# Oscillating Behavior of Logic Programs

Katsumi Inoue[1] and Chiaki Sakama[2]

[1] National Institute of Informatics,
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
[2] Department of Computer and Communication Sciences, Wakayama University,
Sakaedani, Wakayama 640-8510, Japan

**Abstract.** We examine oscillation behavior of normal logic programs. Both the Gelfond-Lifschitz operator and the $T_P$ operator are used to update Herbrand interpretations, and any interpretation finally reaches in an oscillator. It has been shown that the supported model semantics of normal logic programs can characterize point attractors of Boolean networks. We here newly define supported classes of normal logic programs to investigate periodic oscillation induced by the $T_P$ operator, and apply them to characterize cycle attractors of Boolean networks. We also relate stable classes and supported classes of normal logic programs.

## 1 Introduction

A logic program can be viewed as a state transition system, in which each rule in the program represents how an entity (atom) is affected by other entities. Given an Herbrand interpretation as an initial state of the entities, a logic program defines the next state as an Herbrand interpretation through an operator associated with the program. It has been observed in [5,14] that both the $T_P$ operator [26,3] and the Gelfond-Lifschitz operator [12,6] exhibit oscillating behavior for normal logic programs. This paper investigates such behavior in detail.

For a definite program $P$, van Emden and Kowalski [26] considered the $T_P$ *operator* for $P$. This $T_P$ operator is monotone, so that any interpretation finally reaches a fixpoint. In particular, the empty set $\emptyset$ reaches the least fixpoint, $T_P \uparrow \omega$, which is the *least model* of $P$. For a normal logic program $P$, however, $T_P$ is not monotone. Apt, Blair and Walker [3] showed that a fixpoint of $T_P$ for a normal logic program $P$ is a *supported model* of $P$, and vice versa.

Given an Herbrand interpretation $I$, Baral and Subrahmanian [5,6] defined an operator for a normal logic program $P$ called the $F_P$ *operator* such that $F_P(I) = T_{P^I} \uparrow \omega$, where $P^I$ is the *Gelfond-Lifschitz reduct* of $P$ with respect to $I$ [12]. By definition, a fixpoint of $F_P$ is a *stable model* of $P$. This $F_P$ is anti-monotone, so ${F_P}^2$ is monotone. Then, [5] defined a *stable class* of $P$, which is a set $\mathcal{S}$ of Herbrand interpretations such that $\mathcal{S} = \{F_P(I) \mid I \in \mathcal{S}\}$. A stable model corresponds to a singleton stable class, i.e., $|\mathcal{S}| = 1$. It has been proved that every normal logic program has at least one stable class [5] and that the *well-founded semantics* [27] corresponds to a stable class that is minimal with respect to the Hoare ordering [6].

E. Erdem et al. (Eds.): Lifschitz Festschrift, LNCS 7265, pp. 345–362, 2012.

Recently, Inoue [14] revealed that the $T_P$ operator for a normal logic program $P$ precisely captures the synchronous update of *Boolean networks*, which have been used as a mathematical model of genetic networks and complex adaptive systems [18,19]. The stable states and dynamics of Boolean networks are characterized by their *attractors*. Then, the supported model semantics of normal logic programs captures the *point attractors* of Boolean networks. On the other hand, *cycle attractors* of Boolean networks represent periodic oscillation of the $T_P$ operator. By this way, [14] observes the similarity between normal logic programs and Boolean networks. Actually, any normal logic program $P$ can be converted to a Boolean network whose point attractors correspond to the supported models of $P$. Yet, [14] has not presented to what classes of interpretations cycle attractors precisely correspond in the semantics of logic programs.

In this paper, we first revisit the stable class semantics of normal logic programs, and analyze the oscillation behavior of them. Next, the *supported classes* of normal logic programs are newly defined to extend the supported models so that the supported models correspond to the singleton supported classes. This extension is made in a similar way that the stable classes are defined by extending the stable models in [5,6]. Then, we characterize both point attractors and cycle attractors of Boolean networks by supported classes of corresponding logic programs. This result also extends the previous result in [14], which characterizes point attractors by supported models. We also investigate the relationships between stable classes and supported classes of normal logic programs, thereby illustrating different oscillating behaviors of programs. Moreover, stable classes of a program can be shown to be translated into supported classes of a prerequisite-free program, which enables us to compute stable classes via computation of attractors of the corresponding Boolean network.

In the rest of this paper, Section 2 revisits the stable class semantics of normal logic programs and characterizes it in a different way. Section 3 defines the supported class semantics of normal logic programs and shows its relation to supported models. Section 4 considers attractors of Boolean networks and their relations to supported classes of logic programs. Section 5 presents more relationships between stable classes and supported classes of normal logic programs, and discusses their computational issues.

## 2 Stable Classes

We consider a first-order language and denote the Herbrand base (the set of all ground atoms) as $\mathcal{B}$. A *normal logic program* (NLP) is a set of rules of the form

$$A \leftarrow A_1 \wedge \cdots \wedge A_m \wedge \neg A_{m+1} \wedge \cdots \wedge \neg A_n \tag{1}$$

where $A$ and $A_i$'s are atoms ($n \geq m \geq 0$). For any rule $R$ of the form (1), the atom $A$ is called the *head* of $R$ and is denoted as $h(R)$, and the conjunction to the right of $\leftarrow$ is called the *body* of $R$ and we represent the positive and negative literals in the body as $b^+(R) = \{A_1, \ldots, A_m\}$ and $b^-(R) = \{A_{m+1}, \ldots, A_n\}$,

respectively. An NLP is also called a *logic program* or a *program* in this paper. An NLP $P$ is a *definite program* if $b^-(R) = \emptyset$ for every rule $R$ in $P$.

Let $ground(P)$ be the set of ground instances of all rules in a logic program $P$. An (*Herbrand*) *interpretation* $I$ is a subset of $\mathcal{B}$, and is called an (*Herbrand*) *model* of $P$ if $I$ *satisfies* all ground rules from $P$, that is, for any rule $R \in ground(P)$, $b^+(R) \subseteq I$ and $b^-(R) \cap I = \emptyset$ imply $h(R) \in I$.

For a definite program $P$, there exists a unique minimal model [26], i.e., the *least model* of $P$. It is characterized by the *immediate consequence operator* (or $T_P$ *operator*), which is defined as a mapping $T_P : 2^{\mathcal{B}} \to 2^{\mathcal{B}}$:

$$T_P(I) = \{\, h(R) \mid R \in ground(P), \text{ the body of } R \text{ is true in } I \,\}. \tag{2}$$

If $P$ is a definite program, then $T_P(I)$ for an Herbrand interpretation $I$ becomes

$$T_P(I) = \{\, h(R) \mid R \in ground(P),\, b^+(R) \subseteq I \,\}. \tag{3}$$

The ordinal powers of $T_P$ are defined as:

$$\begin{aligned} T_P \uparrow 0 &= \emptyset, \\ T_P \uparrow n+1 &= T_P(T_P \uparrow n), \\ T_P \uparrow \omega &= \textstyle\bigcup_{n \in \omega} T_P \uparrow n, \end{aligned}$$

where $n$ is a successor ordinal and $\omega$ is a limit ordinal. Note that the mapping $T_P$ is monotone for any definite program $P$. Then $T_P \uparrow \omega$ is the least fixpoint of $T_P$, which is exactly the least model of $P$ [26].

The stable model semantics was defined by Gelfond and Lifschitz [12] through the reduct of a program. Given a logic program $P$ and an Herbrand interpretation $I$, the *reduct* of $P$ relative to $I$ is defined as the definite program:

$$P^I = \left\{ \left( h(R) \leftarrow \bigwedge_{B \in b^+(R)} B \right) \;\middle|\; R \in ground(P),\, b^-(R) \cap I = \emptyset \right\}.$$

An Herbrand model $I$ is a *stable model* [12] of $P$ if $I$ is the least model of $P^I$, i.e., $T_{P^I} \uparrow \omega = I$. Since $P^I = P$ holds for any definite program $P$ and any Herbrand interpretation $I$, the unique stable model of a definite program is its least model.

**Definition 2.1 (Baral and Subrahmanian [5,6]).** Given a logic program $P$ and an Herbrand interpretation $I$, the *Gelfond-Lifschitz operator* (or the $F_P$ *operator*) is defined as:

$$F_P(I) = T_{P^I} \uparrow \omega. \tag{4}$$

A non-empty set $\mathcal{S}$ of Herbrand interpretations[1] is a *stable class* of $P$ iff it holds that

$$\mathcal{S} = \{F_P(I) \mid I \in \mathcal{S}\}. \tag{5}$$

A stable class $\mathcal{S}$ of $P$ is *strict* iff no proper subset of $\mathcal{S}$ is a stable class of $P$.

[1] The non-emptiness condition on $\mathcal{S}$ is necessary, since $\mathcal{S} = \emptyset$ satisfies (5). This condition is stated in [30]. On the other hand, the definition in [6] imposes the condition that $\mathcal{S}$ is finite to avoid an infinite transition. We can allow an infinite $\mathcal{S}$ in this section, but later the finiteness is required in Section 4.

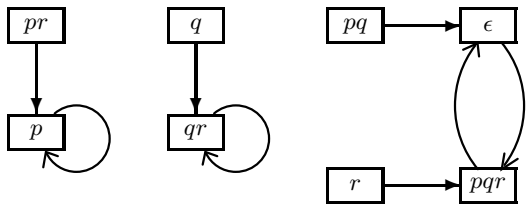


**Fig. 1.** State transition graph induced by $F_{P_1}$

A stable model corresponds to a singleton stable class $\mathcal{S}$, i.e., $|\mathcal{S}| = 1$. According to [5], every NLP has at least one stable class. It is easy to see that the union of two stable classes of $P$ is also a stable class of $P$. Hence, strict stable classes are meaningful to represent such minimal cores.

**Example 2.1.** Consider the logic program $P_1$:

$$p \leftarrow \neg q,$$
$$q \leftarrow \neg p,$$
$$r \leftarrow q.$$

There are three strict stable classes of $P_1$:

$$S_1 = \{\{p\}\}, \quad S_2 = \{\{q, r\}\}, \quad S_3 = \{\emptyset, \{p, q, r\}\}.$$

$S_1$ and $S_2$ correspond to the stable models of $P$. These stable classes can be graphically represented [6] as cycles in a *state transition graph* in which nodes are Herbrand interpretations[2] and there is a directed arc from interpretation $I$ to interpretation $J$ iff $J = F_P(I)$. The state transition graph induced by $F_{P_1}$ is depicted in Figure 1. The stable classes $S_1$ and $S_2$ respectively correspond to the self-loops connecting to $p$ and $qr$, and $S_3$ is represented by the non-singleton cycle $\epsilon \rightarrow pqr \rightarrow \epsilon$.

We now give an alternative characterization for the strict stable classes. A sequence of applications of an operator on the Herbrand interpretations is called an *orbit* [7]. Given a logic program $P$ and an interpretation $I$, the *orbit* of $I$ with respect to the $T_P$ operator is the sequence $\langle {T_P}^k(I)\rangle_{k\in\omega}$, where ${T_P}^0(I) = I$ and ${T_P}^{k+1}(I) = T_P({T_P}^k(I))$ for $k = 0, 1, \ldots$. Similarly, we can now define the *orbit* of $I$ with respect to the $F_P$ operator as the sequence $\langle {F_P}^k(I)\rangle_{k\in\omega}$. Put

$$\mathcal{F}_P(I) = \{{F_P}^k(I) \mid k \in \omega\}. \tag{6}$$

**Theorem 2.1.** *A non-empty set $\mathcal{S}$ of Herbrand interpretations is a strict stable class of a logic program $P$ iff $\mathcal{F}_P(I) = \mathcal{S}$ for every $I \in \mathcal{S}$.*

*Proof.* For any $I \in \mathcal{S}$, $F_P(I) = {F_P}^1(I) \in \mathcal{F}_P(I)$ holds. Then, $\{F_P(I) \mid I \in \mathcal{S}\} \subseteq \bigcup_{I\in\mathcal{S}} \mathcal{F}_P(I)$. Suppose that $\mathcal{F}_P(I) = \mathcal{S}$ for every $I \in \mathcal{S}$. Then, $\bigcup_{I\in\mathcal{S}} \mathcal{F}_P(I) =$

[2] Each interpretation is represented as a sequence of atoms instead of a set of atoms in the graph, e.g., $pq$ means $\{p, q\}$ and the empty string $\epsilon$ means $\emptyset$.

$\bigcup_{I\in\mathcal{S}} \mathcal{S} = \mathcal{S}$. Hence, $\{F_P(I) \mid I \in \mathcal{S}\} \subseteq \mathcal{S}$. Assume that $\mathcal{S} \setminus \{F_P(I) \mid I \in \mathcal{S}\} \neq \emptyset$. Then, there exists $I \in \mathcal{S}$ such that $I \neq F_P(J)$ for every $J \in \mathcal{S}$. Since $J \in \mathcal{S}$, $I \in \mathcal{F}_P(J) = \mathcal{S}$, then $I = {F_P}^k(J)$ for some $k > 1$. But $I = F_P({F_P}^{k-1}(J))$ and ${F_P}^{k-1}(J) \in \mathcal{S}$, a contradiction. Hence, $\{F_P(I) \mid I \in \mathcal{S}\} = \mathcal{S}$. Therefore, $\mathcal{S}$ is a stable class of $P$. Now assume that there exists a subset $\mathcal{S}' \subset \mathcal{S}$ such that $\mathcal{S}'$ is a stable class of $P$, i.e., $\mathcal{S}' = \{F_P(I) \mid I \in \mathcal{S}'\}$. Take any $I \in \mathcal{S}'$. By $I \in \mathcal{S}$, $\mathcal{F}_P(I) = \mathcal{S}$ holds. There must be $J \in \mathcal{S} \setminus \mathcal{S}'$, but $J = {F_P}^k(I)$ holds for some $k > 1$. Then, ${F_P}^{k-1}(I) \notin \mathcal{S}'$, since otherwise $J$ belongs to $\mathcal{S}$. Similarly, ${F_P}^{k-2}(I) \notin \mathcal{S}'$ must hold. Repeating this inference we reach to the conclusion that ${F_P}^1(I) \notin \mathcal{S}'$, a contradiction. Hence, $\mathcal{S}$ is a strict stable class of $P$.

Conversely, suppose that $\mathcal{S}$ is a strict stable class of $P$. Since $\mathcal{S}$ is a stable class of $P$, $\mathcal{S} = \{F_P(I) \mid I \in \mathcal{S}\}$ holds. By ${F_P}^0(I) = I \in \mathcal{F}_P(I)$, $\mathcal{S} \subseteq \bigcup_{I\in\mathcal{S}} \mathcal{F}_P(I)$ holds. Assume that there exists $J \in \bigcup_{I\in\mathcal{S}} \mathcal{F}_P(I) \setminus \mathcal{S}$. Then, $J = {F_P}^k(I)$ for some $I \in \mathcal{S}$ and $k > 1$. Then, ${F_P}^{k-1}(I) \notin \mathcal{S}$, since otherwise $J$ belongs to $\mathcal{S}$. Similarly, ${F_P}^{k-2}(I) \notin \mathcal{S}$ must hold. Repeating this inference we reach to the conclusion that ${F_P}^1(I) \notin \mathcal{S}$, a contradiction. Hence, $\mathcal{S} = \bigcup_{I\in\mathcal{S}} \mathcal{F}_P(I)$.

Now assume that $\mathcal{F}_P(I) \neq \mathcal{S}$ for some $I \in \mathcal{S}$. By $\mathcal{S} = \bigcup_{J\in\mathcal{S}} \mathcal{F}_P(J)$, ${F_P}^k(I)$ must belong to $\mathcal{S}$ for every $k \geq 0$. Hence, $\mathcal{F}_P(I) \subset \mathcal{S}$ holds. Let $\mathcal{S}' = \mathcal{S} \setminus \mathcal{F}_P(I)$. Take any $J \in \mathcal{S}'$. Since $J \notin \mathcal{F}_P(I)$, $F_P(J) \notin \mathcal{F}_P(I)$ holds too. Then, $F_P(J) \in \mathcal{S}'$. Hence, $\mathcal{S}' = \{F_P(J) \mid J \in \mathcal{S}'\}$ holds. This means that $\mathcal{S}'$ is a stable class of $P$. However this is impossible, since $\mathcal{S}$ is a strict stable class of $P$. Therefore, $\mathcal{F}_P(I) = \mathcal{S}$ for every $I \in \mathcal{S}$. □

Theorem 2.1 gives us a new insight into strict stable classes. The original definition of strict stable classes [5] is based on the minimality of stable classes with respect to set inclusion of interpretations. Such minimality is guaranteed if no redundant interpretation is contained in a stable set. The irredundancy of a strict stable class $\mathcal{S}$ is explained in Theorem 2.1 by the property that those interpretations appearing in the orbit of any interpretation in $\mathcal{S}$ exactly coincide with $\mathcal{S}$. Hence, a strict stable class $\mathcal{S}$ composes a *connected component* in the state transition graph.[3] On the other hand, a non-strict stable class must consist of multiple connected components. Hence, a strict stable class represents a minimal oscillation between interpretations.

Since the mapping $F_P$ is deterministic for any NLP $P$, we can further guarantee that the orbit of any interpretation belonging to a *finite* strict stable class $\mathcal{S}$ repeats a sequence of the interpretations appearing in $\mathcal{S}$ always in the same order. In other words, the state transition graph for $\mathcal{S}$ becomes a *directed cycle*, which is shown in the following strengthened result.[4]

**Theorem 2.2.** *Let $P$ be a logic program, and $\mathcal{S}$ a finite set of Herbrand interpretations of $P$. Then, $\mathcal{S}$ is a strict stable class of $P$ iff there is a directed cycle*

[3] In other words, $\mathcal{S}$ is the set of interpretations that are *reachable* from every interpretation in $\mathcal{S}$ in the state transition graph. This property is inspired by the definition of *attractors* in Boolean networks [11,14], and we adapt the notion into stable classes.

[4] Theorem 2.2 is essentially equivalent to [5, Theorem 3]. We here give a proof which can be applied to a similar result for supported classes in Section 3.

$I_1 \to I_2 \to \cdots \to I_k \to I_1$ $(k \geq 1)$ *in the state transition graph induced by* $F_P$ *such that* $\{I_1, I_2, \ldots, I_k\} = \mathcal{S}$.

*Proof.* If there is a directed cycle satisfying the condition of the proposition, then $\mathcal{F}_P(I_i) = \mathcal{S}$ holds for every $I_i \in \mathcal{S}$ $(i = 1, \ldots, k)$. Then, $\mathcal{S}$ is a strict stable class of $P$ by Theorem 2.1.

Conversely, let $\mathcal{S}$ be a strict stable class of $P$ such that $\mathcal{S} = \{I_1, I_2, \ldots, I_k\}$ $(k > 0)$. Then, $\mathcal{F}_P(I_i) = \mathcal{S}$ for every $i = 1, \ldots, k$ by Theorem 2.1. Suppose further that, for each $i = 1, \ldots, k$, $F_P(I_i) = I_{s(i)} \in \mathcal{S}$ for some $1 \leq s(i) \leq k$. Since $\mathcal{S}$ is strict, $\{I_{s(i)} \mid 1 \leq i \leq k\} = \mathcal{S}$ and all $I_{s(i)}$'s are different from each other. Then, there is a 1-1 correspondence between $\{I_{s(i)} \mid 1 \leq i \leq k\}$ and $\mathcal{S}$, that is, $\mathcal{S}$ is a cyclic group, which constitutes a directed cycle $I_1 \to I_{s(1)} \to I_{s(s(1))} \to \cdots \to I_{s^k(1)} \to I_1$ in the state transition graph induced by $F_P$. □

Note that the condition being a strict stable class in Theorem 2.1 is weaker than that in Theorem 2.2, since the former is stated in terms of an unordered set. Then, equivalence of strict stable classes in two programs does not imply equivalence of their state transition graphs.[5]

**Example 2.2.** Consider the logic program $P_2$ containing a cycle with three negations:

$$p \leftarrow \neg q,$$
$$q \leftarrow \neg r,$$
$$r \leftarrow \neg p.$$

$P_2$ has no stable model but has two strict stable classes:

$$S_3 = \{\emptyset, \{p, q, r\}\}, \quad S_4 = \{\{p\}, \{p, q\}, \{q\}, \{q, r\}, \{r\}, \{p, r\}\}.$$

Let us consider the logic program $P_3$ obtained by taking the contrapositive of each rule in $P_2$:

$$q \leftarrow \neg p,$$
$$r \leftarrow \neg q,$$
$$p \leftarrow \neg r.$$

$P_3$ has exactly the same stable classes as $P_2$. However, the directed cycle for $S_2$ in $P_2$ is $p \to pq \to q \to qr \to r \to pr \to p$, while that in $P_3$ is in the reverse order, i.e., $p \to pr \to r \to qr \to q \to pq \to p$.

Since all meaningful stable classes are strict, from now on we simply call each strict stable class as a *stable class* as long as there is no confusion. Each (strict) stable class is minimal in the sense of set inclusion over the sets of stable classes. Over the sets of strict stable classes of a logic program, however, we can further define preference relations. For example, we can prefer those minimal interpretations containing as few atoms as possible. Based on this intuition, stable classes have been used to represent several semantics for NLPs as follows.

[5] To remedy this problem, we could define a stable class as a directed cycle satisfying the condition in Theorem 2.2, while we follow the original definition of [5] here.

- Baral and Subrahmanian [5] prefer the stable classes that are minimal with respect to the *Smyth ordering* $\preceq_s$.[6] For Example 2.1, $S_1$ and $S_2$ are the $\preceq_s$-minimal (strict) stable classes, which correspond to the stable model semantics in this case.
- Baral and Subrahmanian [6] show that the *well-founded semantics* [27] corresponds to a stable class that is minimal with respect to the *Hoare ordering* $\preceq_h$. For Example 2.1, $S_3$ is the $\preceq_h$-minimal (strict) stable classes, in which the truth values of $p, q, r$ are undefined, that is, they are true in $\{p, q, r\} \in S_3$ and are false in $\emptyset \in S_3$.
- You and Yuan [30] use results from [16] and show that a $\preceq_s$-minimal stable class $\mathcal{S}$ such that (i) $\mathcal{S}$ is a maximal fixpoint of the ${F_P}^2$ operator, and that (ii) $\mathcal{S} = \{I, F_P(I)\}$ for some $I \in 2^{\mathcal{B}}$ satisfying $I \subseteq F_P(I)$ corresponds to a *partial stable model* [23], a *preferred extension* [9], and a *regular model* [29].

These correspondences with various semantics of logic programming are an important aspect of the stable class semantics. In this paper, we further focus on another feature that has never been explored in detail. That is, we regard that the oscillating behavior is very useful in representing dynamic systems. This aspect will be highlighted in the following sections where supported classes are defined and are applied to represent dynamic systems.

## 3 Supported Classes

An Herbrand interpretation $I \in 2^{\mathcal{B}}$ is *supported* in an NLP $P$ if for any ground atom $A \in I$, there exists a rule $R \in ground(P)$ such that $h(R) = A$, $b^+(R) \subseteq I$, and $b^-(R) \cap I = \emptyset$. $I$ is a *supported model* of $P$ if $I$ is a model of $P$ and is supported in $P$ [3]. It is known that every stable model is a supported model [21], but not vice versa. For example, the logic program $\{\, p \leftarrow p,\ q \leftarrow \neg p \,\}$, has the supported models $\{p\}$ and $\{q\}$, but only the latter is its stable model.

For a logic program $P$ and an Herbrand interpretation $I$, Apt *et al.* [3] have defined the $T_P$ operator $T_P : 2^{\mathcal{B}} \to 2^{\mathcal{B}}$ as

$$T_P(I) = \{\, h(R) \mid R \in ground(P),\ b^+(R) \subseteq I,\ b^-(R) \cap I = \emptyset \}. \tag{7}$$

Actually, equation (7) is equivalent to equation (2), and is a generalization of (3). Then, $I$ is a model of $P$ iff $I$ is a pre-fixed point of $T_P$, i.e., $T_P(I) \subseteq I$. By definition, $I$ is supported iff $I \subseteq T_P(I)$. Hence, $I$ is a supported model of $P$ iff $I$ is a fixpoint of $T_P$, i.e., $T_P(I) = I$. Moreover, $I$ is a model of $Comp(P)$, which is the Clark's *completion* of $P$, iff $T_P(I) = I$. This means that the supported models of $P$ are precisely the models of $Comp(P)$.

Now we give the definition of the supported class semantics. Remember that $I$ is a stable model of $P$ iff $F_P(I) = I$ and that a stable class is defined as a fixpoint over sets of interpretations. The supported classes are then defined

[6] Given a pre-ordered set $(D, \preccurlyeq)$ and $X, Y \subseteq D$, the *Smyth order* is defined as $X \preceq_s Y$ iff $(\forall x \in X)(\exists y \in Y)\, x \preccurlyeq y$, and the *Hoare order* is defined as $X \preceq_h Y$ iff $(\forall y \in Y)(\exists x \in X)\, x \preccurlyeq y$ [6]. Note that these orders are defined reversely in [13].

in the same way as the stable classes using the $T_P$ operator instead of the $F_P$ operator in (5).

**Definition 3.1.** A *supported class* of a logic program $P$ is defined as a non-empty set $\mathcal{S}$ of Herbrand interpretations satisfying the fixpoint equation:

$$\mathcal{S} = \{T_P(I) \mid I \in \mathcal{S}\}. \tag{8}$$

A supported class $\mathcal{S}$ of $P$ is *strict* if no proper subset of $\mathcal{S}$ is a supported class of $P$.[7]

Alternatively, we can define the strict supported classes in the same way as Theorem 2.1. Given a logic program $P$ and an Herbrand interpretation $I$, let $\mathcal{T}_P(I)$ be the set of Herbrand interpretations appearing in the orbit $\langle {T_P}^k(I)\rangle_{k\in\omega}$ of $I$ with respect to $T_P$:

$$\mathcal{T}_P(I) = \{{T_P}^k(I) \mid k \in \omega\}. \tag{9}$$

**Theorem 3.1.** *A non-empty set $\mathcal{S}$ of Herbrand interpretations is a strict supported class of a logic program $P$ iff $\mathcal{T}_P(I) = \mathcal{S}$ for every $I \in \mathcal{S}$.*

*Proof.* The proof of Theorem 2.1 can be applied here by substituting $F_P(I)$ and $\mathcal{F}_P(I)$ in Theorem 2.1 with $T_P(I)$ and $\mathcal{T}_P(I)$, respectively. □

As in the case of strict stable classes, Theorem 3.1 can be further strengthened by taking the cyclic group into account.

**Theorem 3.2.** *Let $P$ be a logic program, $\mathcal{S}$ a finite set of Herbrand interpretations of $P$. Then, $\mathcal{S}$ is a strict supported class of $P$ iff there is a directed cycle $I_1 \to I_2, \to \cdots \to I_k \to I_1$ $(k \geq 1)$ in the state transition graph induced by $T_P$ such that $\{I_1, I_2, \ldots, I_k\} = \mathcal{S}$.*

*Proof.* The proof of Theorem 2.2 can be applied here by respectively substituting $F_P(I)$ and $\mathcal{F}_P(I)$ in Theorem 3.1 with $T_P(I)$ and $\mathcal{T}_P(I)$. □

**Example 3.1.** Consider the program $P_1$ in Example 2.1. There are three strict supported classes of $P_1$:

$$S_1 = \{\{p\}\}, \quad S_2 = \{\{q, r\}\}, \quad S_5 = \{\{p, q\}, \{r\}\}.$$

$S_1$ and $S_2$ are also stable classes of $P$, but the stable class $S_3 = \{\emptyset, \{p, q, r\}\}$ is now replaced by $S_5$. The state transition graph induced by $T_{P_1}$ is depicted in Figure 2.

The next proposition states that two strict supported classes are orthogonal.

**Proposition 3.3.** *Suppose $\mathcal{S}$ and $\mathcal{S}'$ are strict supported classes of a logic program $P$ that has a finite Herbrand base. Then, $\mathcal{S} \neq \mathcal{S}'$ iff $\mathcal{S} \cap \mathcal{S}' = \emptyset$.*

[7] Again, the union of any two sets of interpretations satisfying (8) satisfies (8).

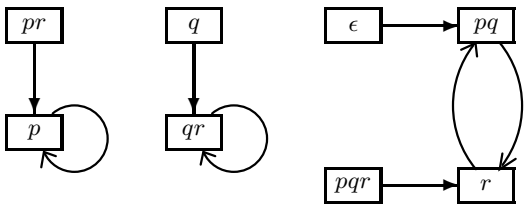


**Fig. 2.** State transition graph induced by $T_{P_1}$

*Proof.* The if direction is obvious. To prove the only-if direction, assume that $\mathcal{S} \cap \mathcal{S}' \neq \emptyset$ and there exists an Herbrand interpretation $I \in \mathcal{S} \cap \mathcal{S}'$. Applying Theorem 3.1 twice, we have $\mathcal{T}_P(I) = \mathcal{S}$ and $\mathcal{T}_P(I) = \mathcal{S}'$, and thus $\mathcal{S} = \mathcal{S}'$. □

As in the case of stable classes, all meaningful supported classes are strict. Hence, we will simply call each strict supported class as a *supported class* as long as there is no confusion.

**Proposition 3.4.** *$I$ is a supported model of a logic program $P$ iff $\{I\}$ is a supported class of $P$.*

*Proof.* $\{I\}$ is a supported class of $P$ iff $\mathcal{T}_P(I) = \{I\}$ iff $T_P(I) = I$ iff $I$ is a supported model of $P$. □

Since the $T_P$ operator is the immediate consequence operator, the utility of the supported class semantics can be seen in representing state transition systems, in which inference of stepwise changes is simulated by the $T_P$ operator.

**Example 3.2.** Consider the logic program $P_4$:

$$produce \leftarrow \neg stock,$$
$$stock \leftarrow produce,$$

which represents that a shortage of stock triggers an additional production and that a production causes a stock. Neither stable model nor supported model exists for $P_4$. Both the stable class $S_4 = \{\emptyset, \{produce, stock\}\}$ and the supported class $S_5 = \{\emptyset, \{produce\}, \{produce, stock\}, \{stock\}\}$ of $P_4$ can represent continuous changes of stocks. While $S_4$ represents the cycle $\epsilon \rightarrow produce{\cdot}stock \rightarrow \epsilon$, $S_5$ is a more refined cycle $\epsilon \rightarrow produce \rightarrow produce{\cdot}stock \rightarrow stock \rightarrow \epsilon$. Hence, supported classes can be used to represent time delay occurring in positive inference, while stable classes infer all consequences of the reduct at once.

## 4 Boolean Networks

In this section, we will see that the supported class semantics can be well applied to *Boolean networks*, which was proposed as a mathematical model of genetic networks and complex adaptive systems [18,19], and has been used as a discrete

model of gene regulatory, signal transduction and protein interaction networks. *Cellular automata* [28] are regarded as special cases of Boolean networks. Inoue [14] has shown that the $T_P$ operator is useful to characterize the dynamics of synchronous Boolean networks. Here, we extend the result of [14] and show that the attractors of Boolean networks can be completely characterized by supported classes of associated logic programs.

A *Boolean network* (BN) is a pair $N = (V, F)$, where $V = \{v_1, \ldots, v_n\}$ is a finite set of nodes and $F = \{f_1, \ldots, f_n\}$ is a corresponding set of Boolean functions.[8] Let $v_i(t)$ denote the value of $v_i$ at time $t$, which takes either 1 or 0. The overall expression level of all nodes in $N$ at time step $t$ is called a *state* of $N$ at $t$, and is expressed as a vector $\mathbf{v}(t) = (v_1(t), \ldots, v_n(t))$. The value of node $v_i$ at the next time step $t+1$ is determined by $v_i(t+1) = f_i(v_{i_1}(t), \ldots, v_{i_k}(t))$, where $v_{i_1}, \ldots, v_{i_k}$ are the set of *input nodes* of $v_i$, and the number $k$ is called the *indegree* of $v_i$. In this paper, we consider Boolean networks in which the value of each node is updated *synchronously*.[9]

A consecutive sequence of states obtained by state transitions is called a *trajectory* of $N$. Since any state transition at any time step is *deterministic*, the trajectory starting from any state is uniquely determined. Let $w \in \{0,1\}^n$ be a state, and $R_N(w)$ be the states reachable in the trajectory of $N$ starting from $w$. Then, a set $S$ of states is an *attractor* of $N$ if $R_N(w) = S$ holds for every $w \in S$ [11]. Any trajectory from a node in an attractor $S$ composes a single loop, $w_0, \ldots, w_{p-1}, w_p (= w_0)$, where $p = |S|$ $(1 \leq p \leq 2^n)$. The length $p$ is also called the *period* of the attractor $S$, and $S$ is called a *point attractor* (or *singleton attractor*) if $p = 1$; Otherwise $(p > 1)$, it is called a *cycle attractor*. The set of states that reach the same attractor is called its *basin of attraction* [19].

A Boolean network is often represented graphically with two types of edges, which are positive and negative, in which $u \longrightarrow v$ means that $u(t)$ positively takes part in the regulation function for $v(t+1)$ and $u \dashv v$ means that $u(t)$ negatively takes part in the regulation function for $v(t+1)$.[10]

**Example 4.1.** Consider the Boolean network $N_1 = (V_1, F_1)$, where $V_1 = \{p, q, r\}$ and $F_1$ is as follows.

$$\begin{aligned} p(t+1) &= q(t), \\ q(t+1) &= p(t) \wedge r(t), \\ r(t+1) &= \neg p(t). \end{aligned}$$

[8] In a gene regulation network, each node $v_i \in V$ represents a *gene* and each $f_i \in F$ is called the *regulation function* of $v_i$. A node $v_i \in V$ takes the value 1 if the gene is *expressed*, and takes the value 0 if it is not expressed.

[9] There are also *asynchronous Boolean networks* [11], in which not all nodes are necessarily updated at a time. Some asynchronous BNs can be encoded in ASP [14].

[10] A positive edge of the form $u \longrightarrow v$ is often called a *trigger* or an *activator* of $v$, and a negative edge of the form $u \dashv v$ is called an *inhibitor* or a *repressor* of $v$. Note that the exact Boolean function $f_i \in F$ for a node $v_i \in V$ is not captured with this graphical representation. Hence the set $F$ must be always shown with such a graph.

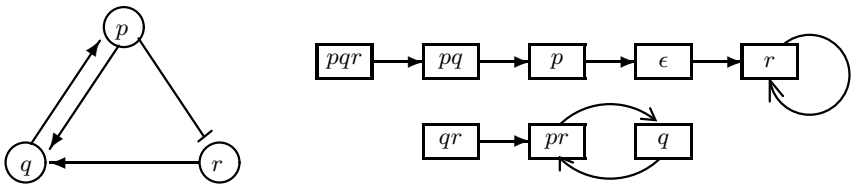


**Fig. 3.** Boolean network $N_1$ and its state transition diagram

The Boolean network $N_1$ is represented in Figure 3 (left). Starting from the initial state $\mathbf{v}(0) = (0, 1, 1)$, the trajectory becomes $(0, 1, 1)$, $(1, 0, 1)$, $(0, 1, 0)$, $(1, 0, 1)$, ..., and $pr \to q \to pr$ is a cycle attractor (Figure 3, right below).[11] $N_1$ has another, point attractor $r \to r$ (Figure 3, right above) whose basin of attraction is $\{pqr, pq, p, \epsilon, r\}$.

Note that any state in a Boolean network deterministically belongs to the basin of attraction of only one attractor. In applications to biological networks [19,24], each attractor represents a cell type, a type of memory, or a cellular state such as proliferation, apoptosis and differentiation.

### 4.1 From Boolean Networks to Logic Programs

We here show a translation of Boolean networks into normal logic programs. Given a Boolean network $N = (V, F)$, each Boolean function $f_i \in F$ for a node $v_i \in V$ is to be embedded into rules in a logic program. Then, $f_i$ should be transformed in disjunctive normal form (DNF). Since this is always possible, we assume that each $f_i \in F$ for a node $v_i \in V$ is already written as a DNF formula:

$$v_i(t+1) = \bigvee_{j=1}^{l_i} B_{i,j}(t), \quad B_{i,j}(t) = \bigwedge_{k=1}^{m_j} v_{i,j,k}(t) \ \wedge \bigwedge_{k=m_j+1}^{n_j} \neg v_{i,j,k}(t), \tag{10}$$

where $v_{i,j,k} \in V$ and $n_j \geq m_j \geq 0$ for $j = 1, \ldots, l_i$. Note that $l_i$ (and $j$) can be 0 for a node $v_i$, so that the indegree of $v_i$ is 0. In this case, $v_i$ is called a *constant node*, and cannot change its value. Let $V_C \subseteq V$ be the constant nodes of $N$.

According to [14], we can associate the *propositional* logic program $\pi(N)$ for a Boolean network $N$ as:

$$\pi(N) = \{\, (v_i \leftarrow B_{i,j}) \mid v_i \in (V \setminus V_C),\ 1 \leq j \leq l_i \,\} \cup \{\, (v_i \leftarrow v_i) \mid v_i \in V_C \,\}, \tag{11}$$

where $B_{i,j}$ is the conjunction of literals obtained from $B_{i,j}(t)$ in (10) by deleting the time argument $(t)$ appearing in every $v_{i,j,k}(t)$ $(k = 1, \ldots, n_j)$ in it. For any state $\mathbf{v}(t) = (v_1(t), \ldots, v_n(t))$ at time step $t$, the interpretation $I^t \subseteq V$ is defined as $I^t = \{v_i \in V \mid v_i(t) = 1\}$. Then, it holds that $I^{t+1} = T_{\pi(N)}(I^t)$.

[11] As in the case of state transition for logic programs, each state is represented as a sequence of nodes whose values are 1, e.g., $pq$ means $(p, q, r) = (1, 1, 0)$ and the empty string $\epsilon$ means $(0, 0, 0)$ in the state transition diagram for a Boolean network.

**Proposition 4.1.** [14] *Let $N$ be a Boolean network, and $\mathbf{v}(t)$ a state at some time step $t$. Then, the orbit of $I^t$ with respect to $T_{\pi(N)}$ is precisely the trajectory of $N$ starting from $\mathbf{v}(t)$. That is, $\mathcal{T}_{\pi(N)}(I^t) = R_N(\mathbf{v}(t))$ holds.*

Proposition 4.1 implies that every attractor can be obtained as an attracting cycle appearing in an orbit of $I^0$ that corresponds to some initial state $\mathbf{v}(0)$, and in particular that a point attractor corresponds to a fixed point in such an orbit.

**Example 4.2.** For the Boolean network $N_1$ in Example 4.1, the logic program $\pi(N_1)$ is given as:

$$\begin{aligned} &p \leftarrow q, \\ &q \leftarrow p \wedge r, \\ &r \leftarrow \neg p. \end{aligned} \tag{12}$$

Then, starting from the state $\mathbf{v}(0) = (0, 1, 1)$, the orbit $\langle I^k \rangle_{k \in \omega}$ of $I^0$ with respect to $T_{\pi(N_1)}$ is $\{q, r\}$, $\{p, r\}$, $\{q\}$, $\{p, r\}$, ..., which is exactly the trajectory of $N_1$ from $\mathbf{v}(0)$. Among them, the repeat $pr \rightarrow q \rightarrow pr$ represents the cycle attractor of $N_1$. On the other hand, starting from $(1, 1, 1)$, the orbit of $\{p, q, r\}$ reaches the fixed point $\{r\}$, which corresponds to the point attractor $r \rightarrow r$ of $N_1$.

We often identify a state $\mathbf{v}(t)$ with an interpretation $I^t$, and will denote it as $I$ whenever the time step $t$ is not important.

**Proposition 4.2.** [14] *Let $N = (V, F)$ be a Boolean network. Then, $\{I\}$ is a point attractor of $N$ iff $I$ is a supported model of $\pi(N)$.*

Proposition 4.2 characterizes the set of all point attractors of a Boolean networks in terms of a single logic program. All supported models of $\pi(N)$ can be enumerated as the models of its completion:

$$Comp(\pi(N)) \equiv \bigwedge_{v_i \in (V \setminus V_C)} \left( v_i \leftrightarrow \bigvee_{j=1}^{l_i} B_{i,j} \right). \tag{13}$$

Using a SAT solver, computation of point attractors can be automated by computing the models of the formula (13).

We now generalize Proposition 4.2 to characterize the set of all attractors of a Boolean network.

**Theorem 4.3.** *Let $N = (V, F)$ be a Boolean network. Then, $S$ is an attractor of $N$ iff $S$ is a supported class of $\pi(N)$.*

*Proof.* $S$ is an attractor of $N$ iff $R_N(I) = S$ for every $I \in S$ (by definition) iff $\mathcal{T}_{\pi(N)}(I) = S$ for every $I \in S$ (by Proposition 4.1) iff $S$ is a supported class of $\pi(N)$ (by Theorem 3.1). □

Theorem 4.3 now characterizes both cycle and point attractors of Boolean networks. In fact, Proposition 4.2 is derived as a corollary of Theorem 4.3 using Theorem 3.4. For Example 4.2, we see that there are two supported classes of $\pi(N_1)$, $\{\{r\}\}$ and $\{\{p, r\}, \{q\}\}$, which respectively correspond to the point attractor and the cycle attractor of $N_1$. Remember that $\{r\}$ is the unique supported model of $\pi(N_1)$, which represents the point attractor by Proposition 4.2.

### 4.2 From Logic Programs to Boolean Networks

Here we show a converse translation from logic programs to Boolean networks. We here assume that the Herbrand base $\mathcal{B}$ is finite. By setting $\mathcal{B}$ as the nodes of a Boolean network, any interpretation $I \in 2^{\mathcal{B}}$ can be identified with a state at some time step $t$, that is, $A_i(t) = 1$ iff $A_i \in I$ and $A_i(t) = 0$ iff $A_i \notin I$ for any $A_i \in \mathcal{B}$. According to [14], given a propositional logic program $P$, we can construct the Boolean network $\nu(P) = (\mathcal{B}, F(P))$, where $F(P)$ is defined as follows. For each $A_i \in \mathcal{B}$, suppose the set of rules in $P$ whose heads are $A_i$ is precisely given by $\{(A_i \leftarrow \Gamma_{i,1}), \ldots, (A_i \leftarrow \Gamma_{i,i_k})\}$, where each $\Gamma_{i,j}$ $(1 \leq j \leq i_k)$ is a conjunction of literals. The Boolean function for $A_i$ is then defined as $A_i(t+1) = (\Gamma_{i,1}(t) \vee \cdots \vee \Gamma_{i,i_k}(t))$ if $i_k \geq 1$, and is assigned 0 if $i_k = 0$ ($A_i$ is a 0-*node*[12] ).

**Proposition 4.4.** [14] *Let $P$ be a propositional logic program. Then, $I$ is a supported model of $P$ iff $\{I\}$ is a point attractor of $\nu(P)$.*

Here, we can generalize Proposition 4.4 as follows.

**Theorem 4.5.** *Let $P$ be a propositional logic program. Then, $S$ is a supported class of $P$ iff $S$ is an attractor of $\nu(P)$.*

*Proof.* Each Boolean function $f_i \in F(P)$ for a node $A_i \in \mathcal{B}$ in $\nu(P) = (\mathcal{B}, F(P))$ can be written as $f_i = (\Gamma_{i,1} \vee \cdots \vee \Gamma_{i,i_k})$. Then, for a time step $t$, $A_i(t+1) = 1$ iff $f_i(t) = 1$ iff $(\exists j)\Gamma_{i,j}(t) = 1$. Suppose an interpretation $I \in 2^{\mathcal{B}}$ such that $(A_i(t) = 1$ iff $A_i \in I)$ and $(A_i(t) = 0$ iff $A_i \notin I)$ hold for any $A_i \in \mathcal{B}$. Then, $J = T_P(I)$ iff $I$ for each $A_i \in J$ there exists a rule $(A_i \leftarrow \Gamma_{i,j})$ in $P$ such that $\Gamma_{i,j}$ is true in $I$ iff $I$ satisfies $f_i$ for every $A_i \in J$ iff $J$ is the next state of $I$ for the Boolean network $\nu(P)$. Then, $S$ is a supported class of $P$ iff $\mathcal{T}_P(I) = S$ for any $I \in S$ (by Theorem 3.1) iff $R_{\nu(P)}(I) = S$ for any $I \in S$ (by Proposition 4.1) iff $S$ is an attractor of $\nu(P)$. □

Note that, given a Boolean network $N$, it may hold that $\nu(\pi(N)) \neq N$. This is because for any constant node $v_i$ in $N$, no Boolean function is defined for $v_i$ in $N$. However, the rule $(v_i \leftarrow v_i)$ is included in $\pi(N)$, which is then kept in $\nu(\pi(N))$ as the Boolean function $f_i{}' = v_i$. Nevertheless, we can identify a Boolean network $N$ and its corresponding logic program $\pi(N)$ and identify a logic program $P$ and its corresponding Boolean network $\nu(P)$ under the supported class semantics.

## 5 Between Stable and Supported Classes

We have seen in Section 2 that the stable class semantics can represent several important semantics of logic programs including the stable model semantics [12] and the well-founded semantics [27].

[12] In a Boolean network, a 0-node can be graphically illustrated as a node that has a single incoming edge from $\bot$ representing the value 0.

On the other hand, the supported class semantics defined in Section 3 is suitable to represent dynamics of state transition systems including Boolean networks [19] as shown in Section 4. Then, another merit of the supported class semantics is that computation of supported classes is possible via algorithms to compute attractors of Boolean networks. For example, attractors are searched on state transition diagrams for Boolean networks with bounded model checking [8], in which the length of trajectories is incrementally varied. Particular focuses are put on point attractors [25,2] based on elaborate translations of Boolean networks into SAT through completion (13). There are also some efficient algorithms to compute cycle attractors with period 2 when Boolean networks are unate [1]. These procedures to compute attractors of Boolean networks can thus be applied to compute supported classes of logic programs. On the other hand, there has been no procedure to compute stable classes except that singleton stable classes, i.e., stable models, can be computed in ASP.

Given those merits of each semantics, we now investigate relationships between stable and supported classes. At first, recall that any stable model of a logic program $P$ is a supported model of $P$ [21]. Any stable model also comprises a singleton stable class [5], and any supported model forms a singleton supported class (Proposition 3.4). Hence, we have the following immediate property.

**Proposition 5.1.** *Let $P$ be a logic program, $I$ a stable model of $P$. Then, $\{I\}$ is both a stable class and a supported class of $P$.*

Next we would expect that this result could be generalized so that any stable class of $P$ is a supported class of $P$. However, as shown in Examples 2.1 and 3.1, this property does not hold. Then, it appears that a supported class might have some delay compared with a corresponding stable class, due to the difference between the $T_P$ and $F_P$ operators applied to reducts, as explained in Example 3.2. Unfortunately, this does not hold either.

**Example 5.1.** Consider the program $P_2$ in Example 2.2, which has two stable classes $S_3$ and $S_4$. The supported classes of $P_2$ are exactly the same as these, corresponding to cycle attractors with period 2 and period 6, respectively.[13] Now, consider the next program $P_5$ obtained from $P_2$ by adding one rule ($p \leftarrow p$):

$$
\begin{aligned}
&p \leftarrow p,\\
&p \leftarrow \neg q,\\
&q \leftarrow \neg r,\\
&r \leftarrow \neg p.
\end{aligned}
$$

$P_5$ has the same stable classes as $P_2$, i.e., $S_3$ and $S_4$, but there is the unique supported class of $P_5$: $S_6 = \{\{p, q\}\}$. In fact, $P_5$ has the unique supported model $\{p, q\}$, and has no cycle attractor in the corresponding Boolean network.

[13] The Boolean network corresponding to $P_2$ is made of three repressors hooked up in a cycle to form a negative feedback loop, and is called a *repressilator* [10]. Such circuits often exhibit very complex oscillations.

The next proposition shows that stable classes and supported classes coincide for logic programs without "prerequisites". A rule is called *prerequisite-free* if $b^+(R) = \emptyset$, and a logic program $P$ is *prerequisite-free* if every rule in $P$ is prerequisite-free.

**Proposition 5.2.** *Let $P$ be a prerequisite-free logic program, and $\mathcal{S}$ a set of interpretations. Then, $\mathcal{S}$ is a stable class of $P$ iff $\mathcal{S}$ is a supported class of $P$.*

*Proof.* Let $I$ be any interpretation. When $P$ is prerequisite-free, it holds that $P^I = \{h(R) \mid R \in ground(P),\ b^-(R) \cap I = \emptyset\}$. Hence, $F_P(I) = T_{P^I} \uparrow \omega = T_{P^I}(I)$. By [21, Theorem 2], $T_{P^I}(I) = T_P(I)$. Therefore, $F_P(I) = T_P(I)$. Hence, for any $I \in \mathcal{S}$, $\mathcal{F}_P(I) = \mathcal{T}_P(I)$ holds. By Theorems 2.1 and 3.1, $\mathcal{S}$ is a stable class of $P$ iff $\mathcal{S}$ is a supported class of $P$. □

Proposition 5.2 gives a bridge between stable classes and supported classes. If we find a translation of a program $P$ into a prerequisite-free program $Q$ without changing the stable classes of $P$, then we can compute supported classes of $Q$ that are equivalent to stable classes of $Q$ (and $P$) by Proposition 5.2. Fortunately, we can use *unfolding* for this purpose.

Let $R$ be a ground rule of the form (1):

$$A \ \leftarrow \ A_1 \wedge \cdots \wedge A_m \wedge \neg A_{m+1} \wedge \cdots \wedge \neg A_n.$$

Suppose that $R_i$ is a ground rule of the form $A_i \leftarrow \Gamma_i$ for $i = 1, \ldots, m$, where $\Gamma_i$ is a conjunction of literals. Then, $U_R(R_1, \ldots, R_m)$ is a ground rule:

$$A \leftarrow \Gamma_1 \wedge \cdots \wedge \Gamma_m \wedge \neg A_{m+1} \wedge \cdots \wedge \neg A_n.$$

Given an NLP $P$ and a prerequisite-free program $Q$, Aravindan and Dung [4] have defined unfolding of the rules in $ground(P)$ with prerequisite-free rules in $Q$ via the following operator $S_P$:

$$S_P(Q) = \{U_R(R_1, \ldots, R_m) \mid R \in ground(P),\ R_i \in Q,\ 1 \leq i \leq m\}.$$

Note that any $U_R(R_1, \ldots, R_m) \in S_P(Q)$ is a prerequisite-free rule, and that $S_P(\emptyset) = \{R \in ground(P) \mid b^+(R) = \emptyset\}$ is the set of prerequisite-free rules in $ground(P)$. Then the *semantic kernel* of $P$, denoted as $SK(P)$, is defined as the prerequisite-free program $S_P \uparrow \omega$, where

$$\begin{array}{ll} S_P \uparrow 0 & = \emptyset, \\ S_P \uparrow n+1 & = S_P(S_P \uparrow n), \\ S_P \uparrow \omega & = \bigcup_{n \in \omega} S_P \uparrow n. \end{array}$$

**Proposition 5.3.** [4, Theorem 3.1] *Let $P$ be an NLP, and $SK(P)$ its semantic kernel. Then, $P$ and $SK(P)$ have the same stable models.*

Proposition 5.3 can be generalized as follows. In [4], it has been shown that $P$ and $SK(P)$ have the same preferred extensions, the same regular models, and the same partial stable models. Hence, we add one more property here.

**Lemma 5.4.** *Let $P$ be an NLP, and $SK(P)$ its semantic kernel. For any atom $A \in \mathcal{B}$, $A \in T_P \uparrow n$ for some $n \in \omega$ iff there is a prerequisite-free rule $Q \in S_P \uparrow n$ such that $h(Q) = A$ and $I \cap b^-(Q) = \emptyset$.*

*Proof.* We prove the lemma by induction on $n$. When $n = 0$, it is obvious. Assume that the lemma holds for $n \leq k$, and consider the case of $n = k + 1$. Suppose that $A$ is derived at the ordinal $k+1$, i.e., $A \in T_{P^I} \uparrow k+1$. Then, there is a rule $R \in \mathit{ground}(P)$ such that $h(R) = A$ and $b^-(R) \cap I = \emptyset$. For this $R$, the reduct $R'$ exists in $P^I$ such that $h(R') = h(R) = A$, $b^+(R') = b^+(R)$ and $b^-(R') = \emptyset$. Moreover, $A_i \in T_P \uparrow k$ holds for every $A_i \in b^+(R')$. By the induction hypothesis, there is a rule $Q_i \in S_P \uparrow k$ such that $h(Q_i) = A_i$, $b^+(Q_i) = \emptyset$ and $I \cap b^-(Q_i) = \emptyset$. Then, by unfolding, the rule $Q = U_R(Q_1, \ldots, Q_m) \in S_P \uparrow k+1$ such that $h(Q) = A$, $b^+(Q) = \emptyset$ and $I \cap b^-(Q) = I \cap (b^-(Q_1) \cup \cdots \cup b^-(Q_m) \cup b^-(R)) = (I \cap b^-(Q_1)) \cup \cdots \cup (I \cap b^-(Q_m)) \cup (I \cap b^-(R)) = \emptyset$. □

**Theorem 5.5.** *Let $P$ be an NLP, and $SK(P)$ its semantic kernel. Then, $P$ and $SK(P)$ have the same stable classes.*

*Proof.* We prove that $F_P(I) = F_{SK(P)}(I)$ holds for any interpretation $I$. Since $SK(P)$ is prerequisite-free, $SK(P)^I = \{h(R) \mid R \in SK(P), b^-(R) \cap I = \emptyset\}$. Then, $F_{SK(P)}(I) = T_{SK(P)^I} \uparrow \omega = SK(P)^I$.

Now, let $A$ be any ground atom. Then, $A \in F_P(I)$
iff $A \in T_{P^I} \uparrow \omega$ iff $A \in T_{P^I} \uparrow n$ for some $n \geq 1$
iff there is a prerequisite-free rule $Q \in S_P \uparrow n$ such that $h(Q) = A$ and $I \cap b^-(Q) = \emptyset$ (by Lemma 5.4)
iff there is $Q \in SK(P)$ such that $h(Q) = A$ and $I \cap b^-(Q) = \emptyset$
iff $A \in SK(P)^I$ iff $A \in F_{SK(P)}(I)$. □

Hence, computing stable classes is now possible through translation of a logic program $P$ into $SK(P)$ (Theorem 5.5) and computing supported classes of $SK(P)$ (Proposition 5.2) through attractor computation (Theorem 4.5). On the other hand, $SK(P)$ does not preserve the supported models as well as the supported classes of $P$, e.g., $P = \{p \leftarrow p\}$ and $SK(P) = \emptyset$.

As for computational complexity, however, we see below that stable classes and supported classes have the same complexity on their existence. As explained in previous sections, existence of stable/supported classes is always guaranteed.

**Proposition 5.6.** *Any logic program has at least one stable class as well as one supported class.*

The next property is easily obtained by complexity results for existence of stable/supported models of logic programs.

**Proposition 5.7.** (1) *Deciding if there is a singleton stable class is NP-complete.* (2) *Deciding if there is a singleton supported class is NP-complete.*

*Proof.* (1) is obtained by [22, Theorem 6.7]. (2) is equivalent to the problem that $\mathit{Comp}(P)$ is consistent for an NLP $P$, which is NP-complete. □

Our interest here is a program that has no stable/supported model. Such a program is regarded as inconsistent in general, but has a meaning under the stable/supported class semantics. A program $P$ is called *$F_P$-periodic* (resp. *$T_P$-periodic*) if the size of minimum stable (resp. supported) classes of $P$ is more than 1. Then, the next property holds by Propositions 5.6 and 5.7.

**Theorem 5.8.** *Deciding if a logic program $P$ is $F_P$-periodic (or $T_P$-periodic) is coNP-complete.*

As the final remark, notice that the difference between the definition of stable classes and that of supported classes only lies in the deterministic operators, $F_P$ and $T_P$, on $2^{\mathcal{B}}$. Hence, the essence in the proofs of Theorems 2.1 and 2.2 can be directly applied to the proofs of Theorems 3.1 and 3.2, respectively. Both the $F_P$ operator and $T_P$ operator are reasonable as deterministic operators that map over Herbrand interpretations, and are useful to characterize semantic and dynamic aspects of logic programs. It is open whether any other interesting deterministic operator exists for normal logic programs.

For the class of *disjunctive logic programs*, we might need a *nondeterministic* operator like [15,20], which is a mapping $2^{\mathcal{B}} \to 2^{2^{\mathcal{B}}}$, yet Kalinski [17] has defined stable classes for disjunctive programs by extending the definition (6) with the notion of *model filters*. Moreover, to characterize *asynchronous Boolean networks*, the use of nondeterministic operators has also been suggested in [14]. These generalizations are future works.

## References


1. Akutsu, T., Melkman, A.A., Tamura, T.: Singleton and 2-periodic attractors of sign-definite Boolean networks. Information Processing Letters 112, 35–38 (2012)
2. Akutsu, T., Melkman, A.A., Tamura, T., Yamamoto, M.: Determining a singleton attractor of a Boolean network with nested canalyzing functions. Journal of Computational Biology 18, 1275–1290 (2011)
3. Apt, K.R., Blair, H.A., Walker, A.: Towards a theory of declarative knowledge. In: Minker, J. (ed.) Foundations of Deductive Databases and Logic Programming, pp. 89–148. Morgan Kaufmann (1988)
4. Aravindan, C., Dung, P.M.: On the correctness of unfold/fold transformation of normal and extended logic programs. Journal of Logic Programming 24, 201–217 (1995)
5. Baral, C., Subrahmanian, V.S.: Stable and extension class theory for logic programs and default logics. Journal of Automated Reasoning 8, 345–366 (1992)
6. Baral, C., Subrahmanian, V.S.: Dualities between alternative semantics for logic programming and nonmonotonic reasoning. Journal of Automated Reasoning 10, 399–420 (1993)
7. Blair, H.A., Dushin, F., Humenn, P.R.: Simulations between Programs as Cellular Automata. In: Fuhrbach, U., Dix, J., Nerode, A. (eds.) LPNMR 1997. LNCS (LNAI), vol. 1265, pp. 115–131. Springer, Heidelberg (1997)
8. Dubrova, E., Teslenko, M.: A SAT-based algorithm for finding attractors in synchronous Boolean networks. IEEE/ACM Transactions on Computational Biology and Bioinformatics 8(5), 1393–1399 (2011)

9. Dung, P.M.: Negations as hypotheses: An abductive foundation for logic programming. In: Proceedings of ICLP 1991, pp. 3–17. MIT Press (1991)
10. Elowitz, M.B., Leibler, S.: A synthetic oscillatory network of transcriptional regulators. Nature 403(6767), 335–338 (2000)
11. Garg, A., Di Cara, A., Xenarios, I., Mendoza, L., De Micheli, G.: Synchronous versus asynchronous modeling of gene regulatory networks. Bioinformatics 24(17), 1917–1925 (2008)
12. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Proceedings of ICLP 1988, pp. 1070–1080. MIT Press (1988)
13. Gunter, C.A., Scott, D.S.: Semantic domains. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science, vol. B, pp. 633–674. North-Holland (1990)
14. Inoue, K.: Logic programming for Boolean networks. In: Proceedings of IJCAI 2011, pp. 924–930 (2011)
15. Inoue, K., Sakama, C.: A fixpoint characterization of abductive logic programs. Journal of Logic Programming 27(2), 107–136 (1996)
16. Kakas, A.C., Mancarella, P.: Preferred extensions are partial stable models. Journal of Logic Programming 14, 341–348 (1992)
17. Kalinski, J.: Stable Classes and Operator Pairs for Disjunctive Programs. In: Marek, V.W., Truszczyński, M., Nerode, A. (eds.) LPNMR 1995. LNCS (LNAI), vol. 928, pp. 358–371. Springer, Heidelberg (1995)
18. Kauffman, S.A.: Metabolic stability and epigenesis in randomly constructed genetic nets. Journal of Theoretical Biology 22(3), 437–467 (1969)
19. Kauffman, S.A.: The Origins of Order: Self-Organization and Selection in Evolution. Oxford University Press (1993)
20. Marek, V.W., Niemelä, I., Truszczyński, M.: Logic programs with monotone abstract constraint atoms. Theory and Practice of Logic Programming 8(2), 167–199 (2007)
21. Marek, W., Subrahmanian, V.S.: The relationship between stable, supported, default and autoepistemic semantics for general logic programs. Theoretical Computer Science 103(2), 365–386 (1992)
22. Marek, W., Truszczyński, M.: Autoepistemic logic. Journal of the ACM 38(3), 588–619 (1991)
23. Saccà, D., Zaniolo, C.: Stable models and non-determinism in logic programs with negation. In: Proceedings of the 9th ACM SIGMOD-SIGACT Symposium on Principles of Database Systems, pp. 205–217 (1990)
24. Shmulevich, I., Dougherty, E.R., Zhang, W.: From Boolean to probabilistic Boolean networks as models of genetic regulatory networks. Proceedings of the IEEE 90(11), 1778–1792 (2002)
25. Tamura, T., Akutsu, T.: Detecting a singleton attractor in a Boolean network utilizing SAT algorithms. IEICE Trans. 92-A(s), 493–501 (2009)
26. van Emden, M.H., Kowalski, R.A.: The semantics of predicate logic as a programming language. Journal of the ACM 23(4), 733–742 (1976)
27. Van Gelder, A., Ross, K., Schlipf, J.S.: The well-founded semantics for general logic programs. Journal of the ACM 38(3), 620–650 (1991)
28. Wolfman, S.: Cellular Automata And Complexity: Collected Papers. Westview Press (1994)
29. You, J.H., Yuan, L.: A three-valued semantics for deductive database and logic programs. Journal of Computer and System Sciences 49, 334–361 (1994)
30. You, J.H., Yuan, L.: On the equivalence of semantics for normal logic programs. Journal of Logic Programming 22(3), 212–222 (1995)