From 3-valued Semantics to Supported Model Computation for Logic Programs in Vector Spaces

Taisuke Sato\textsuperscript{1}*, Chiaki Sakama\textsuperscript{2} and Katsumi Inoue\textsuperscript{3}

\textsuperscript{1}National Institute of Advanced Industrial Science and Technology, Japan
\textsuperscript{2}Wakayama University, Japan
\textsuperscript{3}National Institute of Informatics, Japan

\texttt{satou.taisuke@aist.go.jp, sakama@wakayama-u.ac.jp, inoue@nii.ac.jp}

Keywords: Dualized Logic Program, Supported Model, 3-valued, Vector Space.

Abstract: We propose a linear algebraic approach to computing 2-valued and 3-valued completion semantics of finite propositional normal logic programs in vector spaces. We first consider 3-valued completion semantics and construct the least 3-valued model of \text{comp}(DB), i.e. the iff (if-and-only-if) completion of a propositional normal logic program DB in Kleene’s 3-valued logic which has three truth values \{t(true), f(false), \bot(undef){\text{indef}}\}. The construction is carried out in a vector space by matrix operations applied to the matricized version of a dualized logic program DB\text{d} of DB. DB\text{d} is a definite clause program compiled from DB and used to compute the success set $P_\infty$ as true atoms and finite failure set $N_\infty$ as false atoms. They constitute the least 3-valued model $I_\infty = (P_\infty, N_\infty)$ of \text{comp}(DB). We then construct a supported model of DB, i.e. a 2-valued model of \text{comp}(DB) by carefully assigning t or f to undefined atoms not in $P_\infty$ or $N_\infty$ so that the resulting model is 2-valued and supported. The assigning process is guided by an atom dependency relation on undefined atoms. We implemented our proposal by matrix operations and conducted an experiment with random normal logic programs which demonstrated the effectiveness of our linear algebraic approach to computing completion semantics.

1 INTRODUCTION

Performing logical inference in vector spaces has been studied as an attractive alternative to the traditional symbolic inference which opens a way to take advantage of flexible matrix operations in vector spaces and associated parallelism supported by recent computer technologies (Sato, 2017; Sakama et al., 2017; Sato et al., 2018). For example it was demonstrated Datalog programs can be computed orders of magnitude faster in vector spaces than by symbolic computation when relations are not too sparse (Sato, 2017). Also it is possible to invent new relations for more than $10^4$ entities from real knowledge graphs by reformulating abduction in vector spaces (Sato et al., 2018). In this paper, along the same line, we develop a linear algebraic approach to computing completion semantics of logic programs in vector spaces.

We first consider 3-valued completion semantics and construct the least 3-valued model of \text{comp}(DB) where \text{comp}(DB) is the iff (if-and-only-if) completion of a finite propositional normal logic program DB as the 3-valued denotation of DB, in Kleene’s 3-valued logic which is based on three truth values \{t(true), f(false), \bot(undef){\text{indef}}\} (Fitting, 1985; Kunen, 1987). We prove that the model construction can be deterministically carried out in a vector space by matrix operations applied to the matricized version of a dualized logic program DB\text{d} of DB. DB\text{d} is a definite finite clause program compiled from DB and used to compute DB’s success set $P_\infty$ as true atoms, and DB’s finite failure set $N_\infty$ as false atoms. They constitute the least 3-valued model $I_\infty = (P_\infty, N_\infty)$ of \text{comp}(DB). We then construct a 2-valued model of \text{comp}(DB), i.e. supported model (Apt et al., 1988; Marek and V.S.Subrahmanian, 1992) of DB by appropriately assigning t or f to undefined atoms not in $P_\infty$ or $N_\infty$ so that the resulting model is 2-valued and supported. Every supported model of DB is obtained this way. The assignment process is guided by an atom dependency relation on undefined atoms and processes them from bottom-layers. Since every true (resp.
false) atom in \( I \) remains true (resp. false) in any supported model of \( DB \), we may say that \( I \) represents the "deterministic part" of supported models (or stable models) of \( DB \). To our knowledge, our approach is the first that computes 3-valued completion semantics in vector spaces and also the first that computes the deterministic part of supported models separately.

2 PRELIMINARIES

In this paper, programs mean finite propositional normal logic programs \( DB \) made up of a set of atoms \( A \) unless otherwise stated. Write \( DB = \{ a_i \leftarrow B_i \mid 1 \leq i \leq N, a_i \in A \} \) where \( B_i \) is a conjunction of literals whose atoms are in \( A \). Let \( a \leftarrow B_1, \ldots, a \leftarrow B_k \) be clauses about \( a \) in \( DB \). Introduce new constant atoms \( \text{true} \) and \( \text{false} \) respectively denoting \( t \) and \( f \). We define the iff (if-and-only-if) completion about \( a \) as follows.

If \( k = 0 \), i.e. there is no clause about \( a \), put \( \text{iff}(a) = a \iff \text{false} \). If \( k = 1 \) and \( B_1 \) is an empty conjunction, i.e. \( a \) is a unit clause, put \( \text{iff}(a) = a \iff \text{true} \). Otherwise put \( \text{iff}(a) = a \iff B_1 \lor \cdots \lor B_k \). The completion of \( DB \), denoted by comp\((DB)\), is defined as comp\((DB) = \{ \text{iff}(a) \mid a \in A \} \).

We consider 3-valued completion semantics of logic programs following Fitting and Kunen(Fitting, 1985; Kunen, 1987). Their semantics is based on Kleene’s 3-valued logic and interpret \( \iff \) in a 2-valued way. We stipulate that \( \llbracket A \iff B \rrbracket = t \) if \( \llbracket A \rrbracket = \llbracket B \rrbracket = t \).

It is known that Fitting’s 3-valued semantics of normal logic programs is highly undecidable whereas Kunen’s 3-valued one is recursively enumerable (Fitting, 1985; Kunen, 1987), but they coincide in the case of finite propositional normal logic programs, which we describe next. Here we inductively define a series of 3-valued interpretations \( I_0 = (P_0, N_0), I_1 = (P_1, N_1), \ldots \) (Sato, 1990).

(base case) \( P_0 = N_0 = \emptyset \)

[inductive step]

Put \( I_{n+1} = (P_{n+1}, N_{n+1}) \)

\( P_n = \{ a \in A \mid \text{there is a } B \in DB \text{ s.t. } \llbracket B \rrbracket_{I_n} = t \} \)

\( N_n = \{ a \in A \mid \text{for all } a \not\in B \text{ in } DB, \llbracket B \rrbracket_{I_n} = f \} \)

Figure 1: Defining 3-valued interpretations \( \{ (P_n, N_n) \} \).

It is straightforward to verify \( P_n \cap N_n = \emptyset (n=0,1,\ldots) \), and hence \( I_n = (P_n, N_n) \) is a 3-valued interpretation. Let \( (A_1, A_2) \) and \( (B_1, B_2) \) be pairs of sets. We introduce a partial ordering "\( \subseteq \)" on pairs of sets by \( (A_1, A_2) \subseteq (B_1, B_2) \iff A_1 \subseteq B_1 \) and \( A_2 \subseteq B_2 \). Now we see \( I_0 \subseteq I_1 \subseteq \cdots \) (Sato, 1990). We introduce \( P_\infty \) by \( P_\infty = \bigcup P_n \) and \( N_\infty \) by \( N_\infty = \bigcup N_n \) respectively and put \( I_\infty = (P_\infty, N_\infty) \).

We list some propositions (see (Sato, 1990) for proofs).

**Proposition 1.** \( I_\infty \) is a 3-valued model of comp\((DB)\).

**Proposition 2.** \( I_\infty \) is the least 3-valued model of comp\((DB)\) in the sense of \( \subseteq \)-ordering.

It is apparent that when \( DB \) is a definite clause program, \( P_\infty \) in \( I_\infty = (P_\infty, N_\infty) \) gives the success set of \( DB \), i.e. the least 2-valued model of \( DB \) (whereas \( N_\infty \) gives the finite failure set of \( DB \)). We consider the least 3-valued model \( I_\infty \) as the denotation of \( DB \) in our 3-valued semantics.

A 2-valued completion model of \( DB \), i.e. a 2-valued interpretation satisfying comp\((DB)\), is called a supported model of \( DB \) (Apt et al., 1988; Marek and V.S.Subrahmanian, 1992). DB may have no supported model (think of a \( \leftarrow \neg a \) but if \( DB \) has a supported model represented by a set \( P_i (\subseteq A) \), we see \( I_\infty \subseteq I_i = (P_i, A \setminus P_i) \) thanks to Proposition 2 because every 2-valued completion model is also a 3-valued completion model. In other words, every sup-

\(^1\)Throughout this paper, we assume that if a program has a unit clause \( a \leftarrow \), there is no other clause about \( a \) in the program.

\(^2\)Likewise, a 2-valued model of comp\((DB)\) is called a 2-valued completion model of DB.
ported model is obtained by appropriately assigning \( t \) or \( f \) to the undefined atoms in \( U_\infty = A \setminus (P_m \cup N_m) \). Supported models are a super class of stable models and when DB is \( \textit{tight} \) (no looping caller-callee chain through positive goals), DB’s supported models and DB’s stable models coincide (Erdem and Lifschitz, 2003).

\[
DB_0 = \begin{cases} 
  a \leftarrow \neg b \land c \\
  b \leftarrow \neg a \land c \\
  c \leftarrow \neg d \\
  a \leftarrow \neg b \land c \\
  b \leftarrow \neg a \land c \\
  c \leftarrow \neg d \\
  d \leftarrow \text{false}
\end{cases}
\]

\[
\text{comp}(DB_0) = \begin{cases} 
  a \leftarrow \neg b \land c \\
  b \leftarrow \neg a \land c \\
  c \leftarrow \neg d \\
  a \leftarrow \neg b \land c \\
  b \leftarrow \neg a \land c \\
  c \leftarrow \neg d \\
  d \leftarrow \text{false}
\end{cases}
\]

Figure 2: A program \( DB_0 \) and its completion \( \text{comp}(DB_0) \).

Look at \( DB_0 \) in Figure 2. We see \( (P_0, N_0) = (\emptyset, \emptyset) \), \( (P_1, N_1) = (\emptyset, \{d\}) \), \( (P_2, N_2) = (\{e\}, \{d\}) \) and \( (P_3, N_3) = (P_2, N_2) = \emptyset. \) Then \( \{a, b\} \) are undefined atoms \( U_\infty \) in \( DB_0 \). So any supported model of \( DB_0 \) is obtained by assigning \( t \) or \( f \) to each of \( a \) and \( b \) appropriately. Actually \( \{\{a, c\}, \{b, d\}\} \) and \( \{\{b, c\}, \{a, d\}\} \) exhaust all supported models of \( DB_0 \).

3 MATRICIZING 3-VALUED SEMANTICS

We here compute \( I_m = (P_m, N_m) \) in a vector space by matrix. As a preprocessing step, we standardize a program first and then represent the standardized program by a 0-1 matrix.

3.1 Standardization

Let DB be a program, \( A \) the set of all atoms appearing in DB. For every atom \( a \in A \), do the following.

- If \( a \) has no clause about it, add \( a \leftarrow \text{false} \) to DB.
- If \( a \) has a unit clause \( a \leftarrow \), replace it with \( a \leftarrow \text{true} \).
- If there is more than one clause \( \{ a \leftarrow B_1, \ldots, a \leftarrow B_k \} \) \((k > 1)\) in DB, replace them with a set of new clauses \( \{ a \leftarrow b_1 \lor \cdots \lor b_k, b_1 \leftarrow B_1, \ldots, b_k \leftarrow B_k \} \) containing new atoms \( \{b_1, \ldots, b_k\} \).

Call the resulting program a \( \text{standardized program} \) of DB. In general, if a program is standardized, every atom \( a \in A \) has exactly one clause \( a \leftarrow B \) about it and \( B \) is \( \text{true}, \text{false} \), a conjunction or disjunction of literals in \( A \).

\[\text{Proposition 3. Let} \ DB^d \ \text{be a standardized program of} \ DB. \ \text{Then} \ \text{comp}(DB^d) \ \text{and} \ \text{comp}(DB) \ \text{have the same} \ 3\text{-valued completion models as far as atoms in} \ A \ \text{are concerned (proof omitted).} \]

Proposition 3 tells us that to compute a 3-valued model of \( \text{comp}(DB) \), we may assume DB is standardized. So, hereafter, we only deal with standardized programs.

3.2 Dualized Logic Programs

Here we introduce dualized programs for later use. Let \( DB = \{a_i \leftarrow B_i \mid 1 \leq i \leq N\} \) be a standardized program in a set of atoms \( A = \{a_1, \ldots, a_N\} \). We define a definite clause program \( DB^d \) called a \( \textit{dualized program} \) of DB that can compute DB’s least 3-valued completion model \( I_m = (P_m, N_m) \) as its least 2-valued model.

First introduce a set of new atoms \( \bar{A} = \{n_a, \ldots, n_{a_N}\} \). The idea is that \( n_a \) represents \( \neg a \) by negation-as-failure (NAF), i.e. an SLD derivation with NAF for \( a \). For a finitely fails. We now define a syntactic function \( p(\cdot) \) as follows. First put \( p(\text{true}) = \text{false} \) and \( p(\text{false}) = \text{true} \). Next consider literals. Let \( l \) be a literal whose atom in \( A \). If \( l \) is an atom \( a \), define \( p(l) = na \). Otherwise \( l = \neg a \) and put \( p(l) = a \). We extend \( p(\cdot) \) to conjunction and disjunction by defining \( n(l_1 \land \cdots \land l_m) = n(l_1) \lor \cdots \lor n(l_m) \) and \( n(l_1 \lor \cdots \lor l_m) = n(l_1) \land \cdots \land n(l_m) \). Also we apply \( n(\cdot) \) to a set \( S \) by \( n(S) = \{n(a) \mid a \in S\} \).

Dually we introduce a function \( p(\cdot) \) s.t. \( p(\text{true}) = \text{true} \) and \( p(\text{false}) = \text{false} \). For atoms in \( A \), if \( l \) is an atom \( a \), put \( p(l) = a \). Otherwise \( l = \neg a \) and put \( p(l) = na \). \( p(\cdot) \) is extended to conjunction and disjunction by \( p(l_1 \land \cdots \land l_m) = p(l_1) \lor \cdots \lor p(l_m) \) and \( p(l_1 \lor \cdots \lor l_m) = p(l_1) \land \cdots \land p(l_m) \).

Finally define a \( \text{dualized program} \) \( DB^d \) of DB. It is a definite clause program and defined by \( DB^d = \{a \leftarrow p(B), na \leftarrow n(B) \mid a \leftarrow B \in DB\} \). Note that if DB is standardized, so is \( DB^d \).

Look at \( DB_1 \) in Figure 3. It is a standardization of \( DB_0 \) in Figure 2. Notice that the least 2-valued model of \( DB^d_0 \) is \( \{c, nd\} \) and it exactly corresponds to the least 3-valued model \( \{\{c\}, \{d\}\} \) of \( \text{comp}(DB_1) \).

This is not a coincidence. We next prove that \( DB^d \) computes the least 3-valued model \( I_m \) of \( \text{comp}(DB) \) as its least 2-valued model.

Put \( \bar{A} = \{na \mid a \in A\} \) and \( A^d = A \cup \bar{A} \). Equate \( J \subseteq A^d \) with a 2-valued interpretation s.t. \( [c]_J = t \iff c \in J \) for an atom \( e \in A^d \). Now define a series of 2-valued interpretations \( \{J_i \subseteq A^d\} \) for \( DB^d \) by \( J_0 = \emptyset \) and \( J_n = \{c, c \leftarrow D \in DB^d \mid [D]_{J_{n-1}} = t\} \). We have \( J_0 = J_1 \subseteq J_2 \subseteq \cdots \) and \( J_n = \bigcup_{i=0}^{n} J_i \) gives the least 2-valued model of \( DB^d \) as usual. We call \( \{J_n\} \) the
Theorem 1 tells us that the least 3-valued model of $F$ respectively. Then we have $\theta(i) = 1$ if $\theta(i) = 1$.

- If $B_i$ is true, set $Q(i, j) = \theta(i) = 1$.
- If $B_i = false$, set $Q(i + N, j + N) = 0$.
- Otherwise, let $a_i \leftarrow B_i$ and $n a_i \leftarrow n B_i$ be clauses about $a_i$ and $n a_i$ in $DB_d$ respectively.

Let $I_\infty = N \leftarrow nB_i$ be the least 3-valued model of $DB_d$ respectively.

- If $B_i$ is a conjunction with $|B_i| > 1$, set $\theta(i) = |B_i|$ and $\theta(i + N) = 1$.
- Otherwise $B_i$ is a disjunction. Set $\theta(i) = 1$ and $\theta(i + N) = |n B_i|$.

For $p(1 \leq p \leq m)$, if $I_p$ is an atom $a_j$, put $Q(i, j) = Q(i + N, j + N) = 1$.

Else $I_p$ is a negated atom $\neg a_j$ and set $Q(i, j) = Q(i + N, j) = 1$.

3.3 Matricized Logic Programs

Let $DB = \{a_1 \leftarrow B_1, \ldots, a_N \leftarrow B_N\}$ be a standard normalized logic program where the bodies $B_i$s are either true, false, a conjunction or disjunction of atoms in $A = \{a_1, \ldots, a_N\}$. We represent the dualized program as $DB^d = \{a_1 \leftarrow B_1, \ldots, a_N \leftarrow B_N, n a_1 \leftarrow n B_1, \ldots, n a_N \leftarrow n B_N\}$. We represent $DB$ in terms of a $(2N \times 2N)$ 0-1 matrix $Q$ and a $(2N \times 1)$ threshold vector $\theta$. $\theta$ is said to be a matricized $DB^d$ or a program matrix for $DB^d$. We construct $Q$ and $\theta$ by a procedure in Figure 4.

- Initialize $Q$ to a $(2N \times 2N)$ zero matrix.
- For $i(1 \leq i \leq N)$, do the following:
  - If $B_i = true$, set $Q(i, j) = \theta(i) = 1$.
  - If $B_i = false$, set $Q(i + N, i + N) = \theta(i) = 1$.
  - Otherwise, let $a_i \leftarrow B_i$ and $n a_i \leftarrow n B_i$ be clauses about $a_i$ and $n a_i$ in $DB_d$ respectively.

- If $B_i$ is a conjunction with $|B_i| > 1$, set $\theta(i) = |B_i|$ and $\theta(i + N) = 1$.
- Otherwise $B_i$ is a disjunction. Set $\theta(i) = 1$ and $\theta(i + N) = |n B_i|$.

- Write $B_i$ as $l_1 \cdots l_m (m > 0)$ where $\circ$ denotes either $\land$ or $\lor$.

3. DB only stores information about occurrences of atoms in the clause body and does not make a distinction between conjunction and disjunction. We need to record supplementary information $\theta$ to recover the original $DB^d$.

4. For a conjunction or disjunction $F$, we use $|F|$ to denote the number of literals occurring in $F$. In this paper, a definite clause whose body is a conjunction (resp. disjunction) is called a conjunctive clause (resp. disjunctive clause).
We here introduce a thresholding notation $(x)_{x \geq \theta}$ parameterized with a real number $\theta$ by $(x)_{x \geq \theta} = 1$ if $x \geq \theta$. Otherwise $(x)_{x \geq \theta} = 0$. We further extend $(x)_{x \geq \theta}$ to a vector notation $\mathbf{u}_{x \geq \theta}$ with a threshold vector $\theta$ in such a way that $\mathbf{u}_{x \geq \theta}(i) = (\mathbf{u}(i))_{x \geq \theta(i)}$ for $1 \leq i \leq n$, where $\mathbf{u}$ and $\theta$ are $n$ dimensional vectors.

Now consider the $k$-th clause $c_k \leftarrow D_k \in \text{DB}^d$ ($1 \leq k \leq 2N$) and let $Q(k, \cdot)$ be the $k$-th row encoding $D_k$. By construction, $Q(k, \cdot)\mathbf{u}^{(j)}$ gives the number literals in $D_k$ which are true in $J$. Hence when $D_k$ is a conjunction, $D_k$ is true in $J$ iff $Q(k, \cdot)\mathbf{u}^{(j)} = |D_k| (= \theta(k))$, or equivalently $|D_k| = (Q(k, \cdot)\mathbf{u}^{(j)})_{x \geq \theta(k)}$. Similarly, if $D_k$ is a disjunction, we set $\theta(k) = 1$ and have $|D_k| = (Q(k, \cdot)\mathbf{u}^{(j)})_{x \geq \theta(k)}$. Thus in either case, the clause body $D_k$ of $c_k \leftarrow D_k$ is evaluated by $J$ as $|D_k| = (Q(k, \cdot)\mathbf{u}^{(j)})_{x \geq \theta(k)}$, purely in a vector space in terms of matrix multiplication and thresholding. We summarize the argument so far as

**Proposition 4.** Let $\text{DB} = \{a_1 \leftarrow B_1, \ldots, a_n \leftarrow B_N\}$ be a standardized normal logic program in atoms $\mathcal{A} = \{a_1, \ldots, a_N\}$, $\text{DB}^d$ the dualized logic program of $\text{DB}$, $\text{J}_n$ the least 2-valued model of $\text{DB}^d$, $Q$ a program matrix and $\theta$ a threshold vector for $\text{DB}^d$. Compute a 2N dimensional vector $\mathbf{u}_\infty = (Q\mathbf{u})_{x \geq \theta}$. Then $\mathbf{u}_\infty$ satisfies $\mathbf{u}_\infty = (Q\mathbf{u})_{x \geq \theta}$ (proof omitted).

Proposition 4 says that $\mathbf{u}_\infty$ is a fixedpoint vector of $f(\mathbf{u}) = (Q\mathbf{u})_{x \geq \theta}$ but does not tell us how to compute it, in particular, in a solely linear algebraic way.

**Proposition 5.** Let $\{J_n\}$ be the defining interpretations associated with the least model $J_\infty$ of $\text{DB}^d$. Define a series of 0-1 vectors $\{\mathbf{u}_n\}$ by $\mathbf{u}_1 = \mathbf{u}^{(J_1)}$ and $\mathbf{u}_{n+1} = (Q\mathbf{u}_n)_{x \geq \theta}$. Then $\mathbf{u}_n = \mathbf{u}^{(J_n)}$ for all $n \geq 1$ and $\lim_{n \to \infty} \mathbf{u}_n = \mathbf{u}_\infty$ (proof omitted).

We restate Proposition 5 as the least 3-valued model computation procedure in Figure 6. Given a dualized program $\text{DB}^d$ of a standardized normal logic program $DB = \{a_1 \leftarrow B_1, \ldots, a_N \leftarrow B_N\}$, it computes the least 2-valued model $J_\infty$ of $\text{DB}^d$ as a 2N dimensional 0-1 vector $\mathbf{u}_\infty$ s.t. $\mathbf{u}_\infty = \mathbf{u}^{(J_\infty)}$. We use a program matrix $Q$ and a threshold vector $\theta$ for $\text{DB}^d$.

**Step 1:** Compute $\mathbf{u}_1 = \mathbf{u}^{(J_1)}$.

**Step 2:** Iterate for $n = 1, 2, \ldots$

\[ \mathbf{u}_{n+1} = (Q\mathbf{u}_n)_{x \geq \theta} \text{ until } \mathbf{u}_{n+1} = \mathbf{u}_n \]

**Step 3:** Return $\mathbf{u}_\infty = \mathbf{u}_n$.

Figure 6: A program matrix $Q_1$ and a threshold vector $\theta_1$ for $\text{DB}_1^d$.

Figure 6: Computing $\mathbf{u}_\infty$ by a program matrix $Q$ and a threshold vector $\theta$.

Applying the procedure in Figure 6 to $\text{DB}^d_1$, we see $\mathbf{u}_1 = (0 0 0 0 0 0 0 0)^T$, $\mathbf{u}_2 = (Q\mathbf{u}_1)_{x \geq \theta_1} = (0 0 1 0 0 0 0 0)^T$, $\mathbf{u}_3 = (Q\mathbf{u}_2)_{x \geq \theta_1} = (0 0 1 0 0 0 0 1)^T = \mathbf{u}_2$. Hence $\mathbf{u}_\infty = \mathbf{u}_2 = (0 0 1 0 0 0 0 1)^T$ which coincides with $\mathbf{u}^{(J_\infty)}$ encoding $J_\infty = \{c, n, d\}$ for $\text{DB}_1^d$ that corresponds to the least 3-valued model $(\{c\}, \{d\})$ of $\text{comp(DB)}$.

The procedure in Figure 6 shows that it is possible to compute the least model semantics purely within vector spaces, but unfortunately, it takes $O((2N)^3)$ time if implemented naively using matrix multiplication. On the other hand, although we do not explain algorithmic details here, it is possible to compute the least model of Horn clause programs by matrix operations in $O(\text{size(DB)})$ time by an elaborated implementation where $\text{size(DB)}$ is the number of atoms occurring in $\text{DB}$.

## 4 SUPPORTED MODEL COMPUTATION

Suppose a program $DB = \{a_1 \leftarrow B_1, \ldots, a_N \leftarrow B_N\}$ in $\mathcal{A} = \{a_1, \ldots, a_N\}$ is given and the least 3-valued model $J_\infty = (P_\infty, N_\infty)$ of $\text{comp(DB)}$ has been computed. Put $U_\infty = \mathcal{A} \setminus (P_\infty \cup N_\infty)$. Atoms $a$ in $U_\infty$ cause infinite computation, or looping computation when an SLD-derivation is applied to $\leftarrow a$. As stated before, every supported model can be obtained by appropriately assigning $t$ or $f$ to atoms in $U_\infty$. We construct a
supported model of DB by a divide-and-conquer approach. First we analyze $U_\infty$ and detect strongly connected components (SCCs) in $U_\infty$ explained next. We then process SCCs one by one, i.e. assign, from bottom SCCs upwards, $t$ or $f$ to atoms in each SCC to locally construct a supported model of the SCC. When this process is completed without failure, we have a supported model of DB.

In general, atoms in $U_\infty$ have caller-caller dependency specified by DB and we express it as a dependency graph $G_{U_\infty}$ s.t. nodes are atoms in $U_\infty$ and there is a directed edge from a node $a_i$ to a node $a_j$ iff there is a clause $a_i \leftarrow a_j$ in DB and $a_j$ contains $a_i$. An SCC is a set of atoms written as $[a] = \{a\} \cup \{b \in V \mid \text{there exist paths from } a \text{ to } b \text{ and from } b \text{ to } a \text{ in } G_{U_\infty} \}$ for some $a \in U_\infty$. Intuitively atoms in an SCC are those calling one another, directly or indirectly.

Note that SCCs in $G_{U_\infty}$ have a natural partial ordering. Let $[a]$ and $[b]$ be two SCCs. If there is a path from $a$ to $b$ but not from $b$ to $a$, we write $[b] \prec [a]$. "$\prec$" is a partial ordering on SCCs. Using "$\prec$", SCCs are reverse-topologically sorted like $[a_1], [a_2], ..., [a_k]$ s.t. whenever $i \prec j$, $i < j$ holds. We can obtain this reverse-topologically sorted list by Tarjan’s algorithm in $O(|V|+|E|)$ time where $|V|$ is the number of nodes, $|E|$ the number of edges of $G_{U_\infty}$.

Look at DB$_2$ in Figure 7 which is a slight variant of DB$_1$. The least 3-valued model of $\text{comp}(DB_2)$ is $\{(a,b,c,d) \}$ and all atoms are undefined, i.e. $U_\infty = \{a,b,c,d\}$. There are two SCCs, $[a] = [b] = \{a,b\}$ and $[c] = [d] = \{c,d\}$, and $[c] \prec [a]$ holds. They are reverse-topologically sorted into a list $\text{SCC}_{DB_2} = \{[c,d],[a,b]\}$.

\[
\text{DB}_2 = \begin{align*}
  a &\leftarrow \neg b \land c \\
  b &\leftarrow \neg a \land c \\
  c &\leftarrow \neg d \\
  d &\leftarrow c \\
\end{align*}
\]

$\text{SCC}_{DB_2} = \{[c,d],[a,b]\}$

Figure 7: DB$_2$ having two SCCs.

Let $\text{SCC}_{DB} = \{\text{SCC}_1, \text{SCC}_2, ...\}$ be a reverse-topologically sorted list of SCCs in $U_\infty$. We build a supported model of DB by processing those SCCs from left-to-right. Suppose $\text{SCC}_j, j \prec i$ preceding SCC, have been processed and their atoms are already assigned $t$ or $f$. Write $\text{SCC}_i = \{b_1, ... , b_k\}$ and let $\text{DB}^{\text{SCC}}_i = \{b_i \leftarrow W_j \mid 1 \leq j \leq K\}$ be a subprogram of DB about atoms in $\text{SCC}_i$. Since undefined atoms that appear in $\text{DB}^{\text{SCC}}_i$ other than $\text{SCC}_i$ are already assigned $t$ or $f$, we construct a supported model of $\text{DB}^{\text{SCC}}_i$ by appropriately assigning $t$ or $f$ to atoms in $\text{SCC}_i$ when possible.

Our task w.r.t. $\text{SCC}_i$ is to make every $b_j \leftarrow W_j$ $(1 \leq j \leq K)$ in $\text{comp}(\text{SCC}_i)$ true (in 2-valued logic) by determining the truth values of $b_1, ... , b_K$. Although this is easily formulated as a SAT problem and solvable by a SAT solver, we would like to exploit the form of iff completion which is lost in translation to CNF. Suppose $\text{iff}(b) = b \iff c \land \neg d$ is in DB$^{\text{SCC}}_i$ and $\{b,c,d\}$ are all undefined. We have to find their truth values that make $\text{iff}(b)$ true. Basically this is an exhaustive search but since $\text{iff}(b)$ is an equivalence formula, the truth value of the atom $b$, once determined, propagates to the body atoms on the right-hand side. For example, if $t$ is assigned to $b$, $c = t$ and $d = f$ necessarily follow for $\text{iff}(b)$ to be true. Or if $f$ is assigned to $b$, we nondeterministically choose one of $\{c, \neg d\}$ and make the chosen literal false. Similarly for the disjunctive case such as $\text{iff}(b) = b \iff c \lor \neg d$, an assignment $b = f$ propagates to $c = f$ and $d = t$ in the body and so on. In this way, we make use of the iff completion to efficiently propagate truth values from one atom to another in $\text{SCC}_i$.

We conclude this section with our search procedure for supported models in Figure 8. Suppose we are given a program DB in a set of propositional atoms $A$.

\begin{itemize}
  \item **Step 1:** Compute the least 3-valued model $I_\omega = (P_\omega, N_\omega)$ of $\text{comp}(DB)$ via dualized program $DB^d$ using the procedure in Figure 6 and extract undefined atoms $U_\infty = A \setminus (P_\omega \cup N_\omega)$.
  \item **Step 2:** Analyze $U_\infty$ and obtain a reverse-topologically sorted list $\text{SCC}_{DB} = \{\text{SCC}_1, ..., \text{SCC}_M\}$ of SCCs in $U_\infty$.
  \item **Step 3:** For $i = 1 \rightarrow M$, assign appropriately true values $\{t, f\}$ to atoms in $\text{SCC}_i$ so that the total assignment constitutes a supported model of DB, i.e. a 2-valued model of $\text{comp}(DB)$.
\end{itemize}

Figure 8: Search procedure for a supported model of DB via dualized $DB^d$.

5 EXPERIMENT

In this section, we conduct an experiment with supported model computation\(^8\) using the computation procedure described in Figure 8 which is implemented entirely by matrix operations provided by GNU Octave 4.2.\(^9\). Our experiment is intended to

\(^8\)The experiment is conducted on a PC with Intel(R) Core(TM) i7-8650U CPU (max 4GHz), 16 GB memory.
\(^9\)For example, to implement Step 2, we represent a dependency graph $G_{U_\infty}$ by an adjacency matrix and implement Tarjan’s algorithm on it to compute SCCs.
show the effectiveness of 3-valued model computation as a preprocessing step prior to 2-valued model computation.

Computing the least 3-valued model $L_a = (P_a, N_a)$ of $\text{comp}(DB)$ at Step 1 in Figure 8 has the effect of reducing the search space for supported model construction of DB by detecting atoms whose truth values are determined, or common to all supported models of DB. That is, since atoms in $P_a$ (resp. atoms in $N_a$) are true (resp. false) in any supported model of DB, we have only to consider truth value assignment for the remaining atoms, those in $U_a = A \setminus (P_a \cup N_a)$ when searching for a supported model of DB.

We here conduct an experiment to measure the effect of Step 1 on search space reduction. First we introduce newly determined atoms. They are atoms whose truth values are “newly determined” at Step 1. By “newly determined”, we mean those atoms in $(P_a \cup N_a) \setminus (P_1 \cup N_1)$ because atoms in $P_1 \cup N_1$ are unit clauses (facts) or atoms having no clauses about them, and hence their truth values are immediately known. Since newly determined atoms are removed from the search space for supported model construction, we measure the effect of search space reduction by “reduction_rate” defined by

\[ \text{reduction rate} = \frac{\#\text{newly determined}}{\#\text{total atoms}} \]

where #newly determined is the number of newly determined atoms and n is the total number of atoms.

In this experiment, after setting the number of atoms $n = 100$ and a probability $p = 0.03$, we randomly generate a normal logic program $DB_{(100,0.03)}$ in a set of propositional atoms $A = \{a_1, \ldots, a_{100}\}$, compute its least 3-valued completion model and count newly determined atom. When generating clauses in $DB_{(100,0.03)}$, we specifically consider “base atoms” $\{a_1, \ldots, a_{10}\}$, and convert them to facts (unit clauses) $a_1 \leftarrow \ldots \leftarrow a_{10}$ or to tautologies $a_1 \leftarrow a_1, \ldots, a_0 \leftarrow a_{10}$. When base atoms are converted to facts, $DB_{(100,0.03)}$ will have more newly determined atoms than are converted to tautologies. To generate clauses $a \leftarrow B$ for the remaining atoms $\{a_{11}, \ldots, a_{100}\}$, we randomly pick up atoms in $A$ with probability $p$, negate them with probability 0.5 and make a conjunction of resulting literals with probability 0.5 as the clause body $B$. Otherwise use their disjunction as $B$. Consequently, the body $B$ contains average $n \cdot p = 100 \cdot 0.03 = 3$ atoms. We repeat this process 90 times and construct the remaining clauses about $\{a_{11}, \ldots, a_{100}\}$ in $DB_{(100,0.03)}$.

Table 1 depicts the statistics of this experiment (figures are average over 10 trials). There #empty_body denotes the average number of atoms having no clause about them in the randomly generated $DB_{(100,0.03)}$. Their truth values are \texttt{f} in every supported model of $DB_{(100,0.03)}$. #undef_atom is the average number of undefined atoms in the least 3-valued model of $\text{comp}(DB_{(100,0.03)})$. #newly_determined_atom is therefore computed as $100 - (\#\text{undef_atom} + \#\text{empty_body} + 10)$ when $\{a_1, \ldots, a_{10}\}$ are converted to facts. Otherwise it is $100 - (\#\text{undef_atom} + \#\text{empty_body})$.

In the former case (see as_facts column), we observe newly_determined_atom = 83.9 giving reduction_rate = 83.9%. It means, on average, 83.9% of atoms are newly assigned \texttt{t} or \texttt{f} at Step 1 and removed from the search space for supported model construction. In the latter case (see as_tautology column) in which base atoms are converted to tautologies, we have more undefined atoms, resulting in a smaller newly_determined_atom, 45.1 on average, but still 45.1% of atoms get their truth values automatically determined and removed from the search space at Step 1. Although the effect of preprocessing by computing the deterministic part of supported models at Step 1 may vary depending on programs, as far as randomly generated programs in this experiment are concerned, we may say it greatly reduces the search space associated with supported model construction.

### 6 RELATED WORK

3-valued semantics of logic programs has been developed primarily from a theoretical perspective (Fitting, 1985; Kunen, 1987; Van Gelder et al., 1991; Naish, 2006; Barbosa et al., 2019). Fitting generalized the $T_p$ operator associated with 2-valued logic programs $P$ to a 3-valued operator $\Phi_p$. He used Kleene’s 3-valued logic and transfinite induction, which is similar to the inductive definition of $\{\{P_n, N_n\}\}$ given in Figure 1 where $n$ is replaced by ordinals, and established the existence of the least 3-valued model of arbitrary normal logic programs (Fitting, 1985). However his semantics is highly undecidable, goes far beyond computable relations (at the cost of induction up to the Church-Kleene ordinal, i.e. the smallest non-recursive ordinal). Kunen later proposed to cut off Fitting’s induction at 0 and proved using a 3-valued ultra power model construction that the notion of $DB \models_3 \phi$, a sentence $\phi$ being a 3-valued logical consequence.

<table>
<thead>
<tr>
<th>Base atoms</th>
<th>as_facts</th>
<th>as_tautology</th>
</tr>
</thead>
<tbody>
<tr>
<td>${a_1, \ldots, a_{10}}$</td>
<td>3.6</td>
<td>12.4</td>
</tr>
<tr>
<td>#empty_body</td>
<td>3.5</td>
<td>42.5</td>
</tr>
<tr>
<td>reduction_rate</td>
<td>83.9%</td>
<td>45.1%</td>
</tr>
</tbody>
</table>

Table 1: The number of newly determined atoms.
of a program DB, is recursively enumerable (Kunen, 1987). Van Gelder proposed well-founded semantics (Van Gelder et al., 1991). The denotation of a program \( P \), well-founded partial model, is defined as the least fixed point of some monotonic operator \( W_P \) associated with \( P \), which gives a 3-valued model of \( \text{comp}(P) \). However, since \( W_P \) is asymmetric on the treatment of positive/negative occurrence of atoms in the clause body, well-founded semantics differs from our semantics; for example, \( p \) in a program \( \{ p \leftarrow \} \) receives \( f \) in well-founded semantics.

Supported models of a program DB are 2-valued models of the completed program \( \text{comp}(DB) \) (Apt et al., 1988; Marek and V.S.Subrahmanian, 1992). They can represent solutions of a quite large class of combinatorial problems, and hence their efficient computation is of practical interest. Also it is well-known that stable models used in answer set programming (ASP) are a subclass of supported models and when propositional programs are finite and tight, they are identical (Erdem and Lifschitz, 2003). Our proposal to use 3-valued model computation as a preprocessing step to compute supported models looks new and is applicable to stable model computation as well. It eliminates, as the experiment in Section 5 shows, the extraneous need for finding the right assignment of \( \{ t,f \} \) to the deterministic part of supported models. On the other hand, in ASP, stable models (or supported models) are computed by highly developed SAT technologies as in clingo (Gebser et al., 2019). It is an interesting future topic to merge our matricized approach with existing ASP computation mechanism.

### 7 CONCLUSION

We proposed to compute the least 3-valued completion model of a finite normal logic program DB in a vector space by first converting DB to an equivalent definite clause program DB\(^d\), the dualized version of DB, and then computing its least 2-valued model in a vector space using a matrix representing DB\(^d\), which is translated back to the least 3-valued completion model of DB. We then applied this 3-valued model computation to computing 2-valued completion models of DB, i.e. supported models of DB which are a super class of stable models. We constructed them by appropriately assigning \( t \) or \( f \) to the undefined atoms in the least 3-valued completion model of DB while guided by the completion form of clauses. We implemented the 2-valued and 3-valued completion model computation by matrix operations, and confirmed the effectiveness of 3-valued computation as a preprocessing step prior to 2-valued model computation.

Assigning truth vales to undefined atoms found in this method is the next step to compute 2-valued supported models, and verification of efficiency of this part will be reported in a full version of this paper.

### REFERENCES


