Technical Appendix to paper #8198: ASP-based Declarative Process Mining

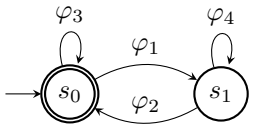

Figure 1: Automaton for constraint ϕ_1.

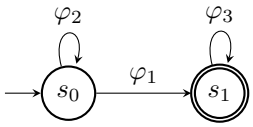

Figure 2: Automaton for constraint ϕ_2.

1 ASP encodings

We report three examples of Clingo ASP programs for Log Generation, Query Checking and Conformance Checking.

Event Log Generation

Consider the instance of Log Generation consisting of the set of constraints $\Phi = \{\phi_1, \phi_2\}$, with:

1. $\phi_1 = \mathbf{G}((a1 \wedge number < 5) \rightarrow Fa2)$;
2. $\phi_2 = F(a1 \wedge number < 3)$.

The automaton for ϕ_1 is depicted in Fig. 1, where:

- $\varphi_1 = a1 \wedge number < 5$;
- $\varphi_2 = a2$;
- $\varphi_3 = \neg\varphi_1$;
- $\varphi_4 = \neg\varphi_2$.

The automaton for ϕ_2 is depicted in Fig. 2, where:

- $\varphi_1 = (a1 \wedge number < 3)$;
- $\varphi_2 = \neg\varphi_1$;
- $\varphi_3 = \mathbf{true}$;

For the length of the trace(s) to return, we use the parameter t, which can be input provided when Clingo is launched. We report the actual code for Clingo below.


```
% PROBLEM INSTANCE

% Command line-parameters:
% t: trace length
(use option -c t=<value> to assign)

% Activities:
activity(a1). activity(a2). activity(a3).
activity(a4). activity(a5). activity(a6).
activity(a7). activity(a8). activity(a9).
activity(10).

% Activity attributes:
% has_attribute(activity, attribute name)
has_attribute(a1,categorical).
has_attribute(a1,number).

% Attribute types:
% value(attribute name, allowed value):
value(number,0..100).
value(categorical,c1). value(categorical,c2).
value(categorical,c3).

% Automata:
% Every automaton has associated a unique index
% initial(automaton index, initial state)
% accepting(automaton index, accepting state)
% transition(automaton index, src state,
%                     formula index, dest state).
% every formula has associated an index and
% is associated to an automaton
% (through automaton index)
% holds(automaton index, formula index,
%                               trace index)

% phi1=G((a1 and number<5)-> F a2)
initial(1,s0).
accepting(1,s0).
transition(1,s0,1,s1).
holds(1,1,T) :- trace(a1,T),
                has_value(number,V,T), V<5.
transition(1,s1,2,s0).
holds(1,2,T) :- trace(a2,T).
transition(1,s0,3,s0).
holds(1,3,T) :- not holds(1,1,T), time(T).
transition(1,s1,4,s1).
holds(1,4,T) :- trace(A,T), activity(A), A != a2.
```

```
% F a1 and number < 3
initial(2,s0).
accepting(2,s1).
transition(2,s0,1,s0).
holds(2,1,T) :- not holds(2,2,T), time(T).
transition(2,s0,2,s1).
holds(2,2,T) :- trace(a1,T),
has_value(number,V,T), V<3.
transition(2,s1,3,s1).
holds(2,3,T) :- time(T).

% Time points
time(0..t).

% PROBLEM ENCODING

% Exactly one activity per trace position
% (0,...,t-1)
{trace(A,T) : activity(A)} = 1 :- time(T), T<t.

% Exactly one assigned value per attribute
% associated with each activity
{has_value(K,V,T) : value(K,V)} = 1 :- trace(A,T),
has_attribute(A,K).

% Initial state
cur_state(I,S,0) :- initial(I,S).

% Transitions
cur_state(I,S',T) :- cur_state(I,S,T-1),
transition(I,S,F,S'), holds(I,F,T-1).

% Trace must be accepted by all automata
% Every automaton must be in at least one
% accepting state at end of run
:- cur_state(I,S,T): not accepting(I,S),
trace_length(T) = 0.

% Show solution
#show trace/2.
#show has_value/3.
```

Query Checking

As explained in the paper, the input of Query Checking includes many traces, each of which is uniquely identified by an index. We show the input for the following two traces:

- $\tau_1 = a2()\ a1(2, c3)\ a2()$;
- $\tau_2 = a1(2, c3)\ a1(1, c2)\ a1(2, c2)\ a2()$,

where the first attribute of activity $a1$ is named $number$ and the second is named $categorical$. Activity $a2$ does not contain any attribute. We remind that since many traces are present, a further index must be added, wrt the encoding for Log Generation, to various predicates, in order to indicate the trace that the predicate refers to. We denote such a predicate with the symbol `J`.

As input constraints, we use the following event formulas with activity variables:

- $\phi_1 = \mathbf{G}((?A1 \wedge number < 5) \rightarrow F?A2)$;
- $\phi_2 = F(?A1 \wedge number < 3)$.

The automaton corresponding to ϕ_1 is depicted in Fig. 1, with:

- $\varphi_1 = ?A1 \wedge number < 5$;
- $\varphi_2 = ?A2$;
- $\varphi_3 = \neg\varphi_1$;
- $\varphi_4 = \neg\varphi_2$.

The automaton corresponding to ϕ_2 is depicted in Fig. 2, where:

- $\varphi_1 = (?A1 \wedge number < 3)$;
- $\varphi_2 = \neg\varphi_1$;
- $\varphi_3 = \mathbf{true}$;

To account for the presence of many traces, predicate `holds`, which models satisfaction of event formulas at a given time point, contains an additional parameter representing the trace identifier. This allows to parameterize satisfaction of a formula wrt to the position of the trace.

Finally, as discussed in the paper, each variable $?A$ is modeled in the ASP program with object `varA` in predicate `variable`, while the assignment to a variable `varA` is modeled with the binary predicate `assignment(varA,val)`, where `val` is the value assigned to `varA`.

The resulting encoding is shown below.

```
% PROBLEM INSTANCE

% traces
% trace(id,activity,position)

% tau1=a2() a1(2,c3) a2()
trace(1,a2,0).
trace(1,a1,1).
has_value(1,integer,2,1).
has_value(1,categorical,c3,1).
trace(1,a2,2).
trace_length(1,3).

% tau2=a1(2,c3) a1(1,c2) a1(2,c2) a2()
trace(2,a1,0).
has_value(2,integer,2,0).
has_value(2,categorical,c3,0).
trace(2,a1,1).
has_value(2,integer,1,1).
has_value(2,categorical,c2,1).
trace(2,a1,2).
has_value(2,integer,1,2).
has_value(2,categorical,c2,2).
trace(2,a2,3).
trace_length(2,4).

% Activities
% Derived from trace
activity(A) :- trace(_,A,_).

% Activity attributes:
% has_attribute(activity, attribute name)
has_attribute(a1,categorical).
has_attribute(a1,number).

% Attribute types:
% value(attribute name, allowed value):
```

```
value(number,0..100).
value(categorical,c1).
value(categorical,c2).
value(categorical,c3).

% Automata encoding:
% Almost same as Log Generation
% differences: holds include trace id)
% holds(automaton index, formula index,
%          trace id, time point)

% G((?A1 and number<5) -> F ?A2)
variable(varA1).
variable(varA2).
initial(1,s0).
accepting(1,s0).
transition(1,s0,1,s1).
holds(1,1,J,T) :- trace(J,A,T), V<5,
  assignment(varA1,A), has_value(J,integer,V,T).
transition(1,s1,2,s0).
holds(1,2,J,T) :- trace(J,A,T),
                  assignment(varA2,A).
transition(1,s0,3,s0).
holds(1,3,J,T) :- not holds(1,1,J,T), time(J,T).
transition(1,s1,4,s1).
holds(1,4,J,T) :- not holds(1,2,J,T), time(J,T).

% F ?A1 and number < 3
initial(2,s0).
accepting(2,s1).
transition(2,s0,1,s0).
holds(2,1,J,T) :- not holds(2,2,J,T), time(J,T).
transition(2,s0,2,s1).
holds(2,2,J,T) :- trace(J,A,T), V<3,
  assignment(varA1,A), has_value(J,integer,V,T).
transition(2,s1,3,s1).
holds(2,3,J,T) :- time(J,T).

%time points
time(J,0..T) :- trace_length(J,T).

% PROBLEM ENCODING

% exactly one activity per trace point
{trace(I,A,T) : activity(A)} = 1 :- time(I,T),
                T < L, trace_length(I,L).

%initial state (J is trace identifier):
cur_state(I,J,S,0) :- initial(I,S), trace(J,_,_).

%transitions (J is trace identifier):
cur_state(I,J,S',T) :- cur_state(I,J,S,T-1),
        transition(I,S,F,S'), holds(I,F,J,T-1),
        trace(J,_,_).

%exactly one assigned value per attribute
has_value(J,K,V,T) : value(K,V) = 1 :-
        trace(J,A,T), has_attribute(A,K).

%exactly one assigned value per variable
{assignment(Var,Val) : activity(Val)} = 1 :-
                    variable(Var).

:- cur_state(I,J,S,L), trace_length(J,L),
   not accepting(I,S).

% solution
#show trace/3.
#show has_value/4.
#show assignment/2.
```

Conformance Checking

As explained in the paper, this is a special case of query checking where the constrains are variable free, i.e., they are fully specified. To obtain the encoding, it is enough to replace the encoding of the automata with one that is fully specified, and remove the predicates related to variables, i.e., `variable` and `assignment`. No predicates are returned as solution, as the instance is solved if the specification is satisfiable.

For completeness, we report the full ASP encoding for the set of traces used in the example of Conformance Checking and the constraints (automata) used for Log Generation.

```
% PROBLEM INSTANCE

% traces
% trace(id,activity,position)

% tau1=a2() a1(2,c3) a2()
trace(1,a2,0).
trace(1,a1,1).
has_value(1,integer,2,1).
has_value(1,categorical,c3,1).
trace(1,a2,2).
trace_length(1,3).

% tau2=a1(2,c3) a1(1,c2) a1(2,c2) a2()
trace(2,a1,0).
has_value(2,integer,2,0).
has_value(2,categorical,c3,0).
trace(2,a1,1).
has_value(2,integer,1,1).
has_value(2,categorical,c2,1).
trace(2,a1,2).
has_value(2,integer,1,2).
has_value(2,categorical,c2,2).
trace(2,a2,3).
trace_length(2,4).

% Activities
% Derived from trace
activity(A) :- trace(_,A,_).

% Activity attributes:
% has_attribute(activity, attribute name)
has_attribute(a1,categorical).
has_attribute(a1,number).

% Attribute types:
% value(attribute name, allowed value):
value(number,0..100).
value(categorical,c1).
value(categorical,c2).
value(categorical,c3).

% Automata encoding:
% Almost same as Log Generation
```

```
% differences: holds include trace id)
% holds(automaton index, formula index,
%          trace id, time point)

% G((a1 and number<5) -> F a2)
initial(1,s0).
accepting(1,s0).
transition(1,s0,1,s1).
holds(1,1,J,T) :- trace(J,a1,T), V<5,
  , has_value(J,integer,V,T).
transition(1,s1,2,s0).
holds(1,2,J,T) :- trace(J,a2,T).
transition(1,s0,3,s0).
holds(1,3,J,T) :- not holds(1,1,J,T), time(J,T).
transition(1,s1,4,s1).
holds(1,4,J,T) :- not holds(1,2,J,T), time(J,T).

% F a1 and number < 3
initial(2,s0).
accepting(2,s1).
transition(2,s0,1,s0).
holds(2,1,J,T) :- not holds(2,2,J,T), time(J,T).
transition(2,s0,2,s1).
holds(2,2,J,T) :- trace(J,a1,T), V<3,
  has_value(J,integer,V,T).
transition(2,s1,3,s1).
holds(2,3,J,T) :- time(J,T).

%time points
time(J,0..T) :- trace_length(J,T).

% PROBLEM ENCODING

% exactly one activity per trace point
{trace(I,A,T) : activity(A)} = 1 :- time(I,T),
                T < L, trace_length(I,L).

%initial state (J is trace identifier):
cur_state(I,J,S,0) :- initial(I,S), trace(J,_,_).

%transitions (J is trace identifier):
cur_state(I,J,S',T) :- cur_state(I,J,S,T-1),
        transition(I,S,F,S'), holds(I,F,J,T-1),
        trace(J,_,_).

%exactly one assigned value per attribute
has_value(J,K,V,T) : value(K,V) = 1 :-
        trace(J,A,T), has_attribute(A,K).

%exactly one assigned value per variable
{assignment(Var,Val) : activity(Val)} = 1 :-
                    variable(Var).

:- cur_state(I,J,S,L), trace_length(J,L),
   not accepting(I,S).
```

2 File Description

In the *encodings* folder you can find the actual encoding used for the experimental evaluation. Since for comparison purposes we focus on DECLARE, the constraints are directly referred by name. The ASP-based representations of the automata corresponding to the DECLARE constraints used in the experiments are in `templates.lp`. Note that, although the encodings of the data-aware version of the problems (denoted by `_data.lp` in the file name) can be used to solve the control-flow-only version as well in the folder is present also a simplified encoding tailored to this version.

In the *real_data* folder there are the 8 real dataset organized in folders. In each of them, together with the XES file there are also the discovered models in declare format and the ASP version of all such files. Note that while the models have the usual extension .lp, the log has extension .asp but is just a regular ASP file. This is done simply to make them easily recognizable and to facilitate the automation of the experiments with a script.

In the *synth_data* folder are present the 4 reference models used for log generation and conformance checking, together with their control-flow-only version. Note how in the latter is dealt with the local conditions to make possible to use the data-aware encoding. Again, it is possible to directly use the tailored encoding. In the folder are also present the logs of size 1000 and traces of varying length used for conformance checking and query checking. Finally, for query checking are present the files with a possible query, the relative conditions and the set of activities. File `activities.lp` needs to be used also for log generation with reference models since, being always the same, they were not included in the model itself.

In the *utils* folder are present files with functions for performing different useful tasks, like converting from .xes and .decl to ASP files. However this has already been performed for our datasets.

3 Experiment instructions

The experiment have been performed using the ASP solver clingo version 5.4.0. Please refer to github.com/potassco/clingo for downloading and installing clingo.

For executing log generation is required to pass to clingo a generation encoding file, a model file, the templates, the value of the parameter `t` corresponding to the length of the traces, and the number of desired answer sets, i.e the log size. For example you can run

```
clingo generation_encoding_data.lp
reference3_data.lp activities.lp
templates.lp -c t=10 10000
```

or

```
clingo generation_encoding.lp
2012_80.lp templates.lp -c t=10 10000
```

making sure that the files are in the current directory or to pass the appropriate path. The clingo output can then be converted into an XES file (see `lp2xes.py` in *utils*).

For executing conformance checking is required to pass to clingo a conformance encoding, a log, a model file, and the templates. For example you can run

Table 1: Number models' constraints

	BPI2012	DD	ID	PL	PTC	RP	RT	Sepsis
60	25	28	79	101	54	30	13	63
70	25	29	73	72	41	30	10	35
80	19	22	45	49	33	24	10	19
90	11	17	29	37	22	19	10	15

```
clingo conformance_encoding.lp
BPI_Challenge_2012.asp 2012_80.lp
templates.lp
```

It will return the unique answer set indicating the pairs trace/constraint such that the constraint is satisfied by the trace (and so the unsatisfied ones by exclusion).

For executing query checking is required to pass to clingo the query encoding, a log, a query, some conditions, the set of activities and the templates. For example you can run

```
clingo query_encoding_data.lp
length_10.lp query.lp conditions.lp
activities.lp templates.lp 0
```

where 0 instructs clingo to return all the assignments.

4 Dataset Specifications

While reference models and synthetic data allows to control variables like the number of constraints and the log size, and so to easily understand the different trends and how the presented approaches scale, a better grasp is also required for the real data in order to understand the effectiveness of such methods. To this end are reported here useful information about the real logs and the relative discovered models. In particular, in table 1 are shown the number of constraints for all the models (with minimum support varying from 60 to 90) of each log and in table 2 are shown information like the size of the logs and the average trace length.

Table 2: Logs Specifications

	BPI2012	DD	ID	PL	PTC	RP	RT	Sepsis
Size	13087	10500	6449	7065	2099	6886	150370	1050
Activities	23	17	34	51	29	19	11	16
Events	164506	56437	72151	86581	18246	36796	561470	15214
Min. trace len	3	1	3	3	1	1	2	3
Max trace len	96	24	27	90	21	20	20	185
Mean trace len	13	5	11	12	9	5	4	14

5 Result Tables

Finally are reported the tables for all the conducted experiments. In particular remember that for space constraints, in the paper is show only information from the last column of tables 3, 4, 5, 6, 15, 16 (log generation with size 10000), and from the last column of tables 7, 8, 9, 10 (conformance checking with synthetic data and 10 constraints).

Table 3: ASP Log generation, with data

Log size → / Trace len ↓		100	250	500	1000	2500	5000	10000
3	10	28	36	51	80	172	308	595
	15	33	47	70	114	240	457	876
	20	44	71	92	150	315	586	1132
	25	62	83	116	283	378	704	1364
	30	63	98	135	221	448	835	1642
5	10	27	37	53	88	175	323	614
	15	38	56	76	121	249	474	894
	20	52	76	105	162	324	598	1155
	25	64	95	132	202	397	731	1413
	30	84	126	147	238	476	879	1701
7	10	30	38	56	88	181	327	622
	15	43	57	84	125	260	472	904
	20	65	84	113	170	335	619	1178
	25	80	115	142	206	418	759	1444
	30	89	130	165	243	495	913	1746
10	10	40	49	62	94	191	346	654
	15	55	74	101	142	273	506	956
	20	70	107	132	184	369	662	1250
	25	94	140	155	229	449	816	1543
	30	113	151	194	280	540	975	1874
↑ # constraints								

Table 4: ASP Log generation, without data

Log size → / Trace len ↓		100	250	500	1000	2500	5000	10000
3	10	10	14	21	35	76	137	249
	15	12	18	27	48	107	184	349
	20	14	21	33	55	119	232	436
	25	18	27	40	73	153	270	519
	30	21	32	48	82	171	322	622
5	10	11	17	22	35	77	141	270
	15	15	21	31	51	107	203	390
	20	22	29	43	66	140	265	496
	25	24	33	48	76	161	293	568
	30	33	38	58	96	194	349	666
7	10	13	18	26	43	83	154	289
	15	18	26	37	61	128	223	408
	20	24	32	45	72	150	278	538
	25	29	40	55	89	177	327	611
	30	35	45	69	106	208	380	726
10	10	18	22	32	50	102	181	340
	15	24	34	51	76	138	248	457
	20	31	41	59	88	176	317	601
	25	38	52	75	110	218	381	712
	30	49	61	87	131	255	450	837
↑ # constraints								

Table 5: Alloy Log generation, with data

Log size → / Trace len ↓		100	250	500	1000	2500	5000	10000
3	10	1568	2408	3544	5377	10226	19082	35975
	15	1866	3058	4624	7035	13806	26182	50649
	20	2432	3837	5633	8839	19222	35286	69608
	25	2839	4407	6490	10559	23713	43606	85127
	30	3255	5020	7489	12029	26772	51534	101518
5	10	1551	2430	3585	5418	10485	19017	35786
	15	1977	3159	4766	7388	14509	27222	51534
	20	2554	4088	5966	9240	19564	35668	70342
	25	3033	4551	6653	10810	23273	43721	85598
	30	3695	5059	7570	12071	26943	51280	101882
7	10	1578	2492	3632	5582	10880	19395	36464
	15	2019	3323	4861	7486	14749	27452	54402
	20	2710	4242	6012	9314	19657	36552	73122
	25	3168	4573	6721	10876	23258	45096	87065
	30	3793	5275	7707	12627	27942	52492	106062
10	10	1636	2524	3697	5662	10673	20492	37688
	15	2221	3443	4999	7803	15893	28515	54749
	20	2764	4205	6151	9594	20393	37987	73222
	25	3333	4938	6987	11207	24102	45570	89210
	30	4027	5575	8127	13144	28890	54214	107520
↑ # constraints								

Table 6: Alloy Log generation, without data

Log size → / Trace len ↓		100	250	500	1000	2500	5000	10000
3	10	1039	1492	2122	3196	5646	10326	18733
	15	1195	1896	2754	4170	7635	13848	25700
	20	1510	2278	3365	5055	9392	17035	32047
	25	1736	2638	3836	5801	11050	20664	39114
	30	2091	3138	4484	6627	13392	24328	46207
5	10	1041	1564	2176	3191	5855	10633	18947
	15	1273	1995	2863	4190	7585	14289	25723
	20	1647	2472	3584	5162	9548	17659	33837
	25	1922	2865	4080	5919	11361	21187	38666
	30	2315	3371	4716	6931	13619	24678	46706
7	10	1084	1563	2257	3318	5978	10784	19539
	15	1390	2141	3102	4478	8284	14640	27344
	20	1686	2506	3619	5247	9637	17785	33107
	25	2065	2959	4018	5985	11406	21500	40556
	30	2404	3514	4645	7026	14110	24530	47613
10	10	1142	1596	2396	3496	6114	11066	20007
	15	1496	2149	3060	4435	8344	14789	26897
	20	1812	2672	3652	5272	9847	18164	33615
	25	2206	3178	4371	6436	12072	22371	41055
	30	2735	3827	4971	7393	14436	25851	49410
↑ # constraints								

Table 7: ASP Conformance Checking, with data

# constraints → / Trace Len ↓	3	5	7	10
10	390	453	526	665
15	523	671	892	1100
20	723	967	1151	1456
25	873	1190	1424	2071
30	1024	1439	1706	2407

Table 8: ASP Conformance Checking, without data

# constraints → / Trace Len ↓	3	5	7	10
10	375	436	537	635
15	517	644	854	1035
20	685	939	1084	1354
25	829	1157	1353	1896
30	1038	1356	1680	2219

Table 9: Declare Analyzer Conformance Checking, with data

# constraints → / Trace Len ↓	3	5	7	10
10	338	365	474	598
15	457	478	616	805
20	614	628	850	1092
25	744	796	1052	1273
30	669	698	997	1337

Table 10: Declare Analyzer Conformance Checking, without data

# constraints → / Trace Len ↓	3	5	7	10
10	65	77	126	110
15	82	105	132	145
20	96	117	130	155
25	127	135	172	177
30	140	162	181	215

Table 11: ASP Query Checking, with data

Constraints → / Trace len ↓	Existence	Responded Existence	Response	Chain Response	Absence	Not Resp. Existence	Not Resp.	Not Chain Response
10	521	736	534	503	566	783	602	385
15	704	1113	801	788	784	1180	879	606
20	1321	1675	1143	1128	1373	1821	1304	865
25	1397	3218	1528	1561	1562	2823	1807	1104
30	1674	2878	1824	1906	1905	2784	2028	1301

Table 12: ASP Query Checking, without data

Constraints → / Trace len ↓	Existence	Responded Existence	Response	Chain Response	Absence	Not Resp. Existence	Not Resp.	Not Chain Response
10	399	658	541	632	441	799	806	772
15	616	1183	824	1057	595	1319	1121	1182
20	903	1778	1339	1550	874	1887	2127	2062
25	1188	2381	1724	2036	1101	3246	3200	2486
30	1461	3278	2066	2632	1333	3391	2766	2846

Table 13: ASP Conf Checking

	BPI2012	DD	ID	PL	PTC	RP	RT	Sepsis
60	33426	13084	49969	78625	8412	9354	49501	7116
70	33242	13245	46388	55475	5596	9359	35537	3796
80	24482	10176	29969	33775	4699	6836	35483	1778
90	8445	4568	17576	26590	2787	5608	35483	731

Table 14: Declare Analyzer

	BPI2012	DD	ID	PL	PTC	RP	RT	Sepsis
60	2882	2771	9800	8521	1549	2122	15262	1194
70	2852	3249	7416	5358	959	2102	10351	705
80	2291	2103	3993	2677	755	1532	11285	318
90	1691	1525	1946	1595	404	1091	10628	250

Table 15: ASP Log Generator

Log size → / Trace len ↓		100	250	500	1000	2500	5000	10000
BPI2012	10	53	80	85	114	206	354	656
	15	103	107	133	170	288	420	817
	20	157	163	177	221	330	495	846
	25	183	200	239	268	402	626	1061
	30	238	254	296	356	533	848	1433
DD	10	59	84	100*	-	-	-	-
	15	91	104	129	182	359	568	887
	20	115	146	162	198	301	461	832
	25	167	168	200	245	352	549	930
	30	195	197	234	286	428	610	1026
ID	10	232	263	366	499	726*	-	-
	15	318	422	502	738	1141	1787	2865
	20	322	377	467	621	991	1707	3160
	25	460	530	619	784	1319	2183	4129
	30	773	866	993	1224	1904	3047	5226
PL	10	476	485	572	726	1189	2193	3901
	15	400	449	536	765	1340	2472	4538
	20	387	464	620	919	1796	2629	4102
	25	619	699	814	1143	2052	3656	6169
	30	697	829	982	1394	2700	4900	9231
PTC	10	131	160	196	246	455	722	1183
	15	155	188	265	388	678	1095	1820
	20	300	324	375	482	817	1309	2194
	25	283	365	454	659	1057	1787	2889
	30	668	702	725	824	1094	1520	2370
RP	10	71	102	119*	-	-	-	-
	15	106	130	164	204	358	593	1069
	20	151	152	185	225	334	484	813
	25	170	191	211	265	392	623	1063
	30	251	275	297	382	494	726	1220
RT	10	24	29	39	59	111	178	319
	15	33	40	65	69	118	193	353
	20	51	61	85	134	273	465	860
	25	60	67	95	117	172	284	483
	30	74	77	99	136	225	357	630
Sepsis	10	45	51	62	93	170	276	460
	15	68	83	96	128	219	320	564
	20	101	103	124	164	241	377	640
	25	129	139	154	217	301	456	780
	30	178	182	219	254	385	551	923

Table 16: Alloy Log Generator

Log size → / Trace len ↓		100	250	500	1000	2500	5000	10000
BPI2012	10	1011	1305	1944	3761	8470	15816	31935
	15	1221	1836	2897	4788	11664	21931	43337
	20	1372	1998	3249	6221	14874	29103	57596
	25	1402	2320	4017	7880	18024	35928	72383
	30	1543	2729	5313	9405	21426	44438	86910
DD	10	1221	1800	2364*	-	-	-	-
	15	3266	4152	5344	7639	16057	30010	58572
	20	9978	11188	12802	16745	27076	44709	80665
	25	24829	26626	28510	32505	47185	70960	118975
	30	53943	55751	57691	64371	81772	110264	181027
ID	10	3812	6148	8153	13103	27795	30762*	-
	15	13553	15473	19015	25266	45454	78753	152188
	20	41791	44977	49225	59342	87100	136461	237777
	25	106212	108716	114462	128149	164872	230086	359665
	30	232738	254063	260476	265864	312084	400863	563794
PL	10	1963	2669	4144	6892	15240	29841	59468
	15	4719	5786	7679	11730	23540	43630	85942
	20	12327	13893	16395	22094	36866	66240	122511
	25	28764	30272	34024	41347	63209	99740	174596
	30	63458	65117	69134	77547	103687	146897	236697
PTC	10	2214	2965	4716	7984	16928	31264	65783
	15	7245	9041	10581	14876	28198	50301	97098
	20	23212	24588	28192	33876	52246	83311	146420
	25	58128	60763	63891	72217	96504	139226	221434
	30	126922	129160	134175	146406	173855	233858	330753
RP	10	1220	1903	2703*	-	-	-	-
	15	3412	4264	5746	8554	17424	34520	66641
	20	10564	11924	13620	17477	29178	50220	95005
	25	26043	27746	30114	35546	50367	76615	134851
	30	59015	59834	62549	68361	88508	119339	187972
RT	10	605	927	1763	2618	6800	12394	24909
	15	690	1209	1981	3686	8745	16589	34408
	20	821	1414	2623	4736	10898	22526	44608
	25	1191	2995	3257	5900	14114	26784	54808
	30	1315	2407	3958	6843	15887	32672	63379
Sepsis	10	1408	1778	2537	4549	9671	19393	38241
	15	3364	4257	5371	8260	16083	29636	57178
	20	10087	11078	12758	16354	27661	45169	85808
	25	25084	26481	28736	34263	47217	71947	120110
	30	59024	59987	63132	68549	85823	115232	174838