Splitting a Hybrid ASP Program

Alex Brik
Google Inc., USA

Hybrid Answer Set Programming (Hybrid ASP) is an extension of Answer Set Programming (ASP) that allows ASP-like rules to interact with outside sources. The Splitting Set Theorem is an important and extensively used result for ASP. The paper introduces the Splitting Set Theorem for Hybrid ASP, which is for Hybrid ASP the equivalent of the Splitting Set Theorem, and shows how it can be applied to simplify computing answer sets for Hybrid ASP programs most relevant for practical applications.

An important result for logic programs is the Splitting Set Theorem [12], which shows how computing an answer set for a program can be broken into several tasks of the same kind for smaller programs. The theorem and its more general variant the Splitting Sequence Theorem are extensively used for proving other theorems, for instance in [1], [9] or [3] among many others. Hybrid Answer Set Programming (Hybrid ASP) [4] is an extension of ASP that allows ASP-like rules to interact with outside sources, which makes Hybrid ASP well suited for practical applications. For instance, recently Hybrid ASP has been used in a system for diagnosing failures of data processing pipelines at Google Inc [8]. The theory of Hybrid ASP, however is not extensively developed. This paper introduces the Splitting Set Theorem for Hybrid ASP and the Splitting Sequence Theorem for Hybrid ASP, which are the equivalents for Hybrid ASP of the similarly named results for ASP, thus making a small step towards developing the theory of Hybrid ASP. The author hopes that the new theorems will have many future applications, in the way analogous to the original Splitting Set Theorem and Splitting Sequence Theorem. The potential of the new theorems to be useful in the future, and the significance of the new results is demonstrated by using them to simplify computation of answer sets for the types of Hybrid ASP programs most relevant for practical applications, i.e. those applications that have answer sets with states having times of the form $k \cdot \Delta t$, such as the programs that result from translating descriptions in action languages Hybrid AL [7] and Hybrid ALE [2], or such as the programs used in other applications of Hybrid ASP [6], [5].

The paper is structured as follows. The first section reviews ASP, The Splitting Set Theorem and Hybrid ASP. The paper then presents The Splitting Set Theorem for Hybrid ASP and The Splitting Sequence Theorem for Hybrid ASP. The following section presents an algorithm that simplifies computing answer sets for Hybrid ASP. Finally a short conclusion follows.

1 Review of the Splitting Set Theorem and Hybrid ASP

We will begin with a brief review of ASP. Let At be a nonempty set of symbols called *atoms*. A *block* is an expression of the form

$$b_1, ..., b_k,\ not\ b_{k+1},\ ...,\ not\ b_{k+m} \tag{1}$$

where $b_1,\ ...,\ b_{k+m}$ are atoms. For a block B as above, let the *set of atoms of* B be defined as $At\,(B) \equiv \{b_1,\ ...,\ b_{k+m}\}$. $B^+ \equiv b_1,\ ...,\ b_k$ is called the *positive part of* B, and $B^- \equiv not\ b_{k+1},\ ...,\ not\ b_{k+m}$ is called the *negative part of* B. A set operation applied to a block B will indicate the same set operation applied to $At\,(B)$ with the block being reconstructed from the result of the set operation. For instance $b_1,\ b_2,\ not\ b_3,\ b_4 \setminus \{b_1,\ b_4\}$ will indicate a block $b_2,\ not\ b_3$.

F. Ricca, A. Russo et al. (Eds.): Proc. 36th International Conference on Logic Programming (Technical Communications) 2020 (ICLP 2020)
EPTCS 325, 2020, pp. 21–34, doi:10.4204/EPTCS.325.8

A *normal propositional logic programming* rule is an expression of the form

$$r \equiv a : - B \tag{2}$$

where a is an atom and B is a block. We define the *head of* r as $head(r) \equiv a$, and we define the *body of* r as $body(r) \equiv B$. We define $At(r) \equiv \{a\} \cup At(B)$.

Given any set $M \subseteq At$ and a block B, we say that M satisfies B, written $M \models B$, if $At(B^+) \subseteq M$ and $At(B^-) \cap M = \emptyset$. For a rule r, we say that M satisfies r, written $M \models r$, if whenever M satisfies the body of r, then M satisfies the head of r. A *normal logic program* P is a set of rules. We say that $M \subseteq At$ is a model of P, written $M \models P$, if M satisfies every rule of P.

A *Horn rule* is the rule with the empty negative part. A Horn program P is a set of Horn rules. Each Horn program P has a least model under inclusion, LM_P, which can be defined using the *one-step provability operator* $T[P]$ as follows. For any set A, let $\mathscr{P}(A)$ denote the set of all subsets of A. The one-step provability operator $T[P] : \mathscr{P}(At) \to \mathscr{P}(At)$ associated with the Horn program P [10] is defined by setting

$$T[P](M) = M \cup \{a : \exists r \in P\ (a = head(r) \wedge M \models body(r))\}$$

for any $M \in \mathscr{P}(At)$. We define $T[P]^n(M)$ by induction by setting $T[P]^0(M) = M$, $T[P]^1(M) = T[P](M)$ and $T[P]^{n+1}(M) = T[P](T[P]^n(M))$. Then the least model LM_P can be computed as $LM_P = \bigcup_{n \geq 0} T[P]^n(\emptyset)$.

If P is a normal logic program and $M \subseteq At$, then the Gelfond-Lifschitz (GL) reduct of P with respect to M [11] is the Horn program P^M which results by eliminating those rules r such that $M \not\models body(r)^-$ and replacing other rules r by $head(r) : -body(r)^+$. We then say that M is a *stable model* for P if M equals the least model of P^M.

An *answer set programming rule* is an expression of the form (2) where $a, b_1, \ldots, b_{k+m}$ are classical literals, i.e., either positive atoms or atoms preceded by the classical negation sign $\neg$. The set of literals of At will be denoted Lit_{At}. *Answer sets* are defined in analogy to stable models, but taking into account that atoms may be preceded by classical negation and that atoms a and classically negated atoms $\neg a$ are mutually exclusive in answer sets.

We will now follow [12] in review of the Splitting Set Theorem and the Splitting Sequence Theorem. A *splitting set* for a program P is any set $U \subseteq At$ such that for every rule $r \in P$ if $head(r) \in U$ then $At(r) \subseteq U$. The set of rules $r \in P$ such that $At(r) \subseteq U$ is called the *bottom* of P relative to the splitting set U and is denoted by $b_U(P)$. The set $P \backslash b_U(P)$ is the *top* of P relative to U.

Consider $X \subseteq At$. For each rule $r \in P$ such that $At(body(r)^+) \cap U \subseteq X$ and $At(body(r)^-) \cap U \cap X = \emptyset$ take the rule r' defined by

$$head(r)\ : -\ body(r) \setminus U$$

The program consisting of all rules r' obtained in this way will be denoted by $\varepsilon_U(P, X)$.

A *solution* to P with respect to U is a pair (X, Y) of sets of literals such that

- X is an answer set for $b_U(P)$
- Y is an answer set for $\varepsilon_U(P \setminus b_U(P), X)$
- $X \cup Y$ is consistent (a set is consistent if for any atom a it does not contain both a and classically negated atom $-a$)

Splitting Set Theorem. *Let U be a splitting set for a program P. A set A of literals is a consistent answer set for P if and only if $A = X \cup Y$ for some solution (X,Y) to P with respect to U.*

We will now review extending the definition of a splitting set to a splitting sequence. A *sequence* is a family whose index set is an initial segment of ordinals, $\{\alpha : \alpha < \mu\}$. The ordinal μ is the *length* of the sequence. A sequence $\langle U_\alpha \rangle_{\alpha<\mu}$ of sets is *monotone* if $U_\alpha \subset U_\beta$ whenever $\alpha < \beta$, and *continuous* if, for each limit ordinal $\alpha < \mu$, $U_\alpha = \bigcup_{\beta<\alpha} U_\beta$.

A *splitting sequence* for a program P is a monotone, continuous sequence $\langle U_\alpha \rangle_{\alpha<\mu}$ of splitting sets for P such that $\bigcup_{\alpha<\mu} U_\alpha = Lit_{At}$. The definition of a solution with respect to a splitting set is extended to splitting sequence as follows. A *solution* to P with respect to $\langle U_\alpha \rangle_{\alpha<\mu}$ is a sequence $\langle X_\alpha \rangle_{\alpha<\mu}$ of sets of literals such that

- X_0 is an answer set for $b_{U_0}(P)$,
- for any α such that $\alpha + 1 < \mu$, $X_{\alpha+1}$ is an answer set for $\varepsilon_{U_\alpha}(b_{U_{\alpha+1}}(P) \setminus b_{U_\alpha}(P), \bigcup_{\beta\leq\alpha} X_\beta)$,
- for any limit ordinal $\alpha < \mu$, $X_\alpha = \emptyset$,
- $\bigcup_{\alpha<\mu} X_\alpha$ is consistent.

Splitting Sequence Theorem. *Let $U \equiv \langle U_\alpha \rangle_{\alpha<\mu}$ be a splitting sequence for a program P. A set A of literals is a consistent answer set for P if and only if $A = \bigcup_{\alpha<\mu} X_\alpha$ for some solution $\langle X_\alpha \rangle_{\alpha<\mu}$ to P with respect to U.*

We will now proceed with the review of Hybrid ASP. A Hybrid ASP program P has an underlying parameter space S. Elements of S are of the form $\mathbf{p} = (t, x_1, \ldots, x_l)$ where t is time and x_i are arbitrary parameter values. We shall let $t(\mathbf{p})$ denote t and $x_i(\mathbf{p})$ denote x_i for $i = 1, \ldots, l$. We refer to the elements of S as *generalized positions*. Let At be a set of atoms of P. Then the universe of P is $At \times S$. Let B be a block. We will define

$$B \times \mathbf{p} \equiv \{(x, \mathbf{p}) : x \in B\}.$$

If $M \subseteq At \times S$, we let $GP(M) = \{\mathbf{p} \in S : (\exists a \in At)((a, \mathbf{p}) \in M)\}$. Given an *initial condition*, defined as a subset $I \subseteq S$ let $GP_I(M) = GP(M) \cup I$. Given $M \subseteq At \times S$ and $\mathbf{p} \in S$, we say that M and initial condition I satisfy a block B of the form (1) at the generalized position $\mathbf{p}$, written $M \models_I (B, \mathbf{p})$, if the following holds:

- if $B^+ \neq \emptyset$ then $B^+ \times \mathbf{p} \subseteq M$ and $B^- \times \mathbf{p} \cap \mathbf{M} = \emptyset$
- if $B^+ = \emptyset$ then $B^- \times \mathbf{p} \cap \mathbf{M} = \emptyset$ and $\mathbf{p} \in GP_I(M)$.

We say that M satisfies a n-tuple of blocks written as B_1; ...; B_n with the initial condition I at the n-tuple of generalized positions $(\mathbf{p}_1, ..., \mathbf{p}_n)$, written $M \models_I (B_1; ...; B_n, (\mathbf{p}_1, ..., \mathbf{p}_n))$, if $M \models_I (B_i, \mathbf{p}_i)$ for $i = 1, ..., n$.

There are two types of rules in Hybrid ASP. **Advancing rules** are of the form

$$r \equiv a : -B_1; B_2; \ldots; B_n : A, O \tag{3}$$

where A is a function returning a set of generalized positions, $body\,(r) \equiv B_1$, ..., B_n are blocks, $head\,(r) \equiv a$ is a literal, and O is a subset of S^n such that if $(\mathbf{p}_1, \dots, \mathbf{p}_n) \in O$, then $t(\mathbf{p}_1) < \cdots < t(\mathbf{p}_n)$ and $A\,(\mathbf{p}_1, \dots, \mathbf{p}_n)$ (A applied to $\mathbf{p}_1, \dots, \mathbf{p}_n$) is a subset of S such that for all $\mathbf{q} \in A\,(\mathbf{p}_1, \dots, \mathbf{p}_n)$, $t(\mathbf{q}) > t(\mathbf{p}_n)$. Here and in the next rule, we allow blocks to be empty for any i. O is called the *constraint set* of the rule r and will be denoted by $CS(r)$. A is called the *advancing algorithm* of the rule r and is denoted by $\mathrm{Adv}(r)$. The arity of rule r, $N\,(r)$, is equal to n.

The idea is that if $(\mathbf{p}_1, \dots, \mathbf{p}_n) \in O$ and for each i, B_i is satisfied at the generalized position $\mathbf{p}_i$, then the function A can be applied to $(\mathbf{p}_1, \dots, \mathbf{p}_n)$ to produce a set of generalized positions O' such that if $\mathbf{q} \in O'$, then $t(\mathbf{q}) > t(\mathbf{p}_n)$ and $(a, \mathbf{q})$ holds. Thus advancing rules are like input-output devices in that the function A allows the user to derive possible successor generalized positions as well as certain atoms a which are to hold at such positions. The advancing algorithm A can access outside sources quite arbitrarily in that it may involve functions for solving differential or integral equations, solving a set of linear equations or linear programming equations, solving an optimization problem, etc. (as for example in [5]).

Stationary rules are of the form

$$r \equiv a : -B_1; B_2; \dots; B_n : H, O \tag{4}$$

where $body\,(r) \equiv B_1, ..., B_n$ are blocks, $head\,(r) \equiv a$ is a literal, H is called a *boolean algorithm* of the rule r and will be denoted by $Bool\,(r)$, and $O \subseteq S^k$ is the constraint set of the rule r denoted $CS(r)$. A boolean algorithm is a function returning either true or false. We will sometimes treat a boolean algorithm of the rule as a set. For instance $H \cap O$ will indicate all the n-tuples of generalized positions $(\mathbf{p}_1, \dots, \mathbf{p}_n)$ such that $H\,(\mathbf{p}_1, \dots, \mathbf{p}_n)$ is true and $(\mathbf{p}_1, \dots, \mathbf{p}_n) \in O$. The arity of rule r, $N\,(r)$, is equal to n.

Stationary rules are much like normal logic programming rules in that they allow us to derive new atoms at a given generalized position $\mathbf{p}_n$. The idea is that if $(\mathbf{p}_1, \dots, \mathbf{p}_n) \in O \cap H$ and for each i, B_i is satisfied at the generalized position $\mathbf{p}_i$, then $(a, \mathbf{p}_n)$ holds. The difference is that a derivation with our stationary rules can depend on what happens in the multiple past time points and the boolean algorithm H can be any sort of a function which returns either true or false.

For an advancing rule or a stationary rule r as above we define the *positive part of the body* of r, denoted $body\,(r)^+ \equiv B_1^+; ...; B_n^+$ and we define the *negative part of the body* of r, denoted $body\,(r)^- \equiv B_1^-; ...; B_n^-$. For the rest of the paper, we denote by n the arity of a hybrid ASP rule when the rule is clear from the context.

A Hybrid ASP program P is a collection of Hybrid ASP advancing and stationary rules. To define the notion of a stable model of P, we first must define the notion of a Hybrid ASP Horn program and the one-step provability operator for Hybrid ASP Horn programs.

A *Hybrid ASP Horn program* is a Hybrid ASP program which does not contain any negated atoms. Let P be a Horn Hybrid ASP program and $I \subseteq S$ be an initial condition. Then the one-step provability operator $T\,[P,I]$ is defined so that given $M \subseteq At \times S$, $T\,[P,I]\,(M)$ consists of M together with the set of all $(a,J) \in At \times S$ such that

1. there exists a stationary rule r and $(\mathbf{p}_1, \dots, \mathbf{p}_n) \in CS\,(r) \cap Bool\,(r) \cap (GP_I(M))^n$ such that $(head\,(r)\,, J) = (a, \mathbf{p}_n)$ and $M \models (body\,(r)\,,\ (\mathbf{p}_1, ..., \mathbf{p}_n))$ or
2. there exists an advancing rule r and $(\mathbf{p}_1, \dots, \mathbf{p}_n) \in CS\,(r) \cap (GP_I(M))^n$ such that $J \in \mathrm{Adv}\,(r)\,(\mathbf{p}_1, \dots, \mathbf{p}_n)$ and $M \models (body\,(r)\,,\ (\mathbf{p}_1, ..., \mathbf{p}_n))$ and $a = head\,(r)$.

The stable model semantics for Hybrid ASP programs is defined as follows. Let $M \subseteq At \times S$ and I be an initial condition in S. An Hybrid ASP rule $r \equiv a : -B_1; \dots, B_n : A, O$ is *inapplicable* for (M,I) if for all

$(\mathbf{p}_1,\ldots,\mathbf{p}_n) \in O \cap (GP_I(M))^n$, either (i) there is an i such that $M \not\models (B_i^-, \mathbf{p}_i)$, (ii) $A(\mathbf{p}_1,\ldots,\mathbf{p}_n) \cap GP_I(M) = \emptyset$ if A is an advancing algorithm, or (iii) $A(\mathbf{p}_1,\ldots,\mathbf{p}_n) = 0$ if A is a boolean algorithm.

If r is not inapplicable for (M,I) then we define the GL reduct of r over M and I, denoted by $r^{M,I}$ as follows:

1. If r is an advancing rule $r \equiv a : -B_1;...;B_n : A,O$ then $r^{M,I} \equiv B_1^+;\ldots,B_n^+ : A^{M,I}, O^{M,I}$ where $O^{M,I}$ is equal to the set of $(\mathbf{p}_1,\ldots,\mathbf{p}_n)$ in $O \cap (GP_I(M))^n$ such that $M \models_I (body(r)^-, (\mathbf{p}_1,\ldots,\mathbf{p}_n))$ and $A(\mathbf{p}_1,\ldots,\mathbf{p}_n) \cap GP_I(M) \neq \emptyset$, and $A^{M,I}(\mathbf{p}_1,\ldots,\mathbf{p}_n) \equiv A(\mathbf{p}_1,\ldots,\mathbf{p}_n) \cap GP_I(M)$.
2. If r is a stationary rule $r \equiv a : -B_1;...;B_n : A,O$ then $r^{M,I} \equiv a : -B_1^+;\ldots,B_n^+ : H|_{O^{M,I}}, O^{M,I}$ where $O^{M,I}$ is equal to the set of all $(\mathbf{p}_1,\ldots,\mathbf{p}_n)$ in $O \cap (GP_I(M))^n$ such that $M \models_I (body(r)^-, (\mathbf{p}_1,\ldots,\mathbf{p}_n))$ and $H(\mathbf{p}_1,\ldots,\mathbf{p}_n)$ is true.

One note to make about the definition above is that GL reduct cannot derive generalized positions that are not in $GP_I(M)$. This is because the range of $A^{M,I}$ in the definition is restricted to $GP_I(M)$.

We form a GL reduct of P over M and I, $P^{M,I}$ as follows.

1. Eliminate all rules which are inapplicable for (M,I).
2. If a rule $r \in P$ is not eliminated in step 1, then replace it by the rule $r^{M,I}$.

We then say that M is a *stable model of P with initial condition I* if $\bigcup_{k=0}^{\infty} T\left[P^{M,I}, I\right]^k(\emptyset) = M$.

Answer sets are defined in analogy to stable models, but taking into account that atoms may be preceded by classical negation and that $(a,\mathbf{p})$ and $(-a,\mathbf{p})$ are mutually exclusive in answer sets.

2 The Splitting Set Theorem for Hybrid ASP

We will now introduce additional notation that will be used throughout the rest of the paper.

Without loss of generality assume that all advancing rules are of the form

$$a : -B_1;\ ...;\ B_n : O,A$$

and all of stationary rules are of the form

$$a : -B_1;\ ...;\ B_n : O,H$$

where a is a literal, B_1, ..., B_n are blocks, O is a constraint set, A is an advancing algorithm, and H is a boolean algorithm.

Let M be a set of literals and generalized position pairs, and let $\mathbf{p}$ be a generalized position. Define

$$M|_{\mathbf{p}} \equiv \{(a,\mathbf{q}) \in M \ :\ \mathbf{q} = \mathbf{p}\}$$

$$At(M) \equiv \{a \ :\ (a,\mathbf{p}) \in M\}$$

Let $U \subseteq Lit_{At} \times S$. We say that U is a *splitting set of P with initial condition* (w.i.c.) J if for all $r \in P$

1. if r is advancing and $(\mathbf{p}_1,\ ...,\ \mathbf{p}_n) \in CS(r)$ and $\mathbf{p} \in Adv(r)(\mathbf{p}_1,\ ...,\ \mathbf{p}_n)$ and $(a,\mathbf{p}) \in U$ then both for $i = 1,...,n$, $B_i \times \mathbf{p}_i \subseteq U$ and $\{\mathbf{p}_1,\ ...,\ \mathbf{p}_n\} \subseteq GP_J(U)$.

2. if r is stationary and $(\mathbf{p}_1, \ldots, \mathbf{p}_n) \in CS(r)$ and $(a, \mathbf{p}_n) \in U$ then both for $i = 1, \ldots, n$, $B_i \times \mathbf{p}_i \subseteq U$ and $\{\mathbf{p}_1, \ldots, \mathbf{p}_n\} \subseteq GP_J(U)$.

As in the case of the original splitting set theorem [12] the splitting set U acts to split Hybrid ASP program P into the part that can derive U or one of its subsets, and the remaining part of P, which can derive $At \times S \backslash U$ or one of its subsets. The difference, however, is that for a given rule the conclusion of the rule may be in U for some n-tuples of generalized positions $(\mathbf{p}_1, \ldots, \mathbf{p}_n)$ and not for others. So, the splitting set splits not only the program, but the rules themselves. This will be elaborated below.

As in the case of the original splitting set theorem we identify by $b_U(P)$ a set of new rules that capture the rules and generalized positions that may contribute to generating U.

Define $Rules_b(U,P)$ as

$$\{\ r \in P\ :\ \text{if } r \text{ is advancing and there exists } (\mathbf{p}_1, \ldots, \mathbf{p}_n) \in CS(r)$$

$$\text{and } \mathbf{p} \in Adv(r)(\mathbf{p}_1, \ldots, \mathbf{p}_n) \text{ such that } (a, \mathbf{p}) \in U$$

$$\text{if } r \text{ is stationary and there exists } (\mathbf{p}_1, \ldots, \mathbf{p}_n) \in CS(r) \cap Bool(r)$$

$$\text{such that } (a, \mathbf{p}_n) \in U\ \}$$

In other words, $Rules_b(U,P)$ is the set of all rules of P that could contribute to U for some tuple of generalized positions.

For an advancing rule r let

$$CS_b(U,r) \equiv \{\ (\mathbf{p}_1, \ldots, \mathbf{p}_n) \in CS(r)\ :$$

$$\text{there exists } p \in Adv(r)(\mathbf{p}_1, \ldots, \mathbf{p}_n) \text{ such that } (a, \mathbf{p}) \in U\ \}$$

For a stationary rule r let

$$CS_b(U,r) \equiv \{\ (\mathbf{p}_1, \ldots, \mathbf{p}_n) \in CS(r) \cap Bool(r)\ :\ (a, \mathbf{p}_n) \in U\ \}$$

That is $CS_b(U,r)$ are all the generalized position tuples for which r could contribute to U.

For an advancing rule $r \in Rules_b(U,P)$ define $Adv_b(U,r)$ by

$$Adv_b(U,r)(\mathbf{p}_1, \ldots, \mathbf{p}_n) \equiv \{\ \mathbf{p}\ :\ \mathbf{p} \in Adv(r)(\mathbf{p}_1, \ldots, \mathbf{p}_n)$$

$$\text{such that } (a, \mathbf{p}) \in U \text{ if } (\mathbf{p}_1, \ldots, \mathbf{p}_n) \in CS_b(U,r)\ \}$$

$Adv_b(U,r)$ is an advancing algorithm that for any tuple of generalized positions will only generate those $\mathbf{p}$ that contribute to U.

For an advancing rule r let

$$b_U(r) \equiv head(r) :- body(r) : CS_b(U,r),\ Adv_b(U,r)$$

For a stationary rule r let

$$b_U(r) \equiv head(r) :- body(r) : CS_b(U,r),\ Bool(r)$$

Define the *bottom of* P with respect to U, $b_U(P)$ as

$$b_U(P) \equiv \{\ b_U(r)\ :\ r \in Rules_b(U,P)\ \}$$

The idea is that just like in [12], $b_U(P)$ forms only those rules that could contribute to U, and so X will be an answer set of $b_U(P)$ w.i.c. J iff $M \cap U = X$ for some answer set M of P w.i.c. J.

We will now proceed to define $\varepsilon_U(P,X)$ with the understanding that the same rule may contribute to U for some generalized position tuples and contribute to $Lit_{At} \times S \setminus U$ for others.

First, we need to identify remainder $\mathrm{Rem}(U,P)$ of $Rules_b(U,P)$ not captured by $b_U(P)$. That is we need to identify the parts contributing to the complement of U of those rules that have other parts contributing to U. This is due to an important difference between Hybrid ASP and ASP. In ASP a rule contributes a single conclusion. Thus if ASP rule contributes to the splitting set then it must be in the bottom of the program. In Hybrid ASP, however, a rule acts more like a collection of rules contributing different conclusions for different generalized position tuples. Consequently, the parts of the rules that contribute to the complement of the splitting set need to be separated from those that contribute to the splitting set itself. We will now proceed with the definition.

For an advancing rule r define

$$CS_{\mathrm{Rem}}(U,r) \equiv \{\ (\mathbf{p}_1,\ \ldots,\ \mathbf{p}_n) \in CS(r) :$$

$$\text{there exists } \mathbf{p} \in Adv(r)(\mathbf{p}_1,\ \ldots,\ \mathbf{p}_n)\ \ (a,\mathbf{p}) \notin U\ \}$$

For a stationary r define

$$CS_{\mathrm{Rem}}(U,r) \equiv \{\ (\mathbf{p}_1,\ \ldots,\ \mathbf{p}_n) \in CS(r) \cap Bool(r) :\ (a,\mathbf{p}_n) \notin U\ \}$$

That is, $CS_{\mathrm{Rem}}(U,r)$ contains those generalized position tuples such that for them the rule r contributes to the complement of U.

For an advancing rule $r \in Rules_b(U,P)$ and $(\mathbf{p}_1,\ \ldots,\ \mathbf{p}_n)$ define

$$Adv_{\mathrm{Rem}}(U,r)(\mathbf{p}_1,\ \ldots,\ \mathbf{p}_n) \equiv \begin{cases} \{\mathbf{p}\ :\ \mathbf{p} \in Adv(r)(\mathbf{p}_1,\ \ldots,\ \mathbf{p}_n) \text{ s.t. } (a,\mathbf{p}) \notin U\ \} \\ \quad \text{if } (\mathbf{p}_1,\ \ldots,\ \mathbf{p}_n) \in CS_{\mathrm{Rem}}(U,r) \\ \emptyset \text{ if } (\mathbf{p}_1,\ \ldots,\ \mathbf{p}_n) \notin CS_{\mathrm{Rem}}(U,r) \end{cases}$$

That is $Adv_{\mathrm{Rem}}(U,r)$ is a restriction of $Adv(r)$ to those generalized positions such that for them r contributes to the complement of U.

When $CS_{\mathrm{Rem}}(U,r) \neq \emptyset$ define

$$\mathrm{Rem}(U,r) \equiv \begin{cases} head(r) :- body(r) : CS_{\mathrm{Rem}}(U,r),\ Adv_{\mathrm{Rem}}(U,r) \text{ if } r \text{ is advancing} \\ \quad head(r) :- body(r) : CS_{\mathrm{Rem}}(U,r),\ Bool(r) \text{ if } r \text{ is stationary} \end{cases}$$

In other words, $\mathrm{Rem}(U,r)$ is the part of r that contributes to the complement of U.

Define

$$\mathrm{Rem}(U,P) \equiv \{\ \mathrm{Rem}(U,r) :\ r \in Rules_b(U,P) \text{ and } CS_{\mathrm{Rem}}(U,r) \neq \emptyset\ \}$$

That is $\mathrm{Rem}(U,P)$ contain those parts of the rules in $Rules_b(U,P)$ that contribute to the complement of U.

Let $X \subseteq U$. For a rule r define

$$CS_{\varepsilon}(U,r,X) \equiv \{\ (\mathbf{p}_1,\ \ldots,\ \mathbf{p}_n) \in CS(r) :$$

$$\text{for } i = 1,\ ...,\ n\ \{B_i^+ \times \mathbf{p}_i\} \cap U \subseteq X \text{ and } \{B_i^- \times \mathbf{p}_i\} \cap X = \emptyset\ \}$$

That is $CS_\varepsilon(U,r,X)$ is the set of those generalized position tuples such that for them the "projection" of $body(r)$ onto U is satisfied by X.

Finally

$$\varepsilon_U(P,X) \equiv \{$$

$$r' \equiv a :- B_1 \backslash At(U|_{\mathbf{p}_1});\ ...;\ B_n \backslash At(U|_{\mathbf{p}_n}) :\ \{(\mathbf{p}_1,\ ...,\ \mathbf{p}_n)\},\ Q\ |$$

$$r \equiv a :- B_1;\ ...;\ B_n : O, Q\ \in\ \{\ r \in P :\ CS(\varepsilon_U, r, X) \neq \emptyset\ \} \text{ and}$$

$$(\mathbf{p}_1,\ ...,\ \mathbf{p}_n) \in CS_\varepsilon(U,r,X)\ \}$$

In other words, for every rule $r \in P$ such that $CS(\varepsilon_U,r,X) \neq \emptyset$ and for every $(\mathbf{p}_1,\ ...,\ \mathbf{p}_n) \in CS_\varepsilon(U,r,X)$ where the "projection" of $body(r)$ onto U is satisfied by X at $(\mathbf{p}_1,\ ...,\ \mathbf{p}_n)$, we add to $\varepsilon_U(P,X)$ a rule r', which is a part of rule r that will be active only for that $(\mathbf{p}_1,\ ...,\ \mathbf{p}_n)$ with the "projection" part removed.

Theorem 1. *(The Splitting Set Theorem for Hybrid ASP). Let P be a Hybrid ASP program over $Lit_{At} \times S$. Let $U \subseteq Lit_{At} \times S$ be a splitting set of P w.i.c. $J \subseteq S$. A set M is a answer set of P w.i.c. J iff $X \equiv M \cap U$ is a answer set of $b_U(P)$ w.i.c. J and $M \backslash U$ is a answer set of $\varepsilon_U(P \backslash Rules_b(U,P) \cup$* $\mathrm{Rem}(U,P)$*, $X)$ w.i.c. $GP_J(X)$.*

Sketch of a proof. We first prove that if M is an answer set of P w.i.c. J then $X \equiv M \cap U$ is an answer set of $b_U(P)$ w.i.c. J. That is, we want to show that $X = \bigcup_{k=0}^{\infty} T\left[b_U(P)^{X,J}, J\right]^k(\emptyset)$. In $\supseteq$ direction we show by induction on k in one-step provability operator $T\left[b_U(P)^{X,J}, J\right]^k$ that if a rule $b_U(r)^{X,J}$ in $b_U(P)^{X,J}$ derives $(a,\mathbf{p})$ in $T\left[b_U(P)^{X,J}, J\right]^{k+1}(\emptyset)$, then the rule $r^{M,J}$ must derive $(a,\mathbf{p})$ in $T\left[P^{M,J}, J\right]^{m+1}(\emptyset)$ for some m. In $\subseteq$ direction we show by induction on k in $T\left[P^{M,J}, J\right]^k(\emptyset)$ that if $r^{M,J}$ derives $(a,\mathbf{p})$ in $T\left[P^{M,J}, J\right]^{k+1}(\emptyset)$ where $(a,\mathbf{p}) \in U$, then $b_U(r)^{X,J}$ derives $(a,\mathbf{p})$ in $T\left[b_U(P)^{X,J}, J\right]^{m+1}(\emptyset)$ for some m.

We then proceed to prove that if M is an answer set of P w.i.c. J, and $Y \equiv M \backslash U$ then Y is an answer set of $Q \equiv \varepsilon_U(P \backslash Rules_b(U,P) \cup \mathrm{Rem}(U,P), X)$ *w.i.c.* $L \equiv GP_J(X)$. That is, we want to show that $Y = \bigcup_{k=0}^{\infty} T\left[Q^{Y,L}, L\right]^k(\emptyset)$. In $\supseteq$ direction we prove by induction that if $r^{Y,L}$ derives $(a,\mathbf{p})$ in $T\left[Q^{Y,L}, L\right]^{k+1}(\emptyset)$ then there is a corresponding rule $q^{M,J}$ in $P^{M,J}$ that derives $(a,\mathbf{p})$ in $T\left[P^{M,J}, J\right]^{m+1}(\emptyset)$ for some m. In $\subseteq$ direction we prove by induction on k in $T\left[P^{M,J}, J\right]^k(\emptyset)$ that if $q^{M,J}$ derives $(a,\mathbf{p})$ in $T\left[P^{M,J}, J\right]^{k+1}(\emptyset)$ where $(a,\mathbf{p}) \in M \backslash U$ then there is a corresponding r in $Q^{Y,L}$ that derives $(a,\mathbf{p})$ in $T\left[Q^{Y,L}, L\right]^{m+1}(\emptyset)$ for some m.

To finish the proof we need to show that if $X \subseteq U$ is an answer set of $b_U(P)$ w.i.c. J and $Y \subseteq U^C$ is an answer set of Q w.i.c. L then $M \equiv X \cup Y$ is an answer set of P w.i.c. J. That is we want to show that $M = \bigcup_{k=0}^{\infty} T\left[P^{M,J}, J\right]^k(\emptyset)$. We do so by induction in both directions in a manner similar to the previous part of the proof. □

Similar to the Splitting Sequence Theorem of [12] we also prove the Splitting Sequence Theorem for Hybrid ASP.

Theorem 2. *(The Splitting Sequence Theorem for Hybrid ASP). Let $\langle U_\alpha \rangle_{\alpha<\mu}$ be a monotone continuous sequence of splitting sets for a Hybrid ASP program P over $At \times S$ w.i.c. $J \subseteq S$, and $\bigcup_{\alpha<\mu} U_\alpha = Lit_{At} \times S$. M is an answer set of P w.i.c. J iff $M = \bigcup_{\alpha<\mu} X_\alpha$ for a sequence $\langle X_\alpha \rangle_{\alpha<\mu}$ s.t.*

- *X_0 is an answer set of $b_{U_0}(P)$ w.i.c. J*
- *for any α such that $\alpha+1<\mu$ $X_{\alpha+1}$ is an answer set for $\varepsilon_{U_\alpha}(b_{U_{\alpha+1}}(P) \setminus Rules_b(U_\alpha, b_{U_{\alpha+1}}(P)) \cup \mathrm{Rem}(U_\alpha, b_{U_{\alpha+1}}(P)), \bigcup_{\beta\leq\alpha} X_\beta)$ w.i.c. $L_\alpha \equiv GP_J(\bigcup_{\beta\leq\alpha} X_\beta)$ and $X_{\alpha+1} = M \cap (U_{\alpha+1} \setminus U_\alpha)$ and $\bigcup_{\beta\leq\alpha} X_\beta$ is an answer set of $b_{U_\alpha}(P)$ w.i.c. J.*

The proof proceeds by the induction on α and is a direct application of The Splitting Set Theorem for Hybrid ASP.

In the Splitting Sequence Theorem for Hybrid ASP, $b_{U_{\alpha+1}}(P)$ is a program that derives $\bigcup_{\beta\leq\alpha+1} X_\beta$ as its answer set w.i.c. J. Now, $\bigcup_{\beta\leq\alpha+1} X_\beta \subseteq \bigcup_{\beta\leq\alpha+1} U_\beta$. So, to derive $X_{\alpha+1}$ (i.e. the subset of $\bigcup_{\beta\leq\alpha+1} X_\beta$ that is in $U_{\alpha+1} \setminus U_\alpha$) we need to remove from $b_{U_{\alpha+1}}(P)$ the rules that derive $\bigcup_{\beta\leq\alpha} X_\beta$. That is accomplished by subtracting from $b_{U_{\alpha+1}}(P)$ the rules $Rules_b(U_\alpha, b_{U_{\alpha+1}}(P))$. Nevertheless, this subtracts too much as some of the rules in $Rules_b(U_\alpha, b_{U_{\alpha+1}}(P))$ contribute to $X_{\alpha+1}$ for some generalized position tuples. The parts of those rules that contribute to $X_{\alpha+1}$ are $\mathrm{Rem}(U_\alpha, b_{U_{\alpha+1}}(P))$, which we then add back. Applying ε_{U_α} operator to the resulting program (i.e. $b_{U_{\alpha+1}}(P) \setminus Rules_b(U_\alpha, b_{U_{\alpha+1}}(P)) \cup \mathrm{Rem}(U_\alpha, b_{U_{\alpha+1}}(P))$) then removes the "useless" part of the rules with respect to $\bigcup_{\beta\leq\alpha} X_\beta$.

3 An Application: Computing Answer Sets of Hybrid ASP Programs

One of the applications of the Splitting Sequence Theorem for Hybrid ASP is proving the correctness of a certain algorithm for computing answer sets of certain types of Hybrid ASP programs. We will consider only the programs where the set of generalized positions S is such that if $\mathbf{p} \in S$ then $t(\mathbf{p}) = k \cdot \Delta t$ where $k \in \mathbb{N}$, and for any advancing rule r of any arity n, for any $(\mathbf{p}_1, \ldots, \mathbf{p}_n) \in S^n$ we have that for all $\mathbf{q} \in Adv(r)(\mathbf{p}_1, \ldots, \mathbf{p}_n)$, $t(\mathbf{q}) = t(\mathbf{p}_n) + \Delta t$. That is, these are the programs with generalized positions with discrete times of the form $k\Delta t$, and whenever an advancing algorithm produces a new generalized position, that generalized position has time larger by Δt than the largest time in the input arguments. All applications of Hybrid ASP known to the author are restricted to such programs. This is the case for using Hybrid ASP to diagnose failure of data processing pipelines, as described in [2] and [8]. It is the case for the Hybrid ASP programs that are the result of translation from action languages Hybrid AL [7] and Hybrid ALE [2]. It is also the case for using Hybrid ASP to compute optimal finite horizon policies in dynamic domains [5].

The algorithm.

We will first describe the algorithm informally. We will use some of the new notation which will be defined further below. The algorithm is based on the observation that in Hybrid ASP the facts in the "future" cannot affect the facts in the "past". That is for any two generalized position $\mathbf{p}$ and $\mathbf{q}$, if $t(\mathbf{p}) < t(\mathbf{q})$ then the state at $\mathbf{q}$ cannot be used to derive the state at $\mathbf{p}$ (but the state at $\mathbf{p}$ can be used to derive the state at $\mathbf{q}$). Consequently, it should be possible to first derive the states at some minimal time $t_{\min}$, then derive the states at the time $t_{\min} + \Delta t$, then derive the states at time $t_{\min} + 2\Delta t$ and so on.

Without the loss of generality, we will assume that for any initial condition $J \subseteq S$, there exists $\mathbf{p} \in J$ such that $t(\mathbf{p}) = 0$. Let P be a Hybrid ASP program over $Lit_{At} \times S$. Let $J \subseteq S$ be an initial condition. The algorithm will be defined inductively. Suppose the set N of all the (literal, generalized position) pairs for the generalized positions with time up to $k \cdot \Delta t$ is derived by the algorithm for some k. The algorithm will first identify all the advancing rules $Rules_{Adv}(P,N,k\Delta t)$ that could derive generalized positions with time $(k+1)\cdot\Delta t$. These are the advancing rules r such that N satisfies their body for some $(\mathbf{p}_1, ..., \mathbf{p}_n) \in CS(r)$, where $n = arity(r)$ and the time of $\mathbf{p}_n$ is $k \cdot \Delta t$. The set of the "next" generalized positions (i.e. the set of generalized positions with time $(k+1)\cdot\Delta t$) is derived by choosing a subset of the set of all the generalized positions derived by these rules. To formally define such a choice of a subset we introduce a concept of an *advancing selector* F, which is a function s.t. for $M \subseteq Lit_{At} \times S$ and $Z \subseteq S$, $F(M,Z)$ is a subset of Z. We will denote the set of "next" generalized positions derived in this manner by $NextGP(P,F,N,k\Delta t)$.

Now, for every "next" generalized position $\mathbf{q}$ in $NextGP(P,F,N,k\Delta t)$ derived by an advancing rule $r \in Rules_{Adv}(P,N,k\Delta t)$, it must be that $(head(r), \mathbf{q})$ is derived. So, for every $\mathbf{q}$ there is a set of literals that will be derived at $\mathbf{q}$ by the advancing rules in $Rules_{Adv}(P,N,k\Delta t)$. This set of literals will be denoted by $Head_{Adv}(P,N,\mathbf{q})$.

Next we turn our attention to the role of the stationary rules in deriving hybrid state at a "next" generalized position $\mathbf{q}$. There is a set of stationary rules that can contribute to the hybrid state at $\mathbf{q}$. If such a stationary rule r has n blocks, then the first $n-1$ blocks are satisfied by N (at some generalized positions $\mathbf{p}_1$, ..., $\mathbf{p}_{n-1}$) and $(\mathbf{p}_1, ..., \mathbf{p}_{n-1}, \mathbf{q})$ are in $CS(r) \cap Bool(r)$. Thus, only the last block, which we will denote by B_n needs to be evaluated at $\mathbf{q}$. Thus, the relevant part of such a stationary rule r is a regular ASP rule of the form $head(r) :- B_n$. All such regular ASP rules applicable at $\mathbf{q}$ will be denoted by $\mathrm{Red}_{App}(P, N, \mathbf{q})$. A state at $\mathbf{q}$ is then an answer set of a regular ASP program $\mathrm{Red}_{App}(P, N, \mathbf{q}) \cup \{[h : -] : h \in Head_{Adv}(P,N,\mathbf{q})\}$. To formally define such a choice we will use a concept of a *stationary selector* D, which we will define further below.

We will now define the algorithm formally.

For a set $N \subseteq Lit_{At} \times S$ and generalized positions $\mathbf{p}$ and $\mathbf{q}$, let

$$Rules_{Adv}(P,N,k\Delta t) \equiv \{r \in P : r \text{ is an advancing rule and there is}$$

$$(\mathbf{p}_1, ..., \mathbf{p}_n) \in GP_J(N)^n \cap CS(r) \text{ with } t(\mathbf{p}_n) = k \cdot \Delta t \text{ and } N \models_J (body(r), (\mathbf{p}_1, ..., \mathbf{p}_n)\}$$

Let $\mathbf{p}_1$, ..., $\mathbf{p}_n \in GP_J(N)$. We define the set of advancing rules active at $\mathbf{p}_1$, ..., $\mathbf{p}_n$ relative to N as

$$Rules_{Adv}(P, N, (\mathbf{p}_1, ..., \mathbf{p}_n)) \equiv \{r \in Rules_{Adv}(P, N, t(\mathbf{p}_n)) : (\mathbf{p}_1, ..., \mathbf{p}_n) \in CS(r)\}.$$

That is, $Rules_{Adv}(P, N, (\mathbf{p}_1, ..., \mathbf{p}_n))$ is the set of the advancing rules whose body is satisfied by N at $(\mathbf{p}_1, ..., \mathbf{p}_n)$ and $(\mathbf{p}_1, ..., \mathbf{p}_n) \in CS(r)$.

We define the set of "next" generalized positions at $\mathbf{p}_1$, ..., $\mathbf{p}_n$ relative to N as

$$NextGP(P,N, (\mathbf{p}_1, ..., \mathbf{p}_n)) \equiv \bigcup_{r \in Rules_{Adv}(P, N, (\mathbf{p}_1, ..., \mathbf{p}_n))} Adv(r)(\mathbf{p}_1, ..., \mathbf{p}_n).$$

That is $NextGP(P,N, (\mathbf{p}_1, ..., \mathbf{p}_n))$ is the set of "next" generalized positions generated by any advancing rule active at $\mathbf{p}_1$, ..., $\mathbf{p}_n$ relative to N.

For a time $k \cdot \Delta t$, we define the set of all the "next" generalized positions relative to N, $k \cdot \Delta t$ and an advancing selector F as

$$NextGP(P,F,N,k\Delta t) \equiv F(N, \bigcup_{\substack{n \geq 1 \\ \mathbf{p}_1,\ \ldots,\ \mathbf{p}_n \in GP_J(N) \\ t(\mathbf{p}_n) = k\Delta t}} NextGP(P,N,\ (\mathbf{p}_1,\ \ldots,\ \mathbf{p}_n))\ ).$$

The set of all heads at $\mathbf{q} \in NextGP\,(P,F,N,k\Delta t)$ relative to N is then

$$Head_{Adv}(P,\ N,\ \mathbf{q}) \equiv \{\ head\,(r)\ \text{ : there exists } \mathbf{p}_1,\ \ldots,\ \mathbf{p}_n \in GP_J\,(N) \text{ and}$$

$$r \in Rules_{Adv}(P,\ N,\ (\mathbf{p}_1,\ \ldots,\ \mathbf{p}_n))$$

$$\text{such that } \mathbf{q} \in Adv\,(r)\,(\mathbf{p}_1,\ \ldots,\ \mathbf{p}_n)\}.$$

Let $\mathbf{p}_1,\ \ldots,\ \mathbf{p}_n \in GP_J\,(N)$. We define the set of stationary rules active at $\mathbf{p}_1,\ \ldots,\ \mathbf{p}_n$ relative to N as

$$Rules_{Stat}(P,\ N,\ (\mathbf{p}_1,\ \ldots,\ \mathbf{p}_n)) \equiv \{r \in P\ : r \text{ is stationary and}$$

$$(\mathbf{p}_1,\ \ldots,\ \mathbf{p}_n) \in CS\,(r) \cap Bool\,(r) \text{ and for } i = 1,\ \ldots,\ n-1\ N \models_J (B_i,\ \mathbf{p}_i)\ \}.$$

That is $Rules_{Stat}(P,\ N,\ (\mathbf{p}_1,\ \ldots,\ \mathbf{p}_n))$ is the set of stationary rules with $n-1$ blocks satisfied by N at $\mathbf{p}_1,\ \ldots,\ \mathbf{p}_{n-1}$ respectively, and $(\mathbf{p}_1,\ \ldots,\ \mathbf{p}_n) \in CS\,(r) \cap Bool\,(r)$.

We define a *stationary selector* D to be a function such that for $M \subseteq At \times S$ for $\mathbf{z} \in S$ for an ASP program U, $D(M,\ \mathbf{z},\ U)$ is an answer set of U. That is, a stationary selector chooses one of answer sets of a regular ASP programs U.

For a stationary rule r of the form $a : -B_1;\ \ldots;\ B_n : O,H$, we define an *applicable reduct* of r

$$\mathrm{Red}_{App}\,(r) \equiv \{a : -\ B_n\}.$$

For $\mathbf{z} \in NewGP(P,F,N,k\Delta t)$ we define the active reduct of P at $\mathbf{z}$ relative to N as

$$\mathrm{Red}_{App}(P,\ N,\ \mathbf{z}) \equiv \{\mathrm{Red}_{App}\,(r) :\ \text{there exists } n \geq 1 \text{ and } (\mathbf{p}_1,\ \ldots,\ \mathbf{p}_{n-1}) \in GP_J\,(N)^{n-1}$$

$$\text{such that } r \in Rules_{Stat}\,(P,\ N,\ (\mathbf{p}_1,\ \ldots,\ \mathbf{p}_{n-1},\ \mathbf{z})\ \}$$

Finally, for $N \subseteq At \times S$ and $i \in \mathbb{N}$ let $N\,[i] \equiv \{(a,\ \mathbf{p}) \in N :\ t\,(\mathbf{p}) = i \cdot \Delta t\}$. Similarly for $Z \subseteq S$, $Z[i] \equiv \{\mathbf{p} \in Z\ :\ t\,(\mathbf{p}) = i \cdot \Delta t\}$.

We are now ready to formally specify our algorithm. We define a sequence of sets $\langle Y_i \rangle_{i \geq 0}$, $Y_i \subseteq (At \times S)[i]$ as follows:

$$Y_0 \equiv \bigcup_{\mathbf{z} \in J[0]} D(\emptyset,\ \mathbf{z},\ \mathrm{Red}_{App}(P,\ \emptyset,\ \mathbf{z})) \times \mathbf{z}$$

That is, the state at any generalized position $\mathbf{z} \in J$ with time equal to 0 is determined by taking all the stationary rules r with one block (i.e. rules of the form $a : -B : O,H$) such that $\mathbf{z} \in O \cap H$, composing a regular ASP program from the reducts of the form $a : -B$ derived from those rules, and then finding an answer set of that program.

Now, suppose Y_i are defined for $0 \le i \le k$ and $Y_k \ne \emptyset$. Let

$$Z_{k+1} \equiv NextGP(P,F, \bigcup_{i=0}^{k} Y_i,\ k\Delta t).$$

That is Z_{k+1} is the set of generalized positions with time $(k+1)\Delta t$ derived by the advancing rules $Rules_{Adv}\left(P, \bigcup_{i=0}^{k} Y_i,\ k\Delta t\right)$.

Let

$$Y_{k+1} \equiv \bigcup_{\mathbf{z}\in Z_{k+1}} D(\bigcup_{i=0}^{k} Y_i,\ \mathbf{z},\ \mathrm{Red}_{App}(P, \bigcup_{i=0}^{k} Y_i,\ \mathbf{z}) \cup$$

$$\{[a : -]\ :\ a \in Head_{Adv}(P, \bigcup_{i=0}^{k} Y_i,\ \mathbf{z})\})\ \times\ \mathbf{z}$$

if $D(...) \ne \emptyset$ and $Y_{k+1} \equiv \emptyset$ otherwise.

That is, Y_{k+1} is a collection of hybrid states $(Y_{k+1}|_{\mathbf{z}}, \mathbf{z})$ where $\mathbf{z} \in \mathbf{Z}_{k+1}$, and where $Y_{k+1}|_{\mathbf{z}}$ is an answer set of a regular ASP program composed of the active reducts of the stationary rules that can contribute to $\mathbf{z}$ and the heads of the advancing rules that derive $\mathbf{z}$.

Theorem 3. *M is an answer set of P w.i.c. J iff there is advancing selector F and a stationary selector D such that $\bigcup_{i=0}^{\infty} Y_i = M$ with F and D.*

Sketch of a proof. We begin by specifying a sequence of splitting sets $\langle U_i \rangle_{i=0}^{\infty}$ defined as

$$U_i = Lit_{At} \times \{\mathbf{p} :\ \mathbf{p} \in S \text{ and } 0 \le t(\mathbf{p}) \le i\Delta t\ \}$$

We then first show that Y_0 is an answer set of $b_{U_0}(P)$ w.i.c. J. The rules that can contribute to Y_0 are stationary-1 rules r such as $CS(r) \cap Bool(r) \cap J[0] \ne \emptyset$. These rules will contribute regular ASP rules to $\mathrm{Red}_{App}(P,\ \emptyset,\ \mathbf{z})$ for every $\mathbf{z} \in J[0]$. We then show that $D(\emptyset,\ \mathbf{z},\ \mathrm{Red}_{App}(P,\emptyset,\mathbf{z}))$ is an answer set of $\mathrm{Red}_{App}(P,\emptyset,\mathbf{z})$ iff $D(\emptyset,\ \mathbf{z},\ \mathrm{Red}_{App}(P,\emptyset,\mathbf{z})) \times \mathbf{z}$ is an answer set of $b_{U_0}(P)$ w.i.c. J.

The rest is proven by induction using The Splitting Sequence Theorem. That is $M[k+1]$ is an answer set of $E = \varepsilon_{U_k}(b_{U_{k+1}}(P) \setminus Rules_b(U_k, b_{U_{k+1}}(P)) \cup \mathrm{Rem}(U_k, b_{U_{k+1}}(P)),\ \bigcup_{i\le k} M[i])$ w.i.c. $GP_J(L)$, where $L = \bigcup_{i\le k} M[i]$ iff there exists an advancing selector F and a stationary selector D such that $M[k+1]$ is equal to Y_{k+1} as defined by the algorithm.

For the forward direction of the inductive step we define $F(N,\ Y) \equiv Y \cap GP(M)$. We define

$$D(N,\ \mathbf{p},\ Q) \equiv \begin{cases} At(N|_{\mathbf{p}}) \text{ if } At(N|_{\mathbf{p}}) \text{ is an answer set of } Q \\ \emptyset \text{ otherwise} \end{cases}$$

We then show $GP(M[k+1]) = NextGP(P,F,L,k\Delta t)$. We then use the induction on one-step provability operator $T\left[E^{M[k+1],\,GP_J(L)}, GP_J(L)\right]^j$ to show that if $M[k+1]$ is an answer set of E w.i.c. $GP_J(L)$ then $M[k+1]|_{\mathbf{p}} = Y_{k+1}|_{\mathbf{p}}$. That is we show that if $M[k+1]$ is an answer set of E w.i.c. $GP_J(L)$ then the algorithm derives it as Y_{k+1}.

For the reverse direction we first show $\{(head\,(r),\ \mathbf{p}) :\ r \in Head_{Adv}(P,\ L,\ \mathbf{p}),\ \mathbf{p} \in GP\,(Y_k)\} \subseteq T\left[E^{Y_{k+1},GP_J(L)}, GP_J\,(L)\right]^1(\emptyset)$. That is we show that the literals of $Head_{Adv}(P,\ L,\ \mathbf{p})$ are also derived by E at $\mathbf{p}$. We then use induction on one step provability $T\left[K^{At(Y_{k+1}|\mathbf{p})}\right]^i$, where $K \equiv \mathrm{Red}_{App}(P,L,\mathbf{p})$ to show that for all $\mathbf{p} \in GP\,(Y_{k+1})$ it is the case that $\bigcup_{i\geq 0} T\left[K^{At(Y_{k+1}|\mathbf{p})}\right]^i(\emptyset) \times \mathbf{p} \subseteq \bigcup_{j\geq 0} T\left[E^{Y_{k+1},GP_J(L)}, GP_J\,(L)\right]^j(\emptyset)$, for some j. That is, we show that the literals derived by the regular ASP program $\mathrm{Red}_{App}(P,L,\mathbf{p})$ are also derived by E at $\mathbf{p}$. But this merely shows that $Y \equiv \bigcup_{i=0}^{\infty} Y_i \subseteq \bigcup_{j\geq 0} T[P^{Y,J},J](\emptyset)$. We also need to show that $\bigcup_{j\geq 0} T[P^{Y,J},J](\emptyset) \subseteq Y$.

We do that by using induction on one step provability operator $T\left[E^{Y_{k+1},GP_J(L)}, GP_J\,(L)\right]^j$ to show that for all $\mathbf{p} \in GP\,(Y_{k+1})$ it is the case that $\bigcup_{j\geq 0} T\left[E^{Y_{k+1},GP_J(L)}, GP_J\,(L)\right]^j(\emptyset)$ is a subset of $\bigcup_{i\geq 0} T\left[K^{At(Y_{k+1}|\mathbf{p})}\right]^i(\emptyset) \times \mathbf{p}$.

This completes the proof of the theorem. □

The algorithm computes an answer set of the Hybrid ASP program P w.i.c. J inductively, by computing a subset of the answer set at time 0, then at time Δt, and so on through time $k\Delta t$. Moreover, the aglorithm reduces the process of computing an answer set of a Hybrid ASP program to the repeated application of two processes: the process of computing the set of ”next” generalized positions, and the process of computing an answer set of a regular ASP program derived from advancing and stationary Hybrid ASP rules applicable at these ”next” generalized positions.

It’s worth noting that the algorithm is a more general form of The Local Algorithm [5], variation of which is also discussed in [2].

4 Conclusion

The paper presents The Splitting Set Theorem for Hybrid ASP, which is the equivalent for Hybrid ASP of the Splitting Set Theorem [12], and the Splitting Sequence Theorem for Hybrid ASP (which is the equivalent for Hybrid ASP of The Splitting Sequence Theorem). The original Splitting Set Theorem proved to be a widely used result. It is the author’s hope that the new theorem will likewise prove to have many applications. The paper discusses one of the applications of the theorems to computing answer sets of Hybrid ASP programs.

Acknowledgements. The author would like to thank Jori Bomanson for insightful comments that helped to enhance the paper.

References

[1] Marcello Balduccini & Michael Gelfond (2003): *Diagnostic reasoning with A-Prolog*. *TPLP* 3(4-5), pp. 425–461. Available at `http://dx.doi.org/10.1017/S1471068403001807`.

[2] Jori Bomanson & Alex Brik (2019): *Diagnosing Data Pipeline Failures Using Action Languages*. In Marcello Balduccini, Yuliya Lierler & Stefan Woltran, editors: *Logic Programming and Nonmonotonic Reasoning - 15th International Conference, LPNMR 2019, Philadelphia, PA, USA, June 3-7, 2019, Proceedings*, *Lecture Notes in Computer Science* 11481, Springer, pp. 181–194, doi:10.1007/978-3-030-20528-7_14.

[3] Piero A. Bonatti (2008): *Erratum to: Reasoning with infinite stable models [Artificial Intelligence 156 (1) (2004) 75-111]*. *Artif. Intell.* 172(15), pp. 1833–1835, doi:10.1016/j.artint.2008.07.002.

[4] Alex Brik & Jeffrey B. Remmel (2011): *Hybrid ASP*. In John P. Gallagher & Michael Gelfond, editors: *ICLP (Technical Communications)*, *LIPIcs* 11, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, pp. 40–50. Available at `http://dx.doi.org/10.4230/LIPIcs.ICLP.2011.40`.

[5] Alex Brik & Jeffrey B. Remmel (2012): *Computing a Finite Horizon Optimal Strategy Using Hybrid ASP*. In: *14th International Workshop on Non-Monotonic Reasoning, NMR 2012, Rome, Italy, June 8-10, 2012. Procedings*.

[6] Alex Brik & Jeffrey B. Remmel (2015): *Diagnosing Automatic Whitelisting for Dynamic Remarketing Ads Using Hybrid ASP*. In Francesco Calimeri, Giovambattista Ianni & Miroslaw Truszczynski, editors: *Logic Programming and Nonmonotonic Reasoning - 13th International Conference, LPNMR 2015, Lexington, KY, USA, September 27-30, 2015. Proceedings*, *Lecture Notes in Computer Science* 9345, Springer, pp. 173–185, doi:10.1007/978-3-319-23264-5.

[7] Alex Brik & Jeffrey B. Remmel (2017): *Action Language Hybrid AL*. In Marcello Balduccini & Tomi Janhunen, editors: *Logic Programming and Nonmonotonic Reasoning - 14th International Conference, LPNMR 2017, Espoo, Finland, July 3-6, 2017, Proceedings*, *Lecture Notes in Computer Science* 10377, Springer, pp. 322–335, doi:10.1007/978-3-319-61660-5_29.

[8] Alex Brik & Jeffrey Xu (2020): *Diagnosing Data Pipeline Failures Using Action Languages: A Progress Report*. In Ekaterina Komendantskaya & Yanhong Annie Liu, editors: *Practical Aspects of Declarative Languages - 22nd International Symposium, PADL 2020, New Orleans, LA, USA, January 20-21, 2020, Proceedings*, *Lecture Notes in Computer Science* 12007, Springer, pp. 73–81, doi:10.1007/978-3-030-39197-3_5.

[9] Steve Dworschak, Susanne Grell, Victoria J. Nikiforova, Torsten Schaub & Joachim Selbig (2008): *Modeling Biological Networks by Action Languages via Answer Set Programming*. *Constraints* 13(1-2), pp. 21–65, doi:10.1007/s10601-007-9031-y.

[10] Maarten H. van Emden & Robert A. Kowalski (1976): *The Semantics of Predicate Logic as a Programming Language*. *J. ACM* 23(4), pp. 733–742. Available at `http://doi.acm.org/10.1145/321978.321991`.

[11] Michael Gelfond & Vladimir Lifschitz (1988): *The Stable Model Semantics for Logic Programming*. In: *ICLP/SLP*, pp. 1070–1080.

[12] Vladimir Lifschitz & Hudson Turner (1994): *Splitting a Logic Program*. In Pascal Van Hentenryck, editor: *Logic Programming, Proceedings of the Eleventh International Conference on Logic Programming, Santa Marherita Ligure, Italy, June 13-18, 1994*, MIT Press, pp. 23–37.