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 \{\text{true}, \text{false}, \bot (\text{undefined})\}. The construction is carried out in a vector space by matrix operations applied to the matricized version of a dualized logic program \( DB^d \) of \( DB \).\( DB^d \) is a definite clause program compiled from \( DB \) and used to compute the success set \( P_w \) as true atoms and finite failure set \( N_w \) as false atoms that constitute the least 3-valued model \( L_w = (P_w, N_w) \) 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 \( P_w \) or \( N_w \) 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.
false) atom in \( I_\infty \) remains true (resp. false) in any supported model of \( DB \), we may say that \( I_\infty \) 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 \( \mathcal{A} \) unless otherwise stated. Write \( DB = \{ a_i \leftarrow B_i \mid 1 \leq i \leq N, a_i \in \mathcal{A} \} \) where \( B_i \) is a conjunction of literals whose atoms are in \( \mathcal{A} \). 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 \( \text{iff}(a) = a \leftarrow \text{false} \). If \( k = 1 \) and \( B_1 \) is an empty conjunction, i.e. \( a \) is a unit clause, put \( \text{iff}(a) = a \leftarrow \text{true} \). Otherwise put \( \text{iff}(a) = a \leftarrow B_1 \lor \cdots \lor B_k \). The completion of \( DB \), denoted by \( \text{comp}(DB) \), is defined as \( \text{comp}(DB) = \{ \text{iff}(a) \mid a \in \mathcal{A} \} \).

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 that has three truth values \{ \text{true}, \text{false}, \bot \} (undefined). In Kleene’s 3-valued logic, when neither \( A \) nor \( B \) is \( \bot \), conjunction \( A \land B \), disjunction \( A \lor B \) and negation \( \neg A \) are evaluated as usual. However, if one of them is \( \bot \), we obey the following rules: \( A \land B = \bot \) iff one conjunct is \( \bot \) and the other is not false, \( A \lor B = \bot \) iff one disjunct is \( \bot \) and the other is not true, and \( \neg A = \bot \iff A = \text{true} \). We treat implication \( A \land B \) as \( \neg A \lor B \). Accordingly \( \bot \leftarrow \bot = \bot \).

A 3-valued interpretation \( I \) in \( \mathcal{A} \) is a pair \((P,N)\) such that \( P \cup N \subseteq \mathcal{A} \) and \( P \cap N = \emptyset \) where \( P \) stands for true atoms and \( N \) for false atoms. We next define \( [F]_I \in \{ \text{true}, \text{false}, \bot \} \), i.e. the truth value of a Boolean formula \( F \) evaluated by \( I = (P,N) \). We first define \( [\text{true}]_I = t \) and \( [\text{false}]_I = f \), then define \( [F]_I \) for \( F = a \in \mathcal{A} \) by \( [a]_I = t, f \) and \( \bot \) respectively if \( a \in P \), \( a \in N \) and otherwise and extend the definition to compound Boolean formulas according to Kleene’s 3-valued logic.

Since we are interested in \( \text{comp}(DB) \), in particular in \( \text{iff}(a) = a \leftarrow B_1 \lor \cdots \lor B_k \), and the equality of truth values of \( a \) and \( B_1 \lor \cdots \lor B_k \), we deviate here from Kleene’s 3-valued logic and interpret \( \leftrightarrow \) in a 2-valued way. We stipulate that \( [a \leftrightarrow B]_I = t \) if \( [a]_I = [B]_I \). Otherwise \( [a \leftrightarrow B]_I = f \). So \( [\bot \leftrightarrow \bot]_I = \bot \) and \( [\bot \leftrightarrow t]_I = f \). If \( [\text{iff}(a)]_I = t \) holds for every \( a \in \mathcal{A} \), we call \( I \) a 3-valued model of \( \text{comp}(DB) \) or equivalently a 3-valued completion model of \( DB \).

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 \( I_{n+1} = (P_{n+1},N_{n+1}) \), \( P_n = \{ a \in \mathcal{A} \mid \text{there is } a \leftarrow B \text{ in DB s.t. } [B]_{I_{n-1}} = t \} \), \( N_n = \{ a \in \mathcal{A} \mid \text{for all } a \leftarrow B \text{ in DB, } [B]_{I_{n-1}} = f \} \).

It is straightforward to verify \( P_n \cap N_n = \emptyset \) (n=0,1,...), and hence \( I_0 = (P_0,N_0) \) is a 3-valued interpretation. Let (\( A_1, A_2 \)) and \( (B_1, B_2) \) be pairs of sets. We introduce a partial ordering \( \sqsubseteq \) on pairs of sets by \( (A_1, A_2) \sqsubseteq (B_1, B_2) \) iff \( A_1 \subseteq B_1 \) and \( A_2 \subseteq B_2 \). Now we see \( I_0 \sqsubseteq I_1 \sqsubseteq \cdots \) (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 \( \text{comp}(DB) \).

**Proposition 2.** \( I_\infty \) is the least 3-valued model of \( \text{comp}(DB) \) in the sense of \( \sqsubseteq \)-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 \( \text{comp}(DB) \), 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 \( P_1 (\subseteq \mathcal{A}) \), we see \( I_1 \sqsubseteq I_\infty = (P_\infty,\mathcal{A} \setminus P_1) \) 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 \( \text{comp}(DB) \) is called a 2-valued completion model of DB.
ported model is obtained by appropriately assigning \( t \) or \( f \) to the undefined atoms in \( U_\infty = \mathcal{A} \setminus (P_\infty \cup N_\infty) \). Supported models are a super class of stable models and when \( DB \) is tight (no looping caller-callee chain through positive goals), \( DB \)'s supported models and \( DB \)'s stable models coincide (Erdem and Lifschitz, 2003).

\[
\text{DB}_0 = \begin{cases} 
  a \leftarrow \neg b \land c \\
  b \leftarrow \neg a \land c \\
  c \leftarrow \neg d \\
  a \leftrightarrow \neg b \land c \\
  b \leftrightarrow \neg a \land c \\
  c \leftrightarrow \neg d \\
  d \leftrightarrow \text{false} 
\end{cases}
\]

\[
\text{comp}(\text{DB}_0) = \begin{cases} 
  a \leftarrow \neg b \land c \\
  b \leftarrow \neg a \land c \\
  c \leftarrow \neg d \\
  a \leftrightarrow \neg b \land c \\
  b \leftrightarrow \neg a \land c \\
  c \leftrightarrow \neg d \\
  d \leftrightarrow \text{false} 
\end{cases}
\]

Figure 2: A program \( \text{DB}_0 \) and its completion \( \text{comp}(\text{DB}_0) \).

Look at \( \text{DB}_0 \) in Figure 2. We see \( \langle P_0, N_0 \rangle = (\emptyset, \emptyset), \langle P_1, N_1 \rangle = (\{c\}, \{d\}), \langle P_2, N_2 \rangle = (\{c\}, \{d\}) \) and \( \langle P_3, N_3 \rangle = (P_2, N_2) = I_\infty \). Then \( \{a, b\} \) are undefined atoms \( U_\infty \) in \( I_\infty \). So any supported model of \( \text{DB}_0 \) is obtained by assigning \( t \) or \( f \) to each of \( a \) and \( b \) appropriately. Actually \( \langle \{a, c\}, \{b, d\} \rangle \) and \( \langle \{b, c\}, \{a, d\} \rangle \) exhaust all supported models of \( \text{DB}_0 \).

3 MATRICIZING 3-VALUED SEMANTICS

We here compute \( I_\infty = (P_\infty, N_\infty) \) 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, \( \mathcal{A} \) the set of all atoms appearing in \( DB \). For every atom \( a \in \mathcal{A} \), 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 standardized program of \( DB \). In general, if a program is standardized, every atom \( a \in \mathcal{A} \) has exactly one clause \( a \leftarrow B \) about it and \( B \) is \( \text{true}, \text{false} \), a conjunction or disjunction of literals in \( \mathcal{A} \).

Proposition 3. Let \( DB' \) be a standardized program of \( DB \). Then \( \text{comp}(DB') \) and \( \text{comp}(DB) \) have the same 3-valued completion models as far as atoms in \( \mathcal{A} \) 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 \( \mathcal{A} = \{a_1, \ldots, a_N\} \). We define a definite clause program \( DB^d \) called a dualized program of \( DB \) that can compute \( DB \)'s least 3-valued completion model \( I_\infty = (P_\infty, N_\infty) \) as its least 2-valued model.

First introduce a set of new atoms \( \bar{\mathcal{A}} = \{\overline{a_1}, \ldots, \overline{a_N}\} \). The idea is that \( \overline{a_i} \) represents \( \neg a_i \) by negation-as-failure (NAF), i.e., an SLD derivation with NAF for \( a_i \) finitely fails. We now define a syntactic function \( n(\cdot) \) as follows. First put \( n(\text{true}) = \text{false} \) and \( n(\text{false}) = \text{true} \). Next consider literals. Let \( l \) be a literal whose atom is in \( \mathcal{A} \). If \( l \) is an atom \( a_i \), define \( n(l) = \overline{a_i} \). Otherwise \( l \) is a negated atom \( \neg a_i \) and put \( n(\neg a_i) = a_i \). We extend \( n(\cdot) \) 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 \( n(S) = \{n(a_i) \mid a_i \in S\} \).

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

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

Look at \( \text{DB}_1 \) in Figure 3. It is a standardization of \( \text{DB}_0 \) in Figure 2. Notice that the least 2-valued model of \( DB_1^d \) is \( \{c, \text{false}\} \) and it exactly corresponds to the least 3-valued model \( \langle \{c\}, \{d\} \rangle \) of \( \text{comp}(DB_1) \). This is not a coincidence. We next prove that \( DB^d \) computes the least 3-valued model \( I_\infty = \text{comp}(DB) \) as its least 2-valued model.

Put \( \mathcal{A} = \{\overline{a} \mid a \in \mathcal{A}\} \) and \( \mathcal{A}' = \mathcal{A} \cup \mathcal{A} \). Equate \( J \subseteq \mathcal{A}' \) with a 2-valued interpretation s.t. \( [c]_J = \text{true} \) iff \( c \in J \) for an atom \( c \in \mathcal{A} \). Now define a series of 2-valued interpretations \( \{J_0, J_1, \ldots, J_n\} \) for \( DB^d \) by \( J_0 = \emptyset \) and \( J_n = \{c \leftarrow \overline{d} \in DB^d \} \). We have \( J_0 = J_1 \subseteq J_2 \subseteq \ldots \) 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
DB
1 =

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

DB
1d =

\[
\begin{align*}
\text{a} & \leftarrow \neg\text{b} \land \text{c} \\
\text{b} & \leftarrow \neg\text{a} \land \text{c} \\
\text{c} & \leftarrow \text{nd} \\
\text{d} & \leftarrow \text{false}
\end{align*}
\]

Figure 3: Standardized program DB
1 and its dual program DB
1d.

defining interpretations associated with J∞. The following theorem is proved.

Theorem 1. Suppose DB is a standardized normal logic program and DB
2 is the dualized program of DB. Let I∞ = (P∞, N∞) be the least 3-valued model of comp(DB) and J∞ the least 2-valued model of DB
2 respectively. Then we have (P∞, n(N∞)) = (A \cap J∞, A \cap J∞) (proof omitted).

3.3 Matricized Logic Programs

Theorem 1 tells us that the least 3-valued model I∞ = (P∞, N∞) of comp(DB) can be computed as the least 2-valued model J∞ of DB
2. Here we point out that J∞ is conveniently computable in a vector space using matrix operations. Our linear algebraic approach to completion semantics is motivated to exploit rapidly growing computer power enabled by modern parallel computation technologies such as multicores and GPUs.

Let DB = \{a1 \leftarrow B1, \ldots, aN \leftarrow BN\} be a standardized normal logic program where the bodies Bi's are either true, false, a conjunction or disjunction of atoms in A = \{a1, \ldots, aN\}. We write the dualized program as DB
2 = \{a1 \leftarrow B1, \ldots, aN \leftarrow BN, na1 \leftarrow nB1, \ldots, nan \leftarrow nBN\}. We represent DB
2 in terms of a (2N \times 2N) 0-1 matrix Q and a (2N \times 1) threshold vector θ. Q is said to be a matricized DB
2 or a program matrix for DB
2. We construct Q and θ by a procedure in Figure 4.

3Q only stores information about occurrences of atoms in the clause body and does not make a distinction between conjunction and disjunction. So we need to record supplementary information θ to recover the original DB
2.

4For a conjunction or disjunction F, we use |F| to denote the number of literals occurring in F.

- Initialize Q to a (2N \times 2N) zero matrix.
- For \(i \leq i \leq N\), do the following:
  - If B\(i\) = true, set \(Q(i,i) = θ(i) = 1\).
  - If B\(i\) = false, set \(Q(i+N,i+N) = θ(i) = 1\).
  - Otherwise, let \(a1 \leftarrow B1\) and \(na1 \leftarrow nB1\) be clauses about \(a1\) and \(na1\) in DB
2 respectively.
    - If \(B1\) is a conjunction with \(|B1| > 1\), set \(θ(i) = |B1|\) and \(θ(i+N) = 1\).
    - Otherwise \(B1\) is a disjunction. Set \(θ(i) = 1\) and \(θ(i+N) = \|B1\|\).
    - Write \(B1\) as \(I1 \land \cdots \land Im\) \((m > 0)\) where \(\land\) denotes either \(\land\) or \(\lor\).
      - For \(p(1 ≤ p ≤ m)\), put \(Q(i,j) = Q(i+N,j+N) = 1\).
      - Else \(I1\) is a negated atom \(\neg a1\) and set \(Q(i,j+N) = Q(i+N,j) = 1\).

Figure 4: Constructing a program matrix Q and a threshold vector θ for DB
2.

For \(i \leq i \leq N\), the body Bi of a1 \leftarrow Bi in DB
2 is encoded as the i-th row Q(.,i) of Q whereas the body nBi of na1 \leftarrow nBi is encoded as the i-th row \(Q(i+N,.)\) of Q. More specifically, for \(k(1 ≤ k ≤ 2N)\), when \(θ(k) > 1\), \(Q(k,.)\) represents a conjunction of atoms in A\(^2\). However, if \(θ(k) = 1\), \(Q(k,.)\) may represent true, false or a disjunction. We disambiguate them as follows. If \(Q(k,.)\) is a zero vector, \(Q(k,.)\) represents \(c_k \leftarrow \text{false} \in DB
2\). If \(Q(k,k) = 1\) \((Q(k+N,.)\) or \(Q(k-N,.)\) is a zero vector), \(Q(k,.)\) represents \(c_k \leftarrow \text{true} \in DB
2\). Otherwise \(Q(k,.)\) is a disjunction of atoms in A\(^2\). Thus DB
2 is recoverable from Q and θ.

An (8 \times 8) matrix Q1 below is a program matrix for DB1 in Figure 3. As we can see, a \leftarrow \text{nb} \land \text{c} is encoded by Q1(1,.) = (0 0 0 0 0 1 0 0), d \leftarrow \text{false} by Q1(4,.) = (0 0 0 0 0 0 0 0), and nd \leftarrow \text{true} by Q1(8,.) = (0 0 0 0 0 0 0 1). Also we set θ(1) = |nb \land c| = 2 as the threshold associated with a conjunctive clause a \leftarrow \text{nb} \land \text{c} while we set θ(5) = 1 for θ(5) associated with a disjunctive clause na \leftarrow b \lor c\(^5\).

We compute the least 2-valued model J∞ of DB
2 numerically using Q in a vector space. Let J(⊆ A\(^2\)) be an interpretation for DB
2. J is represented by a 2N dimensional 0-1 vector u\(^J\) which is constructed as follows. For \(i \leq i \leq N\), put u\(^J\)(i) = 1 if \(a_i \in J\). Otherwise u\(^J\)(i) = 0. Similarly put u\(^J\)(i+N) = 1 if \(na_i \in J\) and otherwise u\(^J\)(i+N) = 0.

5In 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 \((x) \geq \theta\) parameterized with a real number \(\theta\) by \((x) \geq \theta = 1\) if \(x \geq \theta\). Otherwise \((x) \geq \theta = 0\). We further extend \((x) \geq \theta\) to a vector notation \((u) \geq \theta\) with a threshold vector \(\theta\) in such a way that \((u) \geq \theta(i) = (u(i)) \geq \theta(i) (1 \leq i \leq n)\) when \(u\) and \(\theta\) are \(n\) dimensional vectors.

Now consider the \(k\)-th clause \(c_k \leftarrow D_k \in DB^d\) \((1 \leq k \leq 2N)\) and let \(Q(k,:)=\) the \(k\)-th row encoding \(D_k\). By construction, \(Q(k,:):u^{(J)}\) 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 \((Q(k,:):u^{(J)}) \geq \theta(k)\), or equivalently \([D_k]_J = (Q(k,:):u^{(J)}) \geq \theta(k)\). Similarly, if \(D_k\) is a disjunction, we set \(\theta(k) = 1\) and have \([D_k]_J = (Q(k,:):u^{(J)}) \geq \theta(k)\). Thus in either case, the clause body \(D_k\) of \(c_k \leftarrow D_k\) is evaluated by \(J\) as \([D_k]_J = (Q(k,:):u^{(J)}) \geq \theta(k)\), purely in a vector space in terms of matrix multiplication and thresholding. We summarize the argument so far as

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

Proposition 4 says that \(u_\infty\) is a fixed vector of \(f(u) = (Qu) \geq \theta\) 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 \(DB^d\). Define a series of \(0\)-\(1\) vectors \(\{u_n\}\) by \(u_1 = u^{(J_1)}\) and \(u_{n+1} = (Qu_n) \geq \theta\). Then \(u_n = u^{(J_n)}\) for all \(n \geq 1\) and \(\lim_{n \to \infty} u_n = u_\infty\) (proof omitted).

We restate Proposition 5 as the least \(3\)-valued model computation procedure in Figure 6. Given a dualized program \(DB^d\) 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 \(DB^d\) as a \(2N\) dimensional \(0\)-\(1\) vector \(u_\infty = u^{(J_\infty)}\), using a program matrix \(Q\) and a threshold vector \(\theta\) for \(DB^d\).

**Step 1:** Compute \(u_1 = u^{(J_1)}\)

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

\(u_{n+1} = (Qu_n) \geq \theta\) until \(u_{n+1} = u_n\)

**Step 3:** Return \(u_\infty = u_n\)

Applying the procedure in Figure 6 to \(DB^d\), we see \(u_1 = (0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1)^T\), \(u_2 = (Qu_1) \geq \theta_1 = (0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1)^T\), \(u_3 = (Qu_2) \geq \theta_1 = (0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1)^T\) which coincides with \(u^{(J_\infty)}\) encoding \(J_\infty\) as \(\{c, na\}\) for \(DB^d\) 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 \(DB\).

## 4 SUPPORTED MODEL COMPUTATION

Suppose a program \(DB = \{a_1 \leftarrow B_1, \ldots, a_N \leftarrow B_N\}\) in \(A = \{a_1, \ldots, a_N\}\) is given and the least \(3\)-valued model \(J_\infty = (P_\infty, N_\infty)\) of \(\text{comp}(DB)\) has been computed. Put \(U_\infty = A \setminus (P_\infty \cup N_\infty)\). 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

\[ J_1 = \{a \mid a \leftarrow \text{true} \in DB^d\} \cup \{na \mid na \leftarrow \text{true} \in DB^d\}. \]

So \(u_1\) encodes unit clauses in \(DB^d\).
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 $a_i \leftarrow b_i$ in DB and $b_i$ contains $a_j$. An SCC is a set of atoms written as $[a] = \{a\} \cup \{b \mid \text{there exist paths from } a \text{ to } b \text{ and from } b \text{ to } a \text{ in } G_{U_\infty}\}$ 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 $[S_1,S_2,\ldots]$ s.t. whenever $S_i \prec S_j, i < j$ 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 comp(DB$_2$) is $\{0,\emptyset\}$ 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 $SCC_{DB_2} = \{(c,d), \{a,b\}\}$.

$$DB_2 = \begin{cases}
   a \leftarrow \neg b \land c \\
   b \leftarrow \neg a \land c \\
   c \leftarrow \neg d \\
   d \leftarrow c
\end{cases}$$

$SCC_{DB_2} = \{(c,d), \{a,b\}\}$

Figure 7: DB$_2$ having two SCCs.

Let $SCC_{DB} = \{SCC_1, SCC_2, \ldots\}$ 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 $SCC_j, k (j < k)$ preceding SCC$\_j$ have been processed and their atoms are already assigned t or f. Write $SCC_j = \{b_1, \ldots, b_K\}$ and let $DB^{SCC}_j = \{b_j \leftarrow W_{1j} \mid 1 \leq j \leq K\}$ be a subprogram of DB about atoms in $SCC_j$. Since undefined atoms that appear in $DB^{SCC}_j$ other than $SCC_i$ are already assigned t or f, we construct a supported model of $DB^{SCC}$ by appropriately assigning t or f to atoms in $SCC_i$ when possible.

Our task w.r.t. SCC$\_i$ is to make every $b_j \leftarrow W_{ij} \ (1 \leq j \leq K)$ in comp($SCC_i$) true (in 2-valued logic) by determining the truth values of $b_1, \ldots, b_K$. 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 $iff(b) = b \leftrightarrow c \land \neg d$ is in $DB^{SCC}$ and $\{b, c, d\}$ are all undefined. We have to find their truth values that make $iff(b)$ true. Basically this is an exhaustive search but since $iff(b)$ 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 $iff(b)$ 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 $iff(b) = b \leftrightarrow c \lor \neg d$, 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 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 $A$.

Step 1: Compute the least 3-valued model $I_\infty = (P_\infty, N_\infty)$ of comp(DB) via dualized program DB$^d$ using the procedure in Figure 6 and extract undefined atoms $U_\infty = A \setminus (P_\infty \cup N_\infty)$.

Step 2: Analyze $U_\infty$ and obtain a reverse-topologically sorted list $SCC_{DB} = \{SCC_1, \ldots, SCC_M\}$ of SCCs in $U_\infty$.

Step 3: For $i = 1$ to $M$, assign appropriately truth values $\{t, f\}$ to atoms in $SCC_i$ so that the total assignment constitutes a supported model of DB, i.e. a 2-valued model of comp(DB).

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 4.2.2$^9$. 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 $L_\omega = (P_\omega, N_\omega)$ 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 $P_\omega$ (resp. atoms in $N_\omega$) are true (resp. false) in any supported model of $DB$, we have only to consider truth value assignment for the remaining atoms, those in $U_\omega = \mathcal{A} \setminus (P_\omega \cup N_\omega)$ 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 $(P_\omega \cup N_\omega) \setminus (P_1 \cup N_1)$ 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 $\text{reduction rate} = \frac{\#\text{newly determined}}{n}$ where $\#\text{newly determined}$ 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 $\mathcal{A} = \{a_1, \ldots, a_{100}\}$, compute its least 3-valued completion model and count $\#\text{newly determined}$. 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) $a_1 \leftarrow \ldots \leftarrow a_{10} \leftarrow$ or to tautologies $a_1 \leftarrow a_1, \ldots, a_{10} \leftarrow a_{10}$. 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 $\mathcal{A}$ 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 $\#\text{empty body}$ denotes the average number of atoms having no clause about them in the randomly generated $DB_{(100,0.03)}$. Their truth values are $f$ in every supported model of $DB_{(100,0.03)}$. $\#\text{undef atom}$ is the average number of undefined atoms in the least 3-valued model of $\text{comp}(DB_{(100,0.03)})$. $\#\text{newly determined atom}$ is therefore computed as $100 - (\#\text{undef atom} + \#\text{zero body} + 10)$ when $\{a_1, \ldots, a_{10}\}$ are converted to facts. Otherwise it is $100 - (\#\text{undef atom} + \#\text{zero body})$.

In the former case (see as facts column), we observe $\#\text{newly determined atom} = 83.9$ giving reduction rate $= 83.9\%$. It means, on average, 83.9% of atoms are newly assigned $t$ or $f$ 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 $\#\text{newly determined atom}, 45.1$ 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 $\omega$ 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 1: The number of newly determined atoms.

<table>
<thead>
<tr>
<th>base atoms ${a_1, \ldots, a_{10}}$</th>
<th>as facts</th>
<th>as tautology</th>
</tr>
</thead>
<tbody>
<tr>
<td>$#\text{empty body}$</td>
<td>3.6</td>
<td>12.4</td>
</tr>
<tr>
<td>$#\text{undef atom}$</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>
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 DB, well-founded partial model, is defined as the least fixed point of some monotonic operator \( W_D \) associated with \( P \), which gives a 3-valued model of \( \text{comp}(P) \). However, since \( W_D \) 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 \( \{ p ← \} \) 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 (Erden 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, the dualized version of DB, and then computing its least 2-valued model in a vector space using a matrix representing \( \text{DB}^d \), 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 values 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


