Empirical Evaluation of $\{log\}$'s Decision Procedure for Binary Relations

Maximiliano Cristiá[1] and Gianfranco Rossi[2]

[1] CIFASIS and UNR, Rosario, Argentina
cristia@cifasis-conicet.gov.ar
[2] Università degli studi di Parma, Parma, Italy
gianfranco.rossi@unipr.it

In this document we present the results of an empirical evaluation of the implementation of $\{log\}$'s decision procedure for binary relations, namely $SAT_{\mathcal{BR}}$. The empirical evaluation aims at finding out whether $SAT_{\mathcal{BR}}$ can decide, in practice, the satisfiability of formulas involving binary relations and their associated operators in a reasonable time. The fragment of $SAT_{\mathcal{BR}}$ dealing with partial functions was already empirically assessed [CRF15]. In that assessment we used more than 2,000 formulas automatically generated by the Fastest tool [CAF+14] from 10 different Z specifications [Spi92]. Nevertheless, we cannot use those formulas for the assessment of the whole $SAT_{\mathcal{BR}}$ solver because they seldom include binary relations, as they almost always require sets of ordered pairs to be partial functions.

For that reason, we focus this empirical evaluation on binary relations. That is, we seek for formulas where sets of ordered pairs are not restricted to be partial functions. In order to perform an evaluation as objective as possible, we took as base formulas the *standard partitions* generated by the Test Template Framework [SC96] for the relational operators of the Z mathematical toolkit (ZMT) [Saa97][Spi92, ch. 4]. The standard partitions of the TTF are used in test case generation applications to generate test cases to exercise the implementation of the corresponding operators.

Example 1. The standard partition for the $\oplus$ operator (overriding) is shown in Figure 1 (note that all formulas in the partition are satisfiable). This partition can be used, for instance, to generate test cases to test the implementation of a specification containing the expression $H \oplus \{(x, y)\}$ by substituting R with H and S with $\{(x, y)\}$. Note that, in this case, some of the formulas in the partition become unsatisfiable.

We will call each formula in the TTF partition a *base goal*. Taking these goals as a base, we perform the following experiments:

1. Experiment 1. Call $\{log\}$ on each base goal and count the number of goals for which at least one solution is found. This experiment gives an idea of whether or not $\{log\}$ is able to calculate test cases for the TTF.
2. Experiment 2. Conjoin the constraint $oplus(R, S, T)$ to each base goal, run $\{log\}$ on it and count the number of goals for which at least one solution is found. This experiment gives an idea of whether or not $\{log\}$ is able to

Let $R, S \in X \leftrightarrow Y$ for some X and Y then $R \oplus S$ is partitioned as follows:

$$R = \emptyset \land S = \emptyset$$
$$R = \emptyset \land S \neq \emptyset$$
$$R \neq \emptyset \land S = \emptyset$$
$$R \neq \emptyset \land S \neq \emptyset \land \operatorname{dom} R = \operatorname{dom} S$$
$$R \neq \emptyset \land S \neq \emptyset \land \operatorname{dom} S \subset \operatorname{dom} R$$
$$R \neq \emptyset \land S \neq \emptyset \land \operatorname{dom} R \subset \operatorname{dom} S$$
$$R \neq \emptyset \land S \neq \emptyset \land (\operatorname{dom} R \cap \operatorname{dom} S) = \emptyset$$
$$R \neq \emptyset \land S \neq \emptyset \land (\operatorname{dom} R \cap \operatorname{dom} S) \neq \emptyset$$
$$\quad \land \neg(\operatorname{dom} S \subseteq \operatorname{dom} R) \land \neg(\operatorname{dom} R \subseteq \operatorname{dom} S)$$

Fig. 1. Standard partition for $\oplus$.

compute *oplus*(R, S, T) when R and S are constrained to verify non trivial conditions.

3. EXPERIMENT 3. Conjoin the formula $\mathit{npfun}(R) \land \mathit{npfun}(S)$ to each base goal, run $\{log\}$ on it and count the number of goals for which at least one solution is found. This experiment gives an idea of whether or not $\{log\}$ is able to compute test cases where R and S are not allowed to be functions. Note that some goals become unsatisfiable—for example the first one, as the empty set is a function. Therefore, this experiment also evaluates the ability of $\{log\}$ to determine when a formula is unsatisfiable.
4. EXPERIMENT 4. Conjoin the formula $\mathit{oplus}(R, S, T) \land \mathit{npfun}(R) \land \mathit{npfun}(S)$ to each base goal, run $\{log\}$ on it and count the number of goals for which at least a solution is found. This experiment is a combination of EXPERIMENT 2 and EXPERIMENT 3.
5. EXPERIMENT 5. Conjoin the formula $S = \{(x, y)\}$, where x and y are variables, to each base goal. This experiment turns some satisfiable goals into unsatisfiable goals. The experiment represents a typical case in many systems as it is frequent to modify one element at a time (i.e. $\{(x, y)\}$) in a given data structure (i.e. R).

Example 2. From the fifth base goal in Figure 1 we get the following $\{log\}$ goals:

– EXPERIMENT 1.

```
R neq {} & S neq {} & dom(S,B) & dom(R,A) & ssubset(B,A).
```

– EXPERIMENT 2.

```
oplus(R,S,T) & R neq {} & S neq {} &
dom(S,B) & dom(R,A) & ssubset(B,A).
```

- Experiment 3.

```
npfun(R) & npfun(S) & R neq {} & S neq {} &
dom(S,B) & dom(R,A) & ssubset(B,A).
```

- Experiment 4.

```
oplus(R,S,T) & npfun(R) & npfun(S) & R neq {} & S neq {} &
dom(S,B) & dom(R,A) & ssubset(B,A).
```

- Experiment 5.

```
S = {[X,Y]} & R neq {} & S neq {} &
dom(S,B) & dom(R,A) & ssubset(B,A).
```

Table 1 shows a summary of data collected throughout the experiments. Each band lists the results of the five experiments for each of the relational operators discussed in this paper. For each operator the number of base goals is indicated. Running {*log*} on each goal yields one of four possible answers:

- success: means that the goal is satisfiable and {*log*} is able to find a solution for it;
- time out: {*log*} is not able to check satisfiability of the goal in the given time (10s);
- maybe: the goal *may be* satisfiable and the constraint returned by {*log*} *may* represent a solution for it, but it deserves to be confirmed by further rewritings. Such an answer comes from the fact that, if {*log*} is not able to check satisfiability of the goal in the given time, it tries to deactivate some rewrite rules (e.g. the elimination of $\neq$-constraints) in the hope to become capable of detecting possible inconsistencies. In fact, if the solver ends up in *false*, we can anyway conclude that the input constraint is unsatisfiable (since all applied rewrite rules are guaranteed to be correct); if, on the contrary, the solver ends up with a constraint in an irreducible form (but not the solved one), then we cannot conclude that the input constraint is surely satisfiable, but only that it *may be* satisfiable.
- false: the goal is unsatisfiable.

In Table 1, column # is the number of answers in each category and column T is the time, in seconds, spent by {*log*} in computing all these answers. ϵ means that the time is negligible (usually in the order of 10^{-2}s). At the end of the table the number of total answers in each category for each experiment are shown. As can be seen, {*log*} is able to compute meaningful answers for all the goals in experiments 1, 3 and 5. It also computes meaningful answers for all the goals in all the experiments for all the base constraints (i.e. *comp*, *dom* and *inv*), and *ran* and *rimg*. Observe, that when a meaningful answer is computed the computing time is negligible.

The experiments were ran on the following platform: Intel Core^{TM} i5-2410M CPU at 2.30GHz with 4 Gb of main memory, running Linux Ubuntu 12.04 (precise) of 32-bit with kernel 3.2.0-101-generic-pae. {*log*} 4.9.1-20 over SWI-Prolog

		Exp. 1		Exp. 2		Exp. 3		Exp. 4		Exp. 5	
Operator	Result	#	T	#	T	#	T	#	T	#	T
comp (8 goals)	success	8	ϵ	7	ϵ	5	ϵ	5	10	4	ϵ
	time out	0	0	0	0	0	0	0	0	0	0
	maybe	0	0	0	0	0	0	0	0	0	0
	false	0	0	1	ϵ	3	ϵ	3	ϵ	4	ϵ
dares (7 goals)	success	7	ϵ	6	10	6	ϵ	3	1	5	ϵ
	time out	0	0	1	13	0	0	3	39	0	0
	maybe	0	0	0	0	0	0	0	0	0	0
	false	0	0	0	0	1	ϵ	1	ϵ	2	ϵ
dom (4 goals)	success	4	ϵ	4	ϵ	1	ϵ	1	ϵ	1	ϵ
	time out	0	0	0	0	0	0	0	0	0	0
	maybe	0	0	0	0	0	0	0	0	0	0
	false	0	0	0	0	3	ϵ	3	ϵ	3	ϵ
dres (7 goals)	success	7	ϵ	6	ϵ	6	ϵ	6	2	5	ϵ
	time out	0	0	1	13	0	0	0	0	0	0
	maybe	0	0	0	0	0	0	0	0	0	0
	false	0	0	0	0	1	ϵ	1	ϵ	2	ϵ
inv (5 goals)	success	5	ϵ	5	ϵ	1	ϵ	1	ϵ	1	ϵ
	time out	0	0	0	0	0	0	0	0	0	0
	maybe	0	0	0	0	0	0	0	0	0	0
	false	0	0	0	0	4	3	4	ϵ	4	ϵ
oplus (8 goals)	success	8	ϵ	6	6	5	ϵ	1	ϵ	4	ϵ
	time out	0	0	2	26	0	0	4	53	0	0
	maybe	0	0	0	0	0	0	0	0	0	0
	false	0	0	0	0	3	ϵ	3	ϵ	4	ϵ
ran (5 goals)	success	5	ϵ	5	ϵ	2	ϵ	2	ϵ	1	ϵ
	time out	0	0	0	0	0	0	0	0	0	0
	maybe	0	0	0	0	0	0	0	0	0	0
	false	0	0	0	0	3	ϵ	3	ϵ	4	ϵ
rares (7 goals)	success	7	ϵ	6	8	6	ϵ	2	ϵ	5	ϵ
	time out	0	0	1	13	0	0	3	39	0	0
	maybe	0	0	0	0	0	0	1	10	0	0
	false	0	0	0	0	1	ϵ	1	ϵ	2	ϵ
rimg (7 goals)	success	7	ϵ	7	7	6	ϵ	6	3	5	ϵ
	time out	0	0	0	0	0	0	0	0	0	0
	maybe	0	0	0	0	0	0	0	0	0	0
	false	0	0	0	0	1	ϵ	1	ϵ	2	ϵ
rres (7 goals)	success	7	ϵ	7	7	6	ϵ	5	1	5	ϵ
	time out	0	0	0	0	0	0	0	0	0	0
	maybe	0	0	0	0	0	0	1	10	0	0
	false	0	0	0	0	1	ϵ	1	ϵ	2	ϵ
Totals (65 goals)	success	65		59		44		32		36	
	time out	0		5		0		10		0	
	maybe	0		0		0		2		0	
	false	0		1		21		21		29	

Table 1. Summary of experiments

7.2.3 for i386 was used during the experiments. All the experimental data can be found in the directory named `paper113/goals` located in the virtual machine delivered for "CAV 2016 Artifact Evaluation". A timeout of 10 seconds was set for each goal. The execution time was captured by inserting a `get_time(Tini)` predicate just before calling $\{log\}$ and a `get_time(Tend)` predicate right after it. Hence, the execution time is just `Tend - Tini`.

In summary, the results shown in Table 1 provide empirical evidence that the decision procedure presented in this paper and its implementation on $\{log\}$ are useful in practice. Nevertheless, surely there is room for performance improvements such as implementing special cases of the rewrite rules for some particular, recurring formulas.

References

[CAF$^+$14] Maximiliano Cristiá, Pablo Albertengo, Claudia S. Frydman, Brian Plüss, and Pablo Rodríguez Monetti. Tool support for the Test Template Framework. *Softw. Test., Verif. Reliab.*, 24(1):3–37, 2014.

[CRF15] Maximiliano Cristiá, Gianfranco Rossi, and Claudia S. Frydman. Adding partial functions to constraint logic programming with sets. *TPLP*, 15(4-5):651–665, 2015.

[Saa97] Mark Saaltink. The Z/EVES mathematical toolkit version 2.2 for Z/EVES version 1.5. Technical report, ORA Canada, 1997.

[SC96] P. Stocks and D. Carrington. A Framework for Specification-Based Testing. *IEEE Transactions on Software Engineering*, 22(11):777–793, November 1996.

[Spi92] J. M. Spivey. *The Z notation: a reference manual.* Prentice Hall International (UK) Ltd., Hertfordshire, UK, UK, 1992.