
Chapter 8

Brokering Theorem Proving Services – MathServe at the CADE System Competition

In Chapter 7 we introduced the MathServe framework and showed how semantic descriptions of reasoning web services (Chapter 4) can be used for service matchmaking and automated service composition (Chapters 5 and 6.3.1). We also showed how composite services can be represented in the decision-theoretic language DTGolog. The MathServe Broker can use a DTGolog interpreter to make optimal decisions when faced with a reasoning problem.

In this chapter we evaluate the performance of the MathServe Broker with respect to the brokering of ATP services, i.e. services that attempt to automatically solve theorem proving problems in classical first-order logic with equality. The most important and influential evaluation method for ATP systems is the system competition associated with the annual Conference on Automated Deduction (CADE) [Nieuwenhuis, 2005]. On the CADE ATP System Competition (CASC), fully automatic ATP systems for first-order logic are evaluated on a set of problems composed of old problems randomly chosen from the TPTP Library, and a number of new problems that the ATP systems have not seen before. The systems are evaluated with respect to the number of problems solved, and the average runtime for successful runs.

MathServe participated in the demonstration division of CASC 20 [Sutcliffe, 2005]. The overall aim of MathServe’s participation was to evaluate MathServe’s brokering capabilities for atomic theorem proving services. In particular, the aim was to find answers to the following questions:

- Is the MathServe framework stable enough for the demands of an established system competition?
- Do Specialist Problem Classes (SPCs)¹ constitute a suitable classification of first-order theorem proving problems?

¹Remember that Specialist Problem Classes (SPCs) are classes of problems determined by syntactical features (cf. Section 4.4.3).

- Does the “optimal policy” computed by the DTGolog interpreter perform better than the best ATP systems in the competition division?

In what follows we give answers to these questions. We describe the CASC competition setup (Section 8.1), the preparation of MathServe for the system competition (Section 8.2), and the performance of MathServe during the competition and in additional experiments performed after the competition (Section 8.3).

8.1 The CADE System Competition

The CADE System Competition [Pelletier *et al.*, 2002] is divided into divisions according to problem and system characteristics. There are five competition divisions in which systems are explicitly ranked:

- The *MIX* division contains mixed CNF really non-propositional theorems (with unsatisfiable clause sets)
- The *FOF* division consists of mixed non-propositional first-order form theorems
- The *SAT* division is composed of CNF problems that are non-propositional non-theorems (with satisfiable clause sets)
- In the *EPR* division, provers are tested on effectively propositional theorems (and non-theorems) in CNF
- The *UEQ* division contains unit equality problems in CNF that are non-propositional theorems

There is also a demonstration division in which systems demonstrate their abilities without being formally ranked. The set of problems in the demonstration division is the union of the problem sets of all competition divisions. In all divisions, CPU and wall-clock time limits are imposed on each solution attempt. The CPU time limit is chosen as a reasonable value within the range allowed, and is announced on the day of the competition. The wall-clock time limit is imposed in addition to the CPU time limit, to avoid very high memory usage. The wall-clock time limit is double the CPU time limit. In the demonstration division, each entrant can choose to use either a CPU or a wall-clock time limit.

ATP systems taking part in any of the competition divisions have to be installed on the system machines before the *system installation deadline*, which is typically 3 weeks before the competition. Systems in the demonstration divisions can also be installed later or can be run on hardware provided by the entrant.

Problem Selection. The competition problems are chosen from a new version of the TPTP Library. The version used for the competition is released after the system installation deadline, so that new problems have not been seen by the entrants. The problems have to meet certain criteria to be eligible for selection. Performance data from submitted systems is used for computing the problem ratings for the TPTP Library. From these ratings each problem is classified as:

- Easy: The problem is solvable by all state-of-the-art ATP systems, or
- Difficult: The problem is solvable by some state-of-the-art ATP systems, or
- Unsolved: The problem has not yet been solved by any ATP system, or
- Open: The theorem-hood is unknown.

Difficult problems with a rating in the range 0.21 to 0.99 are eligible. The competition problems are randomly selected from all eligible problems at the start of the competition, based on a seed supplied by the competition panel. The selection is constrained so that no division contains an excessive number of very similar problems. The selection mechanism is biased to select problems that are new in the TPTP Library, until 50% of the problems in each category have been selected. After that, random selection (from old and new problems) continues. The actual percentage of new problems used depends on how many new problems are eligible and the limitation on very similar problems.

It is worth mentioning that developers of ATP systems do not publish the latest versions of the systems before the CADE System Competition. Therefore, these versions of the provers were not available to MathServe before the competition.

8.2 Training the MathServe Broker

To prepare the MathServe framework for the system competition we had to collect performance data of the available ATP services. We were faced with the task of finding suitable training sets of theorem proving problems. Since MathServe was going to be evaluated with respect to a new version of the TPTP Library, it was natural to train MathServe on TPTP Library problems. However, we also looked for a second, independent set of problems to avoid an over-fitting of the learnt policy. The result of our investigation was that the TPTP Library is the only existing problem set of a sufficient size that is also heterogeneous enough with respect to a partitioning in Specialist Problem Classes. The MPTP Library [Urban, 2003, Urban, 2004], which consists over 30000 problems automatically extracted from the Mizar Library, is too homogeneous. All problems of the MPTP Library fall in one of three SPCs (FOF_NKC_RFO_EQU, FOF_NKC_EPR, FOF_NKC_RFO_NEQ) which results from the fact that these problems are automatically generated. This is why we trained the MathServe Broker on the complete TPTP Library (Version 3.0.1) which is also the most common training method for standalone ATP systems.

For every SPC of the TPTP Library we counted the problems solved by the ATP systems Ep, Otter, SPASS and Vampire with a CPU time limit of 300 seconds per problem. Both the percentage of problems solved as well as the CPU time for solved problems were measured. Table 8.1 compares the performance of the four ATP systems. For the sake of readability it contains only the percentage of problems solved by the systems ². Numbers in bold font indicate the *strongest* system in each SPC, which is the system that solved most of the problems. In case two or more systems solved the same number of problems, the best system was determined by the average CPU time

²The full tables including the CPU times can be found in Appendix D.

used for successful problem solving attempts. For instance, Ep, SPASS and Vampire solved all problems in the SPC of first-order problems that are real first-order and contain no equality (FOF_NKC_RFO_NEQ). However, in average, SPASS used the least time resources and was marked as the most successful system.

Specialist Problem Class	No. Probs	Percentage of Problems solved			
		Ep 0.82	Otter 3.3	SPASS 2.1	Vampire 7.0
FOF_CSA_EPR	319	91	0	86	68
FOF_CSA_RFO	18	67	0	67	50
FOF_SAT_EPR	3	34	0	34	34
FOF_SAT_RFO	17	41	0	29	26
FOF_NKC_EPR	395	81	55	98	80
FOF_NKC_RFO_EQU	915	64	34	53	61
FOF_NKC_RFO_NEQ	28	100	96	100	100
FOF_NKS_NUN_EPR	50	14	2	100	8
FOF_NKS_NUN_RFO_NEQ	0	0	0	0	0
FOF_NKS_NUN_RFO_EQU	0	0	0	0	0
CNF_NKS_EPR	476	98	77	99	99
CNF_NKS_RFO_NEQ_NHN	540	70	49	54	67
CNF_SAT_EPR	220	64	0	0	48
CNF_SAT_RFO_NEQ	275	52	0	45	51
CNF_SAT_RFO_EQU_NUE	224	58	0	55	50
CNF_SAT_RFO_PEQ_UEQ	54	13	0	11	11
CNF_NKS_RFO_NEQ_HRN	461	89	72	66	88
CNF_NKS_RFO_SEQ_HRN	390	87	56	49	75
CNF_NKS_RFO_SEQ_NHN	1816	48	25	34	42
CNF_NKS_RFO_PEQ_NUE	337	88	20	75	85
CNF_NKS_RFO_PEQ_UEQ	722	85	73	71	83

Table 8.1: Percentage of Problems solved by the ATP systems Ep, Otter, SPASS and Vampire in the 21 SPCs of the TPTP Library 3.0.1. Systems were run with a 300sec time limit for each problem

The success rates in Table 8.1 and the average CPU times were modelled as conditional stochastic effects of the corresponding ATP services as shown in Section 4.4.3. The ATP service EpService, for instance, has the stochastic effect

$$\text{problemClass}(tstp_problem, mw\#FOF_CSA_EPR) \\ \rightarrow \text{status}(atp_result, stat\#\text{Theorem}) (0.91) (5470\text{ms})$$

which describes the performance of the Ep system on problems of the class FOF_CSA_EPR. These effects were translated into corresponding probabilities and action costs of a DT-Golog domain.

The MathServe Broker was given the following DTGolog program which consists of a simple choice of one out of all available ATP services:

```

proc(atpChoice(TstpProblem, ATPResult),
     otterATP(TstpProblem, Time, ATPResult)
   | epATP(TstpProblem, Time, ATPResult)
   | vampireATP(TstpProblem, Time, ATPResult)
   | spassATP(TstpProblem, Time, ATPResult)).

```

The broker's program analysis detected that all services of the choice construct have conditional stochastic effects whose antecedents are atoms of the form

$$\text{problemClass}(tstp_problem, x).$$

Thus, the broker searched for appropriate sensing actions, i.e. services that have similar atoms in their effect list.³ The only available service fulfilling this requirement was `TstpAnalyser` (cf. Section 4.5.1). This sensing action was introduced as the first step in the extended program that was given to the offline DTGolog interpreter:

```

proc(atpChoiceSensing(TstpProblem, ATPResult),
     tstpAnalyser(TstpProblem, Class) ;
   ( otterATP(TstpProblem, Time, ATPResult)
   | epATP(TstpProblem, Time, ATPResult)
   | vampireATP(TstpProblem, Time, ATPResult)
   | spassATP(TstpProblem, Time, ATPResult))).

```

For this program, the success rates in Table 8.1 do already reflect the optimal policy, in the sense that for every SPC the *strongest* ATP system in that SPC will be chosen. The optimal policy for program `atpChoiceSensing` is a new DTGolog program with 21 conditional statements, one for each SPC:

```

proc(atpChoiceOpt(TstpProblem, ATPResult),
     tstpAnalyser(TstpProblem, Class) ;
     senseEffect(tstpAnalyser(TstpProblem, Class)) ;
     if(problemClass(Var1, 'mw#CNF_NKS_RFO_SEQ_HRN'),
        epATP(Var1, Var4, Var2),
        if(problemClass(Var1, 'mw#FOF_CSA_EPR'),
           epATP(Var1, Var4, Var2),
           if(problemClass(Var1, 'mw#FOF_NKS_NUN_RFO_NEQ'),
              spassATP(Var1, Var4, Var2),
              ...
           )
        )
     )
end

```

For the system competition the MathServe Broker was given the optimal policy `atpChoiceOpt` so that it did not have to re-calculate the policy for every incoming theorem proving problem.

At the time of the competition the MathServe Broker (and its DTGolog interpreter) did not have a sophisticated time resource management. For a sequence of actions

³jz: We have to find a nice way of introducing sensing actions. I suppose the easiest way would be to implement an "online" Golog interpreter which can do some planning and can invoke sensing actions whenever they are needed. So far, we only have an offline interpreter available in Prolog.

δ_1 ; δ_2 the broker did execute δ_1 and measure the wall-clock time used by δ_1 . This time was subtracted from the overall time available. The remaining time was allocated to the program δ_2 . For actions which take a time resource as one of their inputs, all the available time at a particular point of the program execution was given to these actions. For the optimal policy presented above this means that the time used by the problem analyser (TstpAnalyser) was subtracted from the time available to the theorem proving services.

8.3 MathServe at CASC-20

The 20th CADE System Competition (CASC-20) [Sutcliffe, 2005] took place on the 26th July 2005 in Tallinn, Estonia. CASC-20 counted 19 entrants, most of which participated in some of the five competition divisions. The ATP system Ep was the only system participating in all competition divisions.

The problem set for CASC-20 was composed of 660 randomly chosen problems from the TPTP Library. 147 of these problems had not been seen by the provers before. Table 8.2 shows the distribution of the 660 problems over seven SPCs, and the prover

Specialist Problem Class	Problems	ATP chosen
CNF_NKS_RFO_PEQ_NUE	36	Ep
CNF_NKS_EPR	115	SPASS
CNF_NKS_RFO_NEQ_NHN	100	Ep
CNF_NKS_RFO_PEQ_UEQ	160	Ep
CNF_NKS_RFO_SEQ_NHN	99	Ep
FOF_NKC_RFO_NEQ	35	SPASS
FOF_NKC_RFO_EQU	115	Ep
Complete	660	

Table 8.2: Distribution of CASC-20 problems over seven SPCs

chosen by MathServe’s optimal policy for these SPCs. The fixed CPU time limit for each problem was announced on the competition as 600 seconds.

Since MathServe does not constitute a new ATP system, but employs other ATP systems, it participated in the demonstration division. A specialised MathServe client was run sequentially on all 660 problems with 600sec CPU and wall-clock time limit.

Strategy	Problems Solved	Percentage of Given	Average CPU for solved	Average WC for solved	Average WC for query
Optimal	392	59.4	27597	28120	224331
Random Choice	297	45.0	23209	23340	290017
Split & Join	405	61.4	25452	347599	442174

Table 8.3: Performance of MathServe on 660 CASC problems using different strategies. All times are in milliseconds

The client was executed on a competition machine provided by the CASC organisers. For each problem, a corresponding query containing the problem was sent over

the Internet to a MathServe server running on a machine at Saarbrücken University, Germany. The server ran on a Linux machine with four Intel Xeon 2.80GHz CPUs and 2GB total memory. Table 8.3 shows the performance of MathServe on the CASC problems using different strategies.

Performance using the optimal policy. On the competition itself MathServe used the optimal strategy (`atpChoiceOpt`) computed by the DTGolog interpreter. With this strategy it could solve 392 problems, which corresponds to a success rate of 59.4%. The average CPU time used for solved problems was approximately 28 seconds. The average wall-clock time used for all queries was 224 seconds, which also includes the time for unsuccessful calls that account for 600 seconds each.

Performance using random choice. After the CADE System Competition we also tested MathServe with the Golog program performing a random choice of an ATP service (`atpChoice`). For three independent runs, we counted the number of problems solved and the average CPU and wall-clock times. The second row in Table 8.3 shows the average performance over these three runs. When randomly choosing an ATP system, MathServe could solve 95 problems less than with the optimal strategy. However, compared to the optimal strategy, the result for solved problems was (on average) returned in 24 seconds, which is 14% faster than with the optimal strategy. This is mainly due to the problem analysis step (the invocation of the service `TstpAnalyser`), which was performed in the program `atpChoiceOpt`.

Performance with parallel service invocation. At the time of CASC-20, the Golog interpreter of MathServe could not execute programs containing concurrent action execution. However, we defined a composite service in OWL-S which invokes all available ATP services in parallel using the Split & Join construct of OWL-S (cf. Section 3.2.2.2). Executing this composite service, MathServe could solve 405 (61.4%) of the CASC problems. The average CPU time used by the ATP systems which solved the problems first was 25 seconds. This was only slightly above the CPU time measured in the case of random choice. The execution of a Split & Join construct terminates when all parallel service invocations have terminated. This is why the average wall-clock time for successful queries was 348 seconds, which is more than 12 times the wall-clock time needed when using the other strategies.

Ednote!

4

8.4 Comparison with Leading CASC Systems.

MathServe could solve 392 of the 660 problems given in the CASC-20 competition. However, it did not outperform the most powerful ATP systems Ep and Vampire. Table 8.4 shows the performance of MathServe compared to that of Ep and Vampire.

⁴What we actually want to do is to measure the overall CPU time resource used by all parallel service invocations. I'd have to make another run for this.

System	Problems given	Problems solved	Percentage of given	Percentage complete
MathServe 0.62	660	392	59.4%	59.4%
Ep 0.9pre3	660	409	62.0%	62.0%
Vampire 8.0	540	430	79.6%	65.2%
Vampire 7.0	300	262	87.4%	39.7%

Table 8.4: Comparison of MathServe with leading ATP systems Ep and Vampire

8.4.0.1 Comparison with the Ep System.

We performed a detailed comparison of MathServe with the Ep system, which took part in all competition divisions (i.e. Ep was run on all 660 problems). For 38 of the 268 problems not solved by MathServe, Ep (0.9pre) found a solution. In further experiments we identified why this (newer) version of Ep could solve these problems but MathServe could not.

On the one hand, the better performance of Ep was due to the improvements performed on version 0.9 of the Ep system, which entered the competition. In fact, 28 of the 38 problems (i.e. 73.7%) could only be solved by the newer version of Ep and not by the older version used by MathServe. Six of the remaining ten problems could not be solved by MathServe due to their size. The source of this failure was the fact that the queries sent to MathServe are XML documents that have to be parsed to determine the type of the query and to extract the TSTP problem description. We realised that XML parsing leads to a memory usage approximately 20 times the size of the XML document, which causes a memory overflow for all problems with a size bigger than 2.5MB (Megabytes). Table 8.5 shows the names and the sizes of the problems that were too big to be solved by MathServe.

Problem	Size (MB)
SYN831-1	8.6
SYN853-1	11.0
SYN830-1	9.2
SYN816-1	2.8
SYN864-1	3.4
SYN865-1	3.4

Table 8.5: CASC Problems not solved by MathServe due to their size.

For 13 of the 38 unsolved problems, the optimal strategy of MathServe suggested to use the ATP service SPASS (V.2.1) instead of Ep. The six problems in Table 8.5 were among these problems and could not be parsed by MathServe. We ran Ep on the remaining seven problems to see whether MathServe could have solved these problems if it had chosen Ep instead. Only four of the seven problems could have been solved by choosing Ep instead of SPASS.

From the above experiments we concluded that, even if MathServe had managed to deal with theorem proving problems bigger than 2.5MB, and if it had chosen Ep instead of SPASS for some problems, it would have only been able to solve ten problems more than with the original strategy. This is mainly due to the significant improvements done

to Ep from version 0.82 (used by MathServe) and version 0.9, which entered CASC-20. Even with improvements on the MathServe system itself and the optimal policy computed, MathServe would not have been able to outperform the system Ep.

8.4.0.2 Comparison with the Vampire System.

The Vampire system did not participate in all competition divisions. However, Vampire 7.0 and Vampire 8.0 were the two leading systems in the prestigious MIX and FOF divisions of CASC-20. This is why we also performed a detailed comparison of MathServe with the Vampire 8.0.

For 71 of the 268 problems not solved by MathServe, Vampire 8.0 could find a solution. Again, the six problems in Table 8.5 were among these problems. MathServe could theoretically use Vampire 7.0 but did not do so because of the performance data in Table 8.1 and the resulting optimal policy `atpChoiceOpt` shown above.

Vampire 7.0 could solve 45 of the 71 problems (i.e. 63.4%). Thus, the performance difference between two versions of Vampire is smaller than in the case of Ep, where the older version could only solve 26.3% of 38 problems.

Table 8.6 shows the distribution of the 45 problems over five SPCs. The table also shows that MathServe chose Ep for all of these problems except one.

Specialist Problem Class	Solved by Vampire 8.0	TPTP v3.0.1 Problems in SPC	ATP chosen
CNF_NKS_RFO_PEQ_NUE	1	337	Ep
CNF_NKS_RFO_NEQ_NHN	5	540	Ep
CNF_NKS_RFO_SEQ_NHN	15	1816	Ep
FOF_NKC_RFO_NEQ	1	28	SPASS
FOF_NKC_RFO_EQU	23	915	Ep
Complete	45	3636	

Table 8.6: Distribution of 45 problems not solved by MathServe but solved by Vampire, and the ATP chosen by MathServe

We created a new policy in which Vampire was chosen for problems in the SPCs of Table 8.6. We ran MathServe with this new policy on the 384 CASC-20 problems in these SPCs to see how it would have performed if it had chosen Vampire for these problems. With the new policy, MathServe could solve 383 of the 660 CASC-20 problems, i.e. nine problems fewer than with the original policy `atpChoiceOpt`. The average CPU time for solved problems was 33746 milliseconds which is due to the fact that Vampire typically uses more CPU resources than the ATP systems SPASS and Ep. On average, the client had to wait 242 seconds for the result for successfully solved queries, which is 18 seconds more than for the original policy `atpChoiceOpt`.

In a second experiment we created a policy in which Vampire was chosen only for the two SPCs (CNF_NKS_RFO_SEQ_NHN and FOF_NKC_RFO_EQU) that (together) contain 84% of the 45 problems. We ran MathServe with this policy on the 214 CASC-20 problems in these two SPCs. With this policy, MathServe could solve 385 of the 660 CASC-20 problems, which is still seven problems less than with the original policy `atpChoiceOpt`. However, the average CPU time for solved problems, 28628

milliseconds, was just one second longer than with `atpChoiceOpt`. Also, in average, the client had to wait only 228 seconds for an answer to a query.

8.5 MathServe’s Potential

In the previous Section we saw that MathServe solved less problems than the ATP systems Ep and Vampire. This was mainly due to improvements of the ATP systems participating at CASC. Furthermore, MathServe could not process very large problems. To show the full potential of MathServe we integrated the CASC versions of the systems Ep and Vampire into MathServe. Furthermore, we added the systems DCTP and Waldmeister which are specialised on problems that are essentially propositional and on unit equality problems, respectively. We also improved MathServe’s handling of theorem proving problems such that it could also process the large problems shown in Table 8.5.

Specialist Problem Class	Percentage of Problems solved				
	SPASS 2.1	Ep 0.9pre3	Waldmeister 704	DCTP 10.21p	Vampire 8.0
FOF_CSA_EPR	86	92	0	25	0
FOF_CSA_RFO	67	67	0	24	0
FOF_SAT_EPR	34	34	0	67	0
FOF_SAT_RFO	29	41	0	0	0
FOF_NKC_EPR	98	80	0	0	91
FOF_NKC_RFO_EQU*	53	35	0	0	69
FOF_NKC_RFO_NEQ*	100	100	0	0	100
FOF_NKS_NUN_EPR	100	26	0	0	12
FOF_NKS_NUN_RFO_NEQ	0	0	0	0	0
FOF_NKS_NUN_RFO_EQU	0	0	0	0	0
CNF_NKS_EPR*	99	96	0	99	98
CNF_NKS_RFO_NEQ_NHN*	54	68	0	0	75
CNF_SAT_EPR	0	64	0	91	48
CNF_SAT_RFO_NEQ	45	52	0	0	0
CNF_SAT_RFO_EQU_NUE	55	57	0	0	0
CNF_SAT_RFO_PEQ_UEQ	11	13	9	0	0
CNF_NKS_RFO_NEQ_HRN	66	87	0	0	92
CNF_NKS_RFO_SEQ_HRN	49	87	0	0	95
CNF_NKS_RFO_SEQ_NHN*	34	51	0	0	42
CNF_NKS_RFO_PEQ_NUE*	75	87	0	0	88
CNF_NKS_RFO_PEQ_UEQ*	71	85	91	0	85

Table 8.7: Percentage of Problems solved by the ATP systems Ep, DCTP, SPASS, Waldmeister and Vampire in the 21 SPCs of the TPTP Library 3.0.1. Systems were run with a 300sec time limit for each problem. The performance of the Otter system is shown in Table 8.1

Again, we trained MathServe on the full TPTP Library v3.0.1. The percentage of problems solved by the ATP systems is shown in Table 8.7 which corresponds to

Table 8.1 in Section 8.2. SPCs which contain CASC-20 problems are marked with a star.

A new optimal policy was computed by our DTGolog interpreter using the data in Table 8.5. With the new policy, the MathServe broker chose DCTP for essentially propositional problems and Waldmeister for unit equality problems. Vampire was chosen for three of the six SPCs containing CASC-20 problems, Ep for one (CNF_NKS_RFO_SEQ_NHN

Table 8.8 compares MathServe’s performance using the new optimal policy. MathServe could solve 451 problems, i.e. 21 problems more than Vampire 8.0 and 42 problems more than Ep 0.9pre3 alone. MathServe could solve all large problems listed in Table 8.5.

Strategy	Problems Solved	Percentage of Given	Average CPU for solved	Average WC for solved	Average WC for query
Optimal	451	68.3	37618	38437	204262
Random Choice	198	30.0	20206	20786	189073

Table 8.8: Performance of MathServe on 660 CASC problems using the new optimal strategy and the latest versions of DCTP, Ep, Vampire and Waldmeister. All times are in milliseconds

8.6 Summary & Discussion

In this chapter we showed a particular application of MathServe to theorem proving problems in classical first-order logic. MathServe participated in the demonstration division of the 20th CADE System Competition and proved to be stable enough to participate in an established system competition. However, MathServe did not outperform the leading ATP systems Ep and Vampire. Even when invoking all ATP services in parallel, MathServe could only solve 61.4% of the given problems which is still less than the percentage solved by Ep and Vampire. In further experiments we identified that one reason for this was the significant improvements made to the most recent versions of these ATP systems. These improved ATP systems were not available to MathServe at the time of the competition. Furthermore, some problems were too large to be handled by MathServe.

As a consequence we improved MathServe in two ways: 1) we integrated the latest versions of the ATP systems Ep and Vampire as well as the specialised ATP systems DCTP and Waldmeister. 2) We changed the XML processing of MathServe such that it could handle large problems. With these improvements MathServe did outperform both Ep and Vampire.

From the results obtained through our experiments we conclude that SPCs do constitute a suitable classification of first-order theorem proving problems. Also, the data does provide enough evidence to support the thesis that the policy computed from the performance of ATP systems on SPCs performs better than stand-alone ATP systems.

There are many possible directions for future research on the brokering of ATP services. In what follows we briefly discuss some research directions.

So far, parallel execution of ATP services is only possible via the Split & Join construct available in OWL-S. The execution of Split & Join is barrier synchronised, i.e. it completes when all of its components processes have completed. An improved parallel execution should terminate all

A comparison of sets of ATP services (instead of single ATP services) could be used to perform an optimal choice of sets of services for parallel service execution. When executing a set of ATP services in parallel, time resources could be assigned to the individual services according to performance data as shown in Tables ?? and ??. This would minimise the amount of computational resources used while maximising the probability of successfully solving a problem.