Online appendix for the paper

Ticker: A System for Incremental ASP-based Stream Reasoning

published in Theory and Practice of Logic Programming

Harald Beck, Thomas Eiter, and Christian Folie

Institute of Information Systems, Vienna University of Technology
Favoritenstraße 9-11, A-1040 Vienna, Austria
{*beck,eiter*}*@kr.tuwien.ac.at, christian.folie@outlook.com*

Appendix A Notes on the Use of Clingo

Reactive features. We established techniques that allow for incrementally updating a program $\hat{P}_k$ for time or count increment, where Alg. 3 identifies at each tick new rules G^+ that have to be added to the previous translation, and expired ones G^- that must be deleted.

In search of existing systems that might allow such incremental program update, we considered the state-of-the-art ASP solver Clingo (Gebser et al. 2014), which comes with an API for reactive/multi-shot solving.[1] These functionalities are based on (Gebser et al. 2011), have since evolved (Gebser et al. 2012; Gebser et al. 2014) and successfully applied; e.g. viz. (Gebser et al. 2015). Unfortunately, for our purposes, control features in Clingo are not applicable.

First, the control features in Clingo allow addition of new rules, but not removal of existing ones. Technically, removing might be simulated by setting a designated switch atom to false. However, this approach would imply that the program grows over time. Second, we considered using reactive features as illustrated for Rule r of Ex. 5, using a program part that is parameterized for stream variables, including that of tick (t,c).

```
#program tick(t, c, v).
#external now(t).
#external cnt(c).
#external alpha_at(v,t).
high_at(t) :- w_time_2_alpha(v,t), t >= 18.
w_time_2_alpha(v,t) :- now(t), alpha_at(v,t).
w_time_2_alpha(v,t) :- now(t), alpha_at(v,t-1).
w_time_2_alpha(v,t) :- now(t), alpha_at(v,t-2).
```

However, this encoding is not applicable, since atoms in rule heads cannot be redefined, i.e., they cannot be grounded more than once.

Model update. For stratified programs (which have a unique model), repeatedly calling Clingo (by standard one-shot solving) on the encoded program $\hat{P}$ is a practical solution. However, when a program has multiple models, we then have no link between the output of successive ticks, i.e., the model may arbitrarily change. For instance, consider program

```
a :- not b, not c.  b :- not a, not c.  c :- not a, not b.
```

[1] Clingo 5.1.0. API: `https://potassco.org/clingo/python-api/current/clingo.html`

Using Clingo 5.1.0, the answer set of the program that is returned first is {a}, which remains an answer set if we add rule `a :- not c`. However, the first reported answer set now is {c}.

Appendix B Proofs

Proof for Lemma 1

Let $S = (T, \upsilon)$ be a stream that underlies tick stream $\dot{S} = (K, v)$, such that $K = \langle (t_1, c_1), \ldots, (t_m, c_m) \rangle$. By definition, $T = [t_1, t_m]$ and $\upsilon(t) = \bigcup\{v(t,c) \mid (t,c) \in K\}$ for all $t \in T$. We recall that $\tau_n(S)$ (resp. $\tau_n(\dot{S})$) abbreviates $\tau_n(S, t_m)$ (resp. $\tau_n(\dot{S}, (t_m, c_m))$. Thus, by definition, $\tau_n(\dot{S}) = (K', v|_{K'})$, where $K' = \{(t', c') \in K \mid \max\{t_1, t-n\} \leq t' \leq t\}$, and $\tau_n(S) = (T', \upsilon|_{T'})$, where $T' = [t', t_m]$ and $t' = \max\{t_1, t-n\}$. We observe that t' is the minimal time point selected also in K', i.e., $K' = \langle (t_k, c_k), \ldots, (t_m, c_m) \rangle$ implies $t_k = t'$. It remains to show that $(\upsilon|_{T'})(t) = \bigcup\{(v|_{K'})(t,c) \mid (t,c) \in K'\}$ for all $t \in T'$. This is seen from the fact that neither $\tau_n(S)$ nor $\tau_n(\dot{S})$ drops any data within T'. We conclude that $\tau_n(S)$ underlies $\tau_n(\dot{S})$.

Proof Sketch for Lemma 2

The argument is similar as for Lemma 1. The central observation is that a tick stream provides a more fine-grained control over the information available in streams by introducing an order on tuples in addition to the temporal order. Each time point in a stream is assigned a set of atoms, whereas each tick in a tick stream is assigned at most one atom. The tuple-based window function $\#_n$ always counts atoms backwards (from right end to left) and then selects the timeline $[t_1, t]$ with the latest possible left time point t_1 required to capture n atoms. While for tick streams, the order is unique, but multiple options exist for streams in general. If the tuple window $\#_n(S)$ is based on the order in which atoms appeared in S, then it selects the same atoms as $\#_n(\dot{S})$, and thus the same timeline. Consequently, $\#_n(S)$ underlies $\#_n(\dot{S})$.

Proof Sketch for Proposition 1

The desired correspondence is based on two translations: a LARS program P (at a time t) into a logic program $\hat{P} = \mathit{LarsToAsp}(P, t)$ (due to Algorithm 1), and the encoding of a stream S as set $\hat{S}$ of atoms. Given a fixed timeline T, we may view a stream $S = (T, \upsilon)$ as a set of pairs $\{(a(\vec{x}), t) \mid a(\vec{x}) \in \upsilon(t), t \in T\}$. This is the essence of a stream encoding $\hat{S}$ for the tick stream $\dot{S} = (K, v)$; $\hat{S}$ includes the analogous time-pinned atoms: $\{a_@(\vec{x}, t) \mid a(\vec{x}) \in v(t,c), (t,c) \in K\}$. With respect to the correspondence, atoms of form $a_\#(\vec{x}, t, c)$, $\mathit{cnt}(c)$ and $\mathit{tick}(t,c)$ in $\hat{S}$ can be considered auxiliary, as well as the specific counts used in the tick pattern K to obtain time-pinned atoms $a_@(\vec{x}, t)$. Counts play a role only for the specific selection of tuple-based windows, which are assumed to reflect the order of the tick stream. Thus, we may view a stream encoding $\hat{S}$ essentially as a different representation of stream S; additional atoms can be abstracted away as they have no correspondence in the original LARS stream or program. We thus consider only the time-pinned atoms in an encoded stream to read off a LARS stream.

Thus, it remains to argue the soundness of the transformation *LarsToAsp*, which returns a program of form $Q \cup R \cup \{\mathit{now}(t)\}$, where $\mathit{now}(t)$ is auxiliary. The set Q simply identifies time-pinned atoms $a_@(\vec{X}, \dot{N})$ with $a(\vec{X})$ in case $\dot{N}$ is the current time point. This is the information provided by predicate *now* for which a unique atom exists. Thus, Q ensures that a time-pinned

atom $a_@(\vec{x},t)$ is available if $a(\vec{x},t)$ is derived, and vice versa; Q thereby only accounts for redundant representations of atoms that currently hold.

Towards R, we get the translation by the function *larsToAspRules* which returns a set of encoded rules for every LARS rule r. First, the *baseRule* is the corresponding ASP rule, which introduces a new symbol $atm(e)$ for every extended atom in the rule that is not an ordinary atom. In order to ensure that the base rule $\hat{r}$ fires in an interpretation just if the original rule r fires in the corresponding interpretation of program P, for each body element $atm(e)$ in $\hat{r}$ the set of rules to derive $atm(e)$ in lines (14)-(21) is provided; the correspondence between $@_T a(\vec{X})$ and $a_@(\vec{X},T)$ is already given by construction. Thus, each interpretation stream $I \supseteq D$ for P has a corresponding interpretation $\hat{I}$ for $LarsToAsp(P)$ in which besides the time-pinned atoms the atoms $atm(e)$ and $spoil_e(\vec{X})$ occur depending on support from (i.e., firing) of the rules in (14)-(21), such that they correctly reflect the value of the window atoms e in I.

As each atom in an answer of an ordinary ASP program must derived by a rule, it is not hard to see that every answer set of $\hat{P} = LarsToAsp(P,t) \cup \hat{D}$ is of the form $\hat{I}$, where $I \supseteq D$ is an interpretation stream for D. We thus need to show the following: $I \in AS(P,D,t)$ holds iff $\hat{I}$ is an answer set of $\hat{P}$. We do this for ground P (the extension to non-ground P is straightforward).

($\Rightarrow$) For the only-if direction, we show that if $I \in AS(P,D,t)$, that is, I is a minimal model for the reduct $P^{M,t}$ where $M = \langle I, W, B\rangle$, then (i) $\hat{I}$ is a model of the reduct $\hat{P}^{\hat{I}}$, and (ii) no interpretation $J' \subset \hat{I}$ is a model of $\hat{P}^{\hat{I}}$. As for (i), we can concentrate by construction of $\hat{I}$ on the base rules $\hat{r} = baseRule(r)$ in $\hat{P}^{\hat{I}}$ (all other rules will be satisfied). If $\hat{I}$ satisfies $B(\hat{r})$, then by construction I satisfies $B(r)$; as I is a model of $P^{M,t}$, it follows that I satisfies $H(r)$; but then, by construction, $\hat{I}$ satisfies $H(\hat{r})$. As for (ii), we assume towards a contradiction that some $J' \subset \hat{I}$ satisfies $\hat{P}^{\hat{I}}$. We then consider the stream $J \supseteq D$ that is induced by J', and any rule r in the reduct $P^{M,t}$. If J does not satisfy $B(r)$, then J satisfies r; otherwise, if J satisfies $B(r)$, then as $\hat{r}$ is in the reduct $\hat{P}^{\hat{I}}$, we have that $\hat{I}$ falsifies each atom $atm(e)$ in $B^-(\hat{r})$, and as $J' \subset \hat{I}$, also J' falsifies each such $atm(e)$. Furthermore, as J satisfies each atom $e \in B^+(r)$, from the rules for $atm(e)$ among (14)-(21) in the reduct $\hat{P}^{\hat{I}}$ we obtain that J' satisfies each atom $atm(e)$ in $B^+(\hat{r})$. That is, J' satisfies $B(\hat{r})$. As J' satisfies $\hat{r}$, we then obtain that J' satisfies $H(\hat{r})$. The latter means that J satisfies $H(r)$, and thus J satisfies r. As r was arbitrary from the reduct $P^{M,t}$, we obtain that $J \subset I$ is a model of $P^{M,t}$; this however contradicts that I is a minimal model of $P^{M,t}$, and thus (ii) holds.

($\Leftarrow$) For the if direction, we argue similarly. Consider an answer set $\hat{I}$ of $\hat{P}$. To show that $I \in AS(P,D,t)$, we establish that (i) I is a model of $P^{M,t}$ and (ii) no model $J \subset I$ of $P^{M,t}$ exists. As for (i), since in $\hat{I}$ the atoms $atm(e)$ correctly reflect the value of the window atoms e in I, for each r in $P^{M,t}$ the rule $\hat{r} = baseRule(r)$ is in $\hat{P}^{\hat{I}}$; as $\hat{I}$ satisfies $\hat{r}$, we conclude that I satisfies r. As for (ii), we show that every model J of $P^{M,t}$ must contain I, which then proves the result.

To establish this, we use the fact that $\hat{I}$ can be generated by a sequence $\rho = r_1, r_2, r_3 \ldots, r_k$ of rules from $\hat{P}^{\hat{I}}$ with distinct heads such that (a) $\hat{I} = \{H(r_1), \ldots H(r_k)\} =: \hat{I}_k$ and (b) $\hat{I}_{i-1} = \{H(r_1), \ldots, H(r_{i-1})\}$ satisfies $B^+(r_i)$, for every $i = 1, \ldots, k$.

In that, we use the assertion that no cyclic positive dependencies through time-based window atoms $\boxplus^n \Box a$ occur. Formally, positive dependency is defined as follows: an atom $@_{t_1} b$ positively depends on an atom $@_{t_2} a$ in a ground program P at t, if some rule $r \in P$ exists with $H(r) = @_{t_1} b$ and such that either (a) $@_{t_2} a \in B^+(r)$, or (b) $\boxplus^n @_{t_2} a \in B^+(r)$ or (c) $\boxplus^n \star a \in B^+(r)$, $\star \in \{\Box, \Diamond\}$, where in (b) and (c) $t_2 \in [t-n, t]$ holds. As in $LarsToAsp(P,t)$, all ordinary atoms a are here viewed as $@_t a$. A cyclic positive dependency through $\boxplus^n \Box a$ is then a sequence $@_{t_0} a_0$, $@_{t_1} a_1$, $\ldots$, $@_{t_k} a_k$, $k \geq 1$, such that $@_{t_i} a_i$ positively depends on $@_{t_{(i+1) \mod k}} a_{(i+1) \mod k}$, for all $i = 0, \ldots, k$ and $a_0 = b$ and $a_1 = a$ for case (c) with $\star = \Box$.

Given that no positive cyclic dependencies through atoms $\boxplus^n \Box a$ occur in P at t, and thus in $P^{M,t}$, we can w.l.o.g. assume that whenever r_i in ρ has a head ω_e for a window atom $e = \boxplus^n \Box a$, each rule r_j in ρ with a head $a_@(t')$, where $t' \in [t-n,t]$, precedes r_i, i.e., $j < i$ holds.

By induction on $i \geq 1$, we can now show that if $H(r_i) = atm(e)$, then every model J of $P^{M,t}$ must satisfy e; consequently, at $i = k$, J must contain I. From the form of the rules $baseRule(r)$ and $windowRules(e)$, the correspondence between $\hat{P}^{\hat{I}}$ and $P^{M,t}$, and the fact that the external data are facts, only the case $e = \boxplus^n \Box a(\vec{X})$ needs a further argument. Now if r_i is the rule $\omega_e \leftarrow a(\vec{X}), \text{not } spoil_e(\vec{X})$ on line (16), then $\hat{I}$ must satisfy a and falsify $spoil_e(\vec{X})$; in turn, every $a_@(t', \vec{X})$ must be true in $\hat{I}$, for $t' \in [t-n,t]$. From the induction hypothesis, we obtain that $@_{t'} a(\vec{X})$ is true in every model J of $P^{M,t}$, $t' \in [t-n,t]$, and thus $e = \boxplus^n \Box a(\vec{X})$ is true as well. This proves the claim and concludes the proof of the if-case, which in turn establishes the claimed correspondence between $AS(P,D,t)$ and the answer sets of $\hat{P} = LarsToAsp(P,t) \cup \hat{D}$.

Remark. The condition on cyclic positive dependencies excludes that rules $b \leftarrow \boxplus^n \Box a$ and $a \leftarrow b$ occur jointly in a program. A stricter notion of dependency that allows for co-occurrence is to request in (c) for $\star = \Box$ in addition $t_2 < t$; then e.g. any LARS program where the rule heads are ordinary atoms is allowed, and Proposition 1 remains valid.

Proof Sketch for Proposition 2

Assume a LARS program P and two tick data streams $D = (K, v)$ and $D' = (K', v')$ at tick (t_m, c_m) such that $D' \subseteq D$ and $K' = \langle (t_k, c_k), \ldots, (t_m, c_m) \rangle$. Furthermore, assume that (*) all atoms/time points accessible from any window in P are included in D'. We want to show $AS^I(\hat{P}_{D,m}) = AS^I(\hat{P}_{D',m})$. The central observation is that rules need to fire in order for intensional atoms to be included in the answer set, and that no rules can fire based on outdated ticks. Thus, these ticks can also be dropped.

In more detail, we assume $AS^I(\hat{P}_{D,m}) \neq AS^I(\hat{P}_{D',m})$ towards a contradiction. That is to say, a difference in evaluation arises based on data in $D \setminus D'$, i.e., atoms appearing before tick (t_k, c_k). Consider any extended atom e of a (LARS) rule $r \in P$, where the body holds only for one of the two encodings (in the same partial interpretation). Due to the assumption (*), we can exclude a difference arising from a window atom of form $\boxplus^w \star a$, $\star \in \{\Diamond, \Box, @_T\}$.

If e is an atom a, it holds in $\hat{P}_{D,m}$ iff it holds in $\hat{P}_{D',m}$ since an ordinary atom in the answer set of the encoding corresponds to an atom holding at the current time point, and both D and D' include the current time point.

The last option is $e = @_T a$, which may reach back beyond (t_k, c_k) but is viewed in the incremental encoding as syntactic shortcut for $\boxplus^\infty @_T a$. That is, in this case we have $D' = D$ and thus the encodings coincide.

We conclude that assuming $AS^I(\hat{P}_{D,m}) \neq AS^I(\hat{P}_{D',m})$ is contradictory due to these observations. Spelling out the details fully involves essentially a case distinction on the incremental window encodings and arguing about the relationship between (t_k, c_k), the respective expiration annotations, and the fact that rules accessing atoms at ticks before (t_k, c_k) are have already expired.

Proof Sketch for Proposition 3

We argue based on the commonalities and differences of the static encoding $\hat{P} \cup \hat{D}$ and the incremental encoding $\hat{P}_{D,m}$. Instead of body predicates $now(\dot{N})$ and $cnt(\dot{C})$, that are instantiated in

$\hat{P} \cup \hat{D}$ due to the predicates $now(t)$ and $cnt(c)$, $\hat{P}_{D,m}$ directly uses the instantiations of tick variables. In both encodings, the window atom is associated with a set of rules that needs to model the temporal quantifier ($\Diamond$,$\Box$,$@_t$) in the correct range of ticks as expressed by the LARS window atom. This window always includes the last tick. While $\hat{P} \cup \hat{D}$ is based on a complete definition how far the window extends, $\hat{P}_{D,m}$ updates this definition tick by tick. In particular, the oldest tick that is not covered by the window anymore corresponds to the expiration annotation in $\hat{P}_{D,m}$.

The case $\boxplus^n \Diamond a(\vec{X})$ is as follows: in the static rule encoding,

$$\omega_e(\vec{X}) \leftarrow now(\dot{N}), a_@(\vec{X}, T),$$

given $now(t)$, time variable T will be grounded with $t-n, \ldots, t-0$. That is, we get a set of rules

$$\begin{array}{lrcl} (r_0) & \omega_e(\vec{X}) & \leftarrow & now(t), a_@(\vec{X}, t) \\ & & \vdots & \\ (r_n) & \omega_e(\vec{X}) & \leftarrow & now(t), a_@(\vec{X}, t-n), \end{array}$$

where arguments $\vec{X}$ will be grounded due to data and inferences. We observe that (r_0) is the rule that is inserted to the incremental program $\hat{P}_{D,m}$ at time t (minus predicate $now(t)$, since in $\hat{P}_{D,m}$ variable T is instantiated directly with t to obtain $a_@(\vec{X}, t)$), and all rules up to r_n remain from previous calls to *IncrementalRules*. Rule r_n will expire at $t+1$, i.e., the exact time when it will not be included in $\hat{P} \cup \hat{D}$ anymore. The cases for $\boxplus^n @_T a(\vec{X}), \boxplus^n \Box a(\vec{X}), \boxplus^{\#n} \Diamond a(\vec{X})$ and $\boxplus^{\#n} @_T a(\vec{X})$ are analogous; the remaining case $\boxplus^{\#n} \Box a(\vec{X})$ has been argued earlier.

Finally, $\hat{P}_{D,m}$ includes a stream encoding, which is also incrementally maintained: at each tick (t,c) the tick atom $tick(t,c)$ is added, and in case of a count increment, the time-pinned atom $a_@(\vec{X}, t)$ and the tick-pinned atoms $a_\#(\vec{X}, t, c)$ are added to $\hat{P}_{D,m}$ as in $\hat{D}$. This way, we have a full correspondence with the static stream encoding $\hat{D}$.

Thus, at every tick (t,c), $\hat{P} \cup \hat{D}$ and $\hat{P}_{D,m}$ have the same data and express the same evaluations. Disregarding auxiliary atoms, we conclude that their answer sets coincide.

Proof Sketch for Theorem 1

Given a LARS program P, a tick data stream $D = (K, v)$ at tick (t,c) by Prop. 1 S is an answer stream of P for D at t iff $\hat{S}$ is an answer set of $\hat{P} \cup \hat{D}$, where $\hat{P} = LarsToAsp(P,t)$. By Prop. 3, for any set X we have that $X \cup \{now(t), cnt(c)\}$ is an answer set of $\hat{P} \cup \hat{D}$ iff X is an answer set of $\hat{P}_{D,m}$ (modulo auxiliary atoms). In particular this holds for $X = \hat{S}$. As $\{now(t), cnt(c)\} \subseteq \hat{S}$, we obtain that S is an answer stream of P for D at t iff $\hat{S}$ is an answer set of $\hat{P}_{D,m}$, which is the result.

Appendix C Details of Evaluation Results

(See pages 6–7.)

Table C 1. *Results for* A1. *Variable window size n. Results for 1000 timepoints in seconds.*

	Clingo			Incremental		
n	t_{total}	t_{init}	t_{tick}	t_{total}	t_{init}	t_{tick}
20	14.296	0.017	0.014	2.638	0.016	0.002
40	20.526	0.018	0.02	3.006	0.018	0.002
80	34.491	0.025	0.034	2.938	0.018	0.002
120	49.249	0.027	0.049	3.439	0.019	0.003
160	64.661	0.028	0.064	3.554	0.017	0.003
200	79.105	0.036	0.079	3.674	0.018	0.003

Table C 2. *Results for* A2. *Variable window size n. Runtime for 1000 timepoints in seconds.*

	Clingo			Incremental		
n	t_{total}	t_{init}	t_{tick}	t_{total}	t_{init}	t_{tick}
20	15.259	0.02	0.015	2.869	0.016	0.002
40	23.123	0.02	0.023	3.201	0.018	0.003
80	35.962	0.022	0.035	3.365	0.019	0.003
120	49.068	0.026	0.049	3.547	0.02	0.003
160	61.983	0.03	0.061	3.842	0.018	0.003
200	80.899	0.036	0.08	3.7	0.019	0.003

Table C 3. *Results for* A1. *Variable timepoints tp. Runtime for window size* $n = 60$ *in seconds.*

	Clingo			Incremental		
tp	t_{total}	t_{init}	t_{tick}	t_{total}	t_{init}	t_{tick}
100	2.78	0.026	0.027	0.368	0.023	0.003
200	5.49	0.022	0.027	0.674	0.02	0.003
300	8.269	0.022	0.027	1.072	0.026	0.003
400	11.379	0.026	0.028	1.307	0.02	0.003
500	14.192	0.024	0.028	1.695	0.017	0.003
600	16.709	0.023	0.027	1.945	0.02	0.003
700	20.049	0.021	0.028	2.217	0.017	0.003
800	22.534	0.021	0.028	2.627	0.018	0.003
900	25.892	0.024	0.028	3.183	0.022	0.003
1000	27.501	0.021	0.027	3.42	0.021	0.003

Table C 4. *Results for* A2. *Variable timepoints tp. Runtime for window size* $n = 60$ *in seconds.*

	Clingo			Incremental		
tp	t_{total}	t_{init}	t_{tick}	t_{total}	t_{init}	t_{tick}
100	2.998	0.026	0.029	0.418	0.019	0.003
200	5.727	0.023	0.028	0.89	0.017	0.004
300	9.06	0.026	0.03	1.097	0.021	0.003
400	11.783	0.021	0.029	1.563	0.02	0.003
500	14.26	0.021	0.028	1.81	0.017	0.003
600	17.439	0.02	0.029	2.181	0.021	0.003
700	20.321	0.021	0.028	2.438	0.018	0.003
800	23.3	0.02	0.029	3.371	0.02	0.004
900	26.51	0.021	0.029	3.22	0.018	0.003
1000	30.077	0.024	0.03	3.5	0.019	0.003

Table C 5. *Results for* B1. *Variable window size n. Runtime in seconds for 1000 timepoints.*

	Clingo			Incremental		
n	t_{total}	t_{init}	t_{tick}	t_{total}	t_{init}	t_{tick}
20	26.158	0.018	0.026	15.641	0.292	0.015
40	55.898	0.021	0.055	16.726	0.315	0.016
80	425.853	0.019	0.425	21.135	0.299	0.02
120	-	-	-	25.909	0.304	0.025
160	-	-	-	30.659	0.363	0.03
200	-	-	-	33.541	0.306	0.033

Table C 6. *Results for* B2. *Variable window size n. Runtime for 1000 timepoints in seconds.*

	Clingo			Incremental		
n	t_{total}	t_{init}	t_{tick}	t_{total}	t_{init}	t_{tick}
20	24.138	0.018	0.024	34.717	0.292	0.033
40	38.478	0.019	0.038	35.744	0.333	0.034
80	71.827	0.024	0.071	25.767	0.298	0.025
120	104.723	0.023	0.104	33.788	0.29	0.033
160	148.257	0.031	0.148	31.1	0.303	0.03
200	181.991	0.028	0.181	37.612	0.33	0.036

Table C 7. *Results for* B1. *Variable timepoints tp. Window size* $n = 60$ *in seconds.*

	Clingo			Incremental		
tp	t_{total}	t_{init}	t_{tick}	t_{total}	t_{init}	t_{tick}
100	8.57	0.026	0.085	1.895	0.32	0.015
200	16.392	0.022	0.081	3.971	0.293	0.018
300	31.568	0.022	0.105	6.82	0.475	0.021
400	40.927	0.025	0.102	8.518	0.332	0.02
500	55.313	0.021	0.11	10.64	0.351	0.02
600	69.548	0.021	0.115	12.816	0.353	0.02
700	-	-	-	15.773	0.333	0.021
800	-	-	-	16.756	0.318	0.02
900	-	-	-	16.96	0.298	0.018
1000	-	-	-	18.602	0.298	0.018

Table C 8. *Results for* B2. *Variable timepoints tp. Window size* $n = 60$ *in seconds.*

	Clingo			Incremental		
tp	t_{total}	t_{init}	t_{tick}	t_{total}	t_{init}	t_{tick}
100	4.974	0.029	0.049	1.838	0.299	0.015
200	10.06	0.021	0.05	3.982	0.304	0.018
300	15.023	0.02	0.049	6.126	0.359	0.019
400	20.574	0.019	0.051	9.062	0.29	0.021
500	26.075	0.02	0.052	11.625	0.289	0.022
600	31.68	0.02	0.052	14.974	0.297	0.024
700	36.35	0.02	0.051	18.301	0.29	0.025
800	42.391	0.021	0.052	22.947	0.286	0.028
900	48.254	0.021	0.053	28.979	0.366	0.031
1000	54.35	0.02	0.054	28.993	0.334	0.028

References

GEBSER, M., GROTE, T., KAMINSKI, R., OBERMEIER, P., SABUNCU, O., AND SCHAUB, T. 2012. Stream reasoning with answer set programming: Preliminary report. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference, KR 2012, Rome, Italy, June 10-14, 2012*, G. Brewka, T. Eiter, and S. A. McIlraith, Eds. AAAI Press.

GEBSER, M., GROTE, T., KAMINSKI, R., AND SCHAUB, T. 2011. Reactive answer set programming. In *Logic Programming and Nonmonotonic Reasoning - 11th International Conference, LPNMR 2011, Vancouver, Canada, May 16-19, 2011. Proceedings*, J. P. Delgrande and W. Faber, Eds. Lecture Notes in Computer Science, vol. 6645. Springer, 54–66.

GEBSER, M., KAMINSKI, R., KAUFMANN, B., AND SCHAUB, T. 2014. *Clingo* = ASP + control: Preliminary report. In *Technical Communications of the Thirtieth International Conference on Logic Programming (ICLP'14)*, M. Leuschel and T. Schrijvers, Eds. Vol. arXiv:1405.3694v1. Theory and Practice of Logic Programming, Online Supplement.

GEBSER, M., KAMINSKI, R., OBERMEIER, P., AND SCHAUB, T. 2015. Ricochet robots reloaded: A case-study in multi-shot ASP solving. In *Advances in Knowledge Representation, Logic Programming, and Abstract Argumentation - Essays Dedicated to Gerhard Brewka on the Occasion of His 60th Birthday*, T. Eiter, H. Strass, M. Truszczynski, and S. Woltran, Eds. Lecture Notes in Computer Science, vol. 9060. Springer, 17–32.