA: IEEE Computer Society Press. 502–503.

Choi, J. 1993. *Experience-Based Learning in Deductive Reasoning Systems*. PhD dissertation, Technical Report 93-20, Department of Computer Science, State University of New York at Buffalo, Buffalo, NY.

Escalada-Imaz, G., and Ghallab, M. 1988. A practically efficient and almost linear unification algorithm. *Artificial Intelligence* 36(2):249–263.

Forbus, K. D., and de Kleer, J. 1993. *Building Problem Solvers*. Cambridge, MA: MIT Press.

Kandefer, M.; Shapiro, S.; Stotz, A.; and Sudit, M. 2007. Symbolic reasoning in the cyber security domain. In *Proceedings of MSS 2007 National Symposium on Sensor and Data Fusion*.

Shapiro, S. C., and The SNePS Implementation Group. 2008. *SNePS 2.7 User's Manual*. Department of Computer Science and Engineering, University at Buffalo, The State University of New York, Buffalo, NY. Available as `http://www.cse.buffalo.edu/sneps/Manuals/manual27.pdf`.

Shapiro, S. C. 1986. Symmetric relations, intensional individuals, and variable binding. *Proceedings of the IEEE* 74(10):1354–1363.

Shapiro, S. C. 1991. Cables, paths and "subconscious" reasoning in propositional semantic networks. In Sowa, J., ed., *Principles of Semantic Networks: Explorations in the Representation of Knowledge*. Los Altos, CA: Morgan Kaufmann. 137–156.

Shapiro, S. C. 2004. A logic of arbitrary and indefinite objects. In Dubois, D.; Welty, C.; and Williams, M.-A., eds., *Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference (KR2004)*, 565–575. Menlo Park, CA: AAAI Press.

Figures

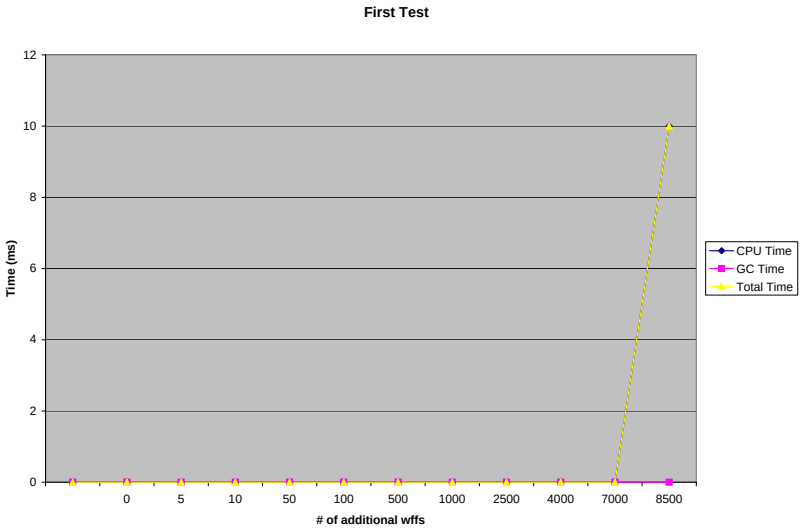

Figure 1: Test 1

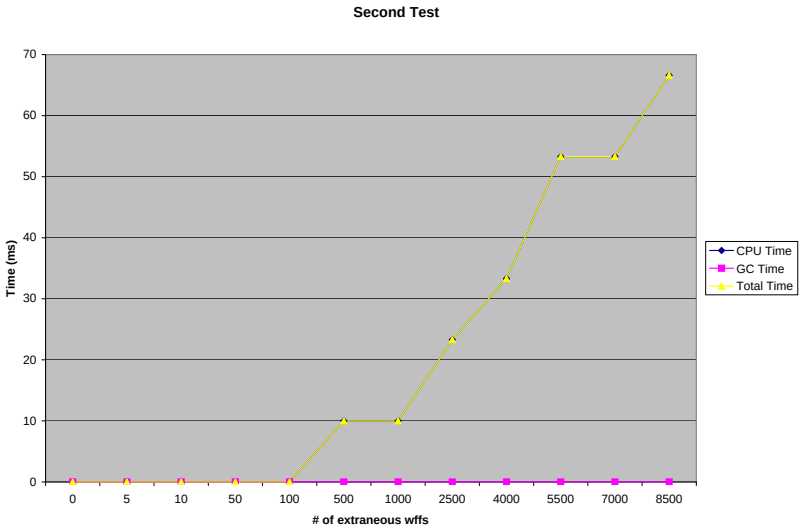

Figure 2: Test 2

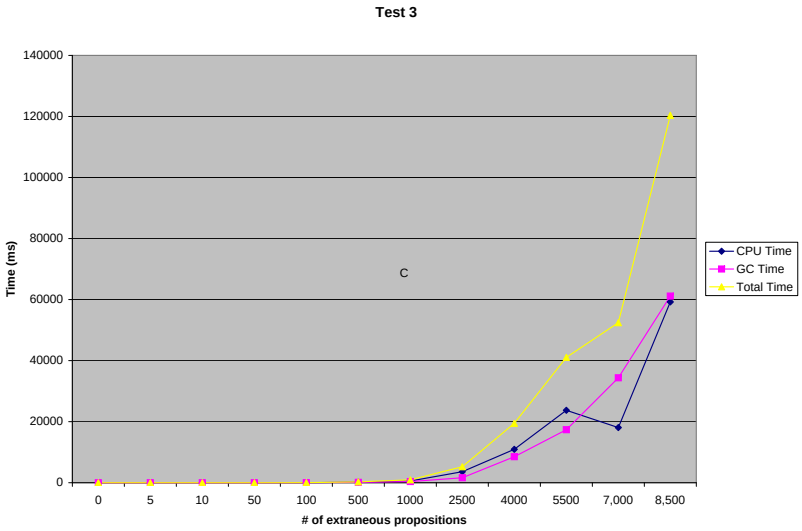

Figure 3: Test 3

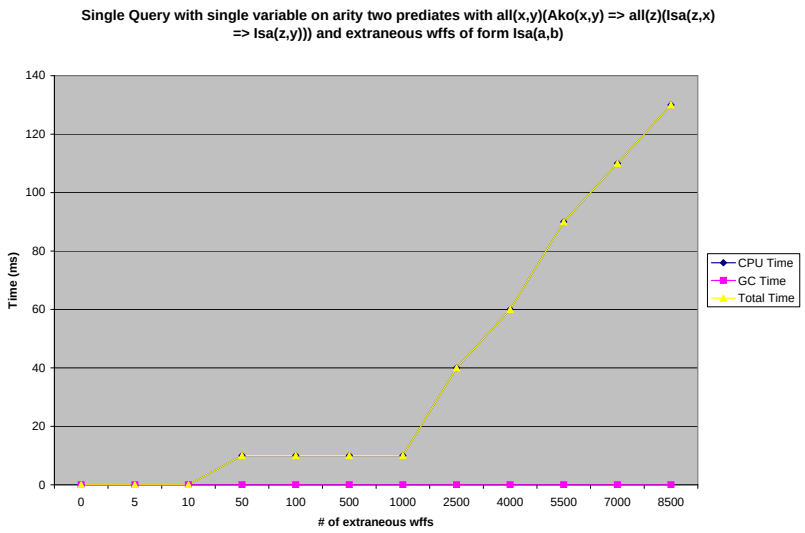

Figure 4: Test 4

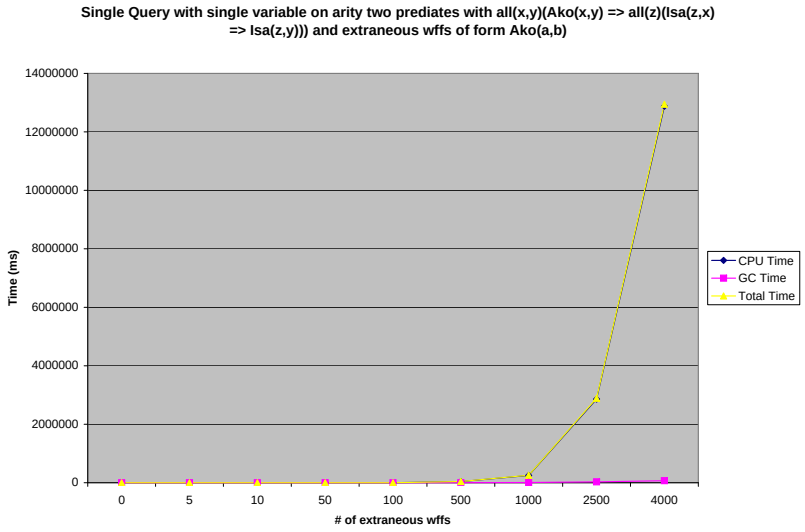

Figure 5: Test 5

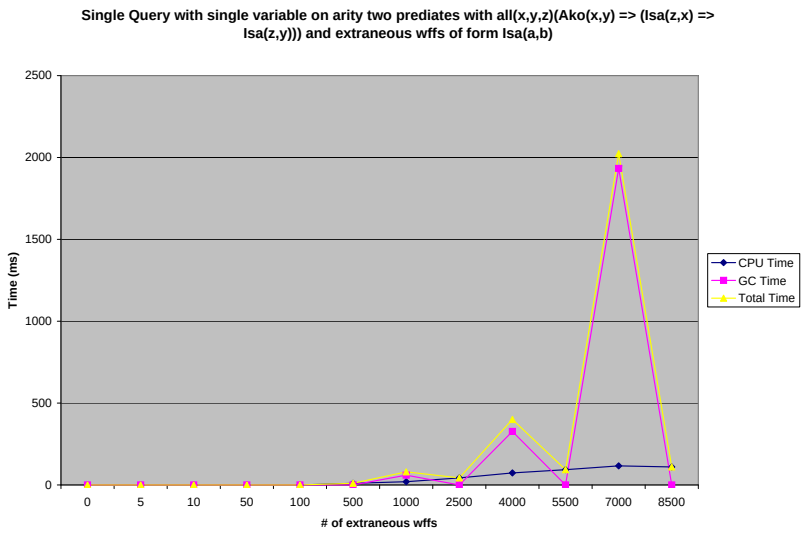

Figure 6: Test 6

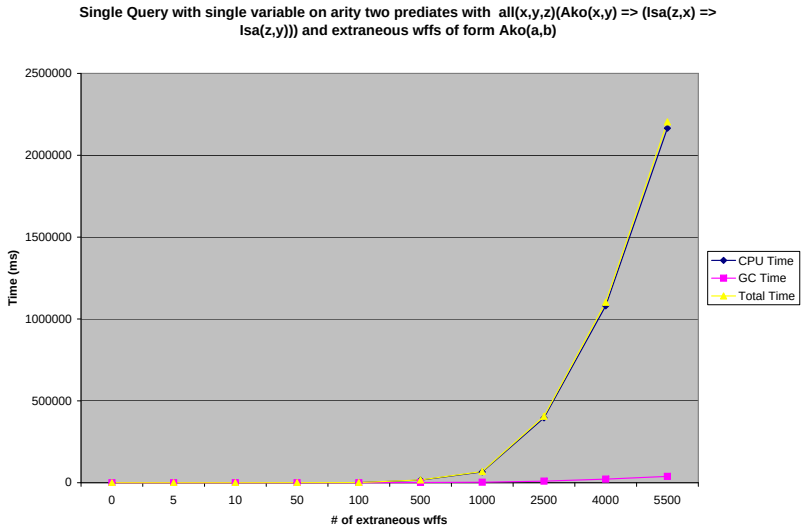

Figure 7: Test 7

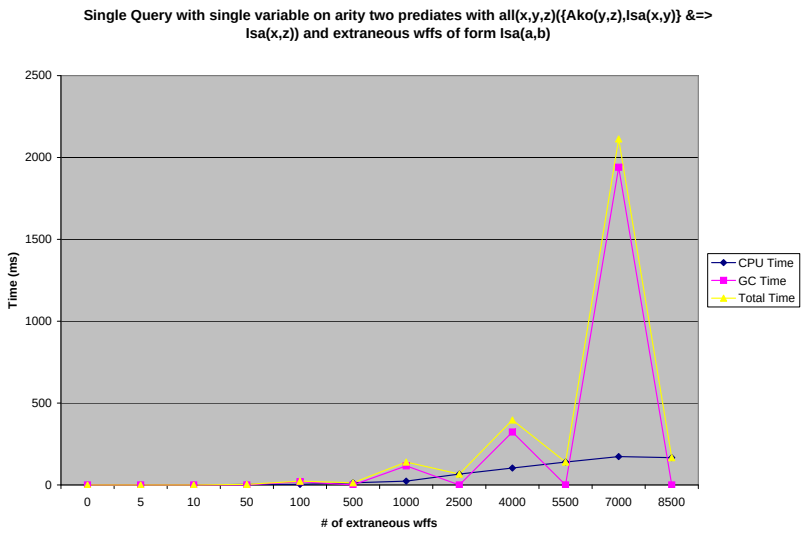

Figure 8: Test 8

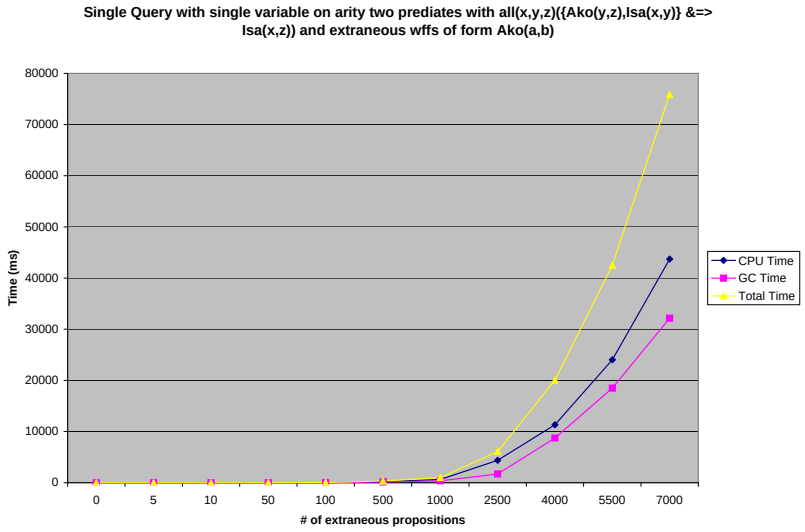

Figure 9: Test 9

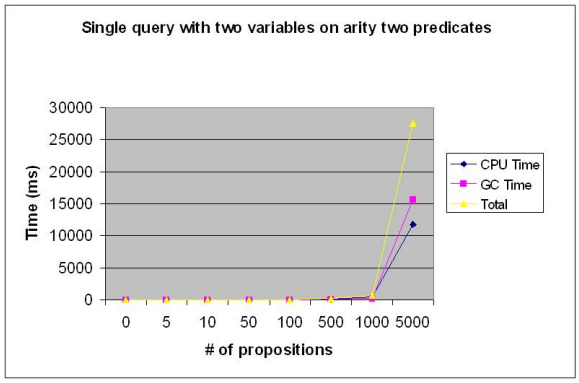

Figure 10: Test 10

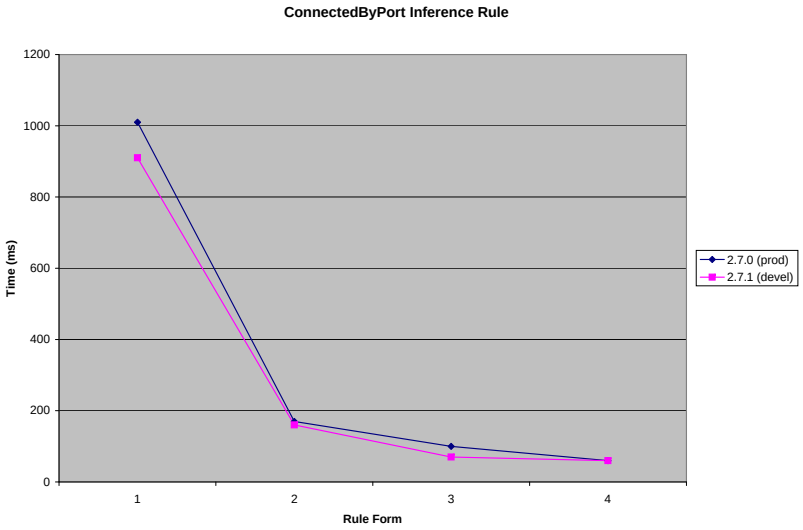

Figure 11: Firewall Rules Inference Test

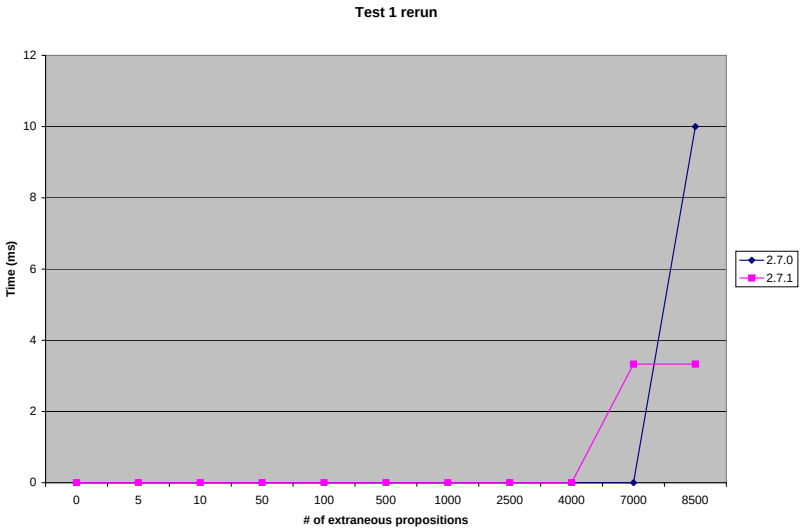

Figure 12: Test 1 rerun

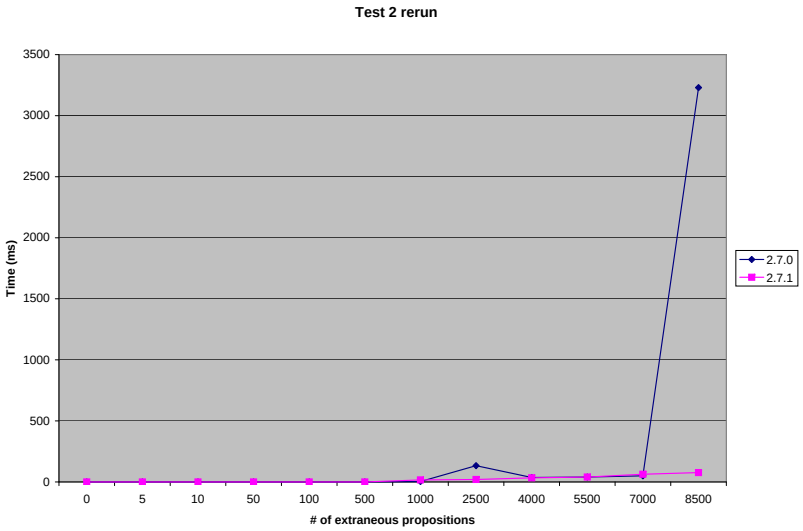

Figure 13: Test 2 rerun

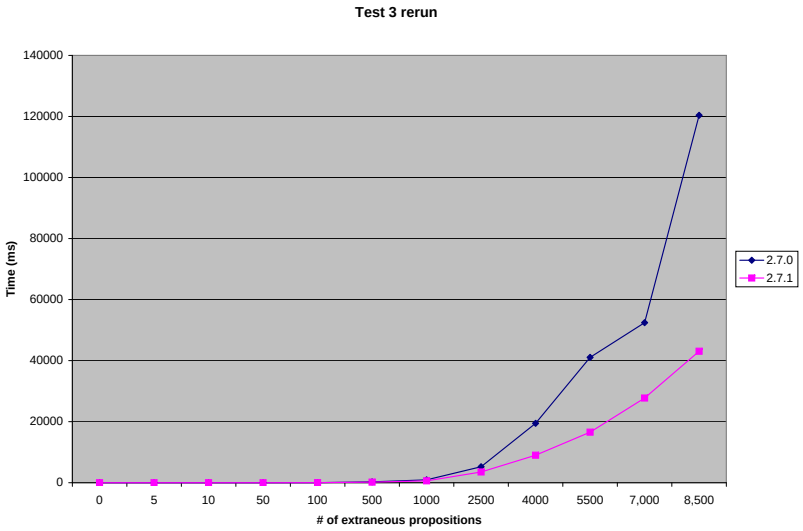

Figure 14: Test 3 rerun

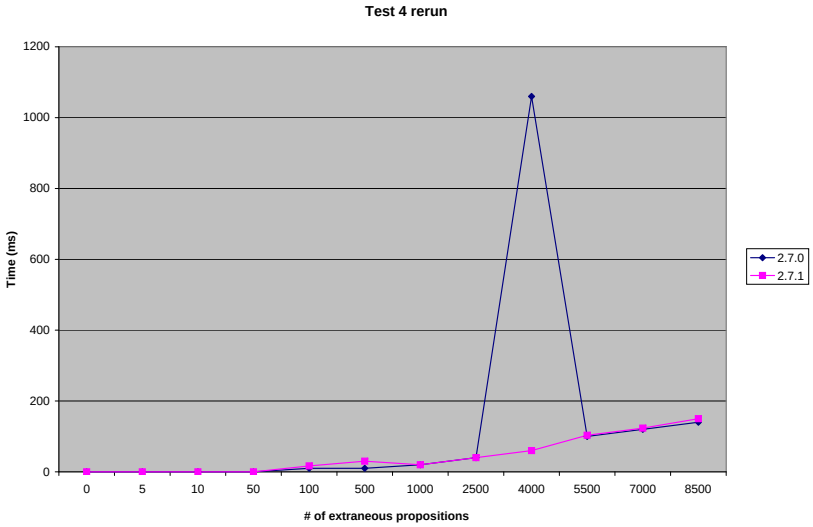

Figure 15: Test 4 rerun

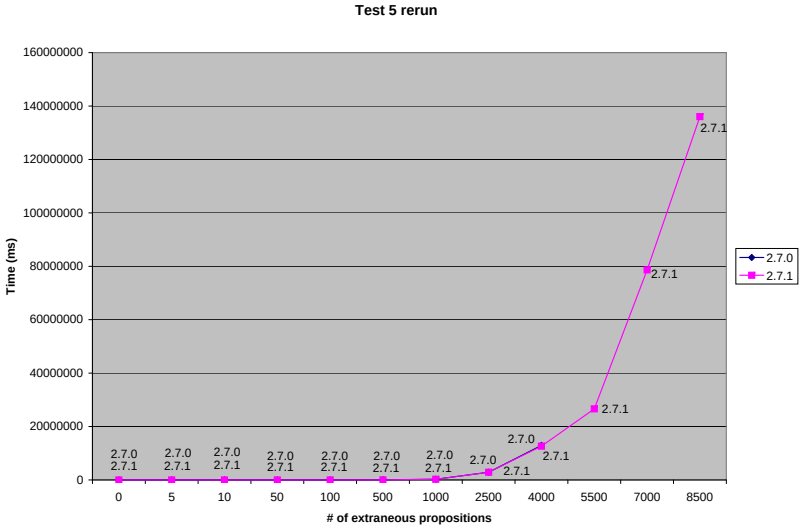

Figure 16: Test 5 rerun

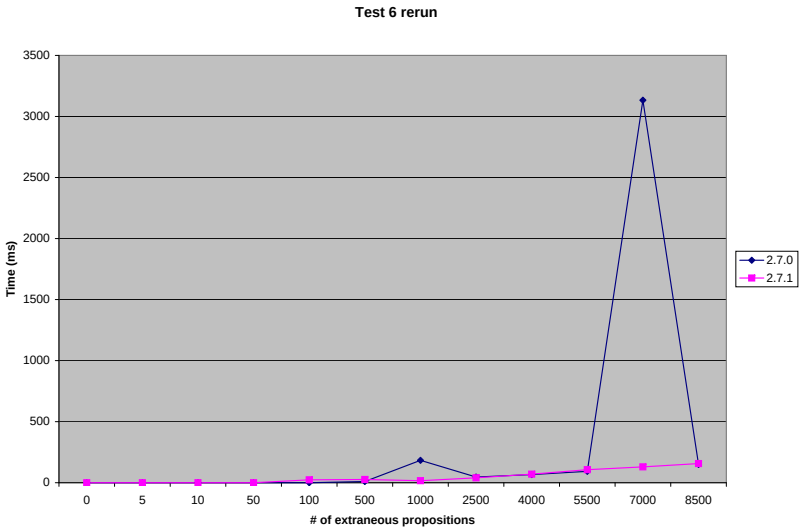

Figure 17: Test 6 rerun

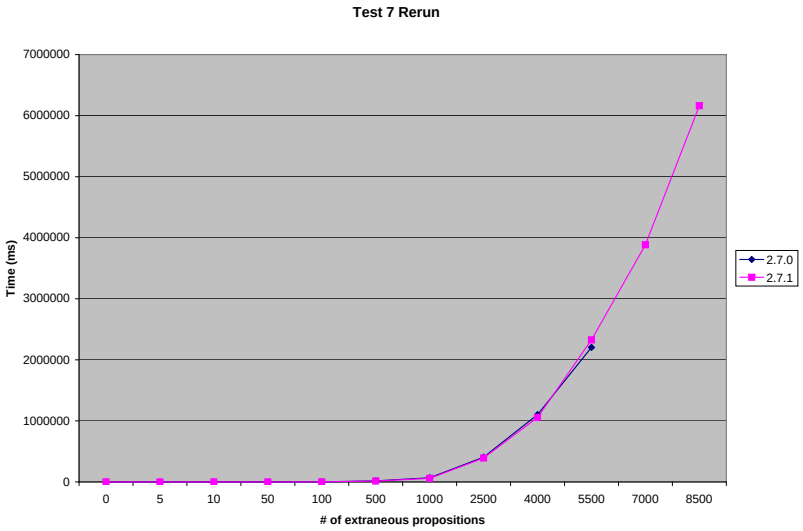

Figure 18: Test 7 rerun

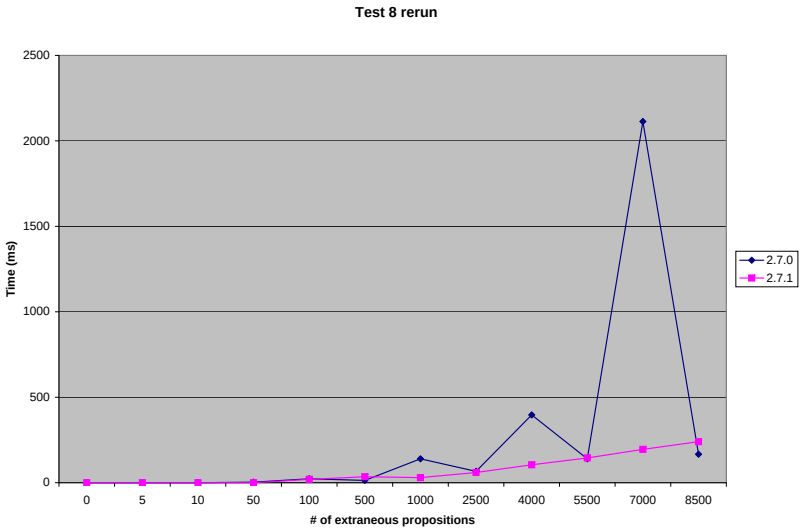

Figure 19: Test 8 rerun

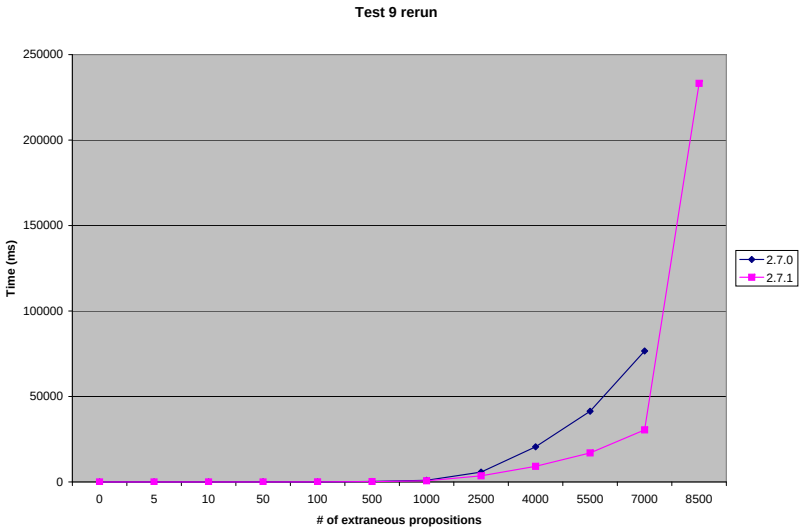

Figure 20: Test 9 rerun

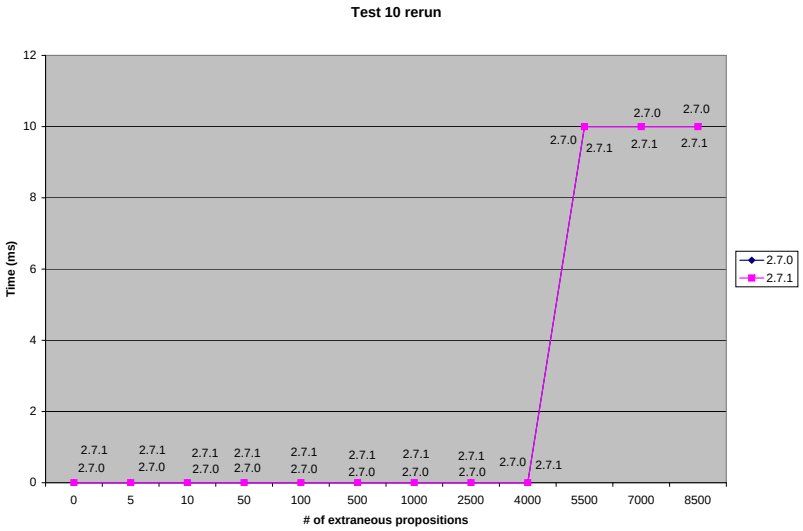

Figure 21: Test 10 rerun

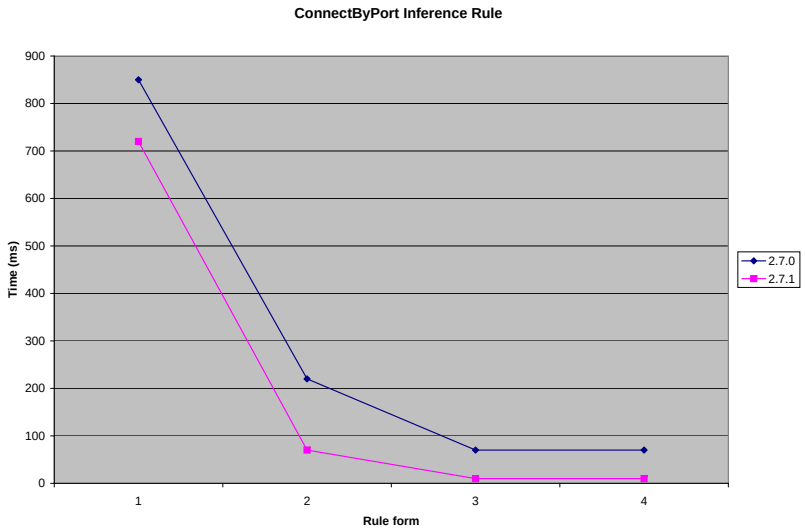

Figure 22: Firewall Rules Inference Test rerun