On Tight Logic Programs and Yet Another Translation from Normal Logic Programs to Propositional Logic

Fangzhen Lin and **Jicheng Zhao**[*]
Department of Computer Science
Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong
{flin,jczhao}@cs.ust.hk

Abstract

Fages showed that if a program is tight, then every propositional model of its completion is also its stable model. Recently, Babovich, Erdem, and Lifschitz generalized Fages' result, and showed that this is also true if the program is tight *on the given model* of the completion. As it turned out, this is quite a general result. Among the commonly known benchmark domains, only Niemelä's normal logic program encoding of the Hamiltonian Circuit (HC) problem does not have this property. In this paper, we propose a new normal logic program for solving the HC problem, and show that the program is tight on every model of its completion. Experimental results showed that for many graphs, this new encoding improves the performance of both SMODELS and ASSAT(Chaff2), especially of the latter system which is based on the SAT solver Chaff2. We also propose a notion of inherently tight logic programs and show that for any program, it is inherently tight iff all its completion models are stable models. We then propose a polynomial transformation from a logic programs to one that is inherently tight, thus providing a reduction of stable model semantics to program completion semantics and SAT.

1 Introduction

It is well-known that a stable model of a logic program is also a logical model of the completion of the logic program, but the converse is not true in general. However, Fages [1994] showed that if the program is tight (i.e. it has no positive loops), then the converse is also true. Recently, Babovich, Erdem, and Lifschitz [2000] generalized Fages' result, and showed that the converse is also true if the program is tight *on the given model* of the completion. As it turned out, this is quite a general result as many interesting logic programs are indeed tight on every model of their completions. In fact, among the commonly known benchmark domains, only Niemelä's normal logic program encoding of the Hamiltonian Circuit (HC) problem [Niemelä, 1999] is not tight on its completion models. This leads naturally to the question: is there a normal logic program encoding of the HC problem that is tight on every model of its completion? At first glance, a positive answer seems easily followed from some known results. For instance, using any of the known transformation from logic programs to propositional theories (e.g. [Ben-Eliyahu and Dechter, 1996; Lin and Zhao, 2002]), one can translate a program to a set of clauses, and then back to a logic program that is tight (on every model of the program's completion). While this is certainly true, it is not really what we wanted. Given a graph, a logic program for solving the HC problem for the graph normally should have two parts - a set of facts that defines the graph in terms of vertices and edges, and a set of general rules. While the above reasoning shows that it follows from the known results that for any given graph, a tight logic program can be found for solving the HC problem on the graph, it does not follow, although it is true as we shall show in this paper, that there is a set of general rules such that for any given graph, the logic program consisting of the set of general rules and the set of facts encoding the graph is always tight.

By modifying Niemelä's encoding, we propose a new normal logic program for solving the HC problem, and show that the program is tight on every model of its completion. Compared with Niemelä's encoding, our new encoding yields much larger programs - about twice more atoms and rules. Surprisingly, despite its larger size, for both SMODELS [Simons, 2003] and ASSAT [Lin and Zhao, 2002], the new encoding performs better[1] on many randomly generated graphs. For ASSAT, it also performs better on complete graphs, which are hard for SMODELS and ASSAT partly because of the sizes of the programs corresponding to these graphs. However, our new encoding is slower than Niemelä's on some of the hand-coded hard graphs in [Lin and Zhao, 2002], which are constructed by taking a few copies of a graph and connecting these copies by some arcs. While these experimental results are mixed, they do suggest that it is worthwhile to try to encode a problem as a logic program whose stable models are the same as its completion models.

This motivated us to investigate a sufficient and necessary

[*]Corresponding author

[1]All performances in this paper are about computing *one* stable model of a program.

condition for a completion model to be a stable model. To this end, we generalize the notion of tight on a set of atoms into that of *inherently tight* on a set of atoms. Specifically, we call a program P *inherently tight* on a set S of atoms if there is a subprogram $Q \subseteq P$ such that S is a stable model of Q and Q is tight on S. We show that for any program, a completion model of this program is also its stable model iff the program is inherently tight on the model.

We then propose a polynomial transformation from any logic program to one that is inherently tight on all its completion models. This provides a reduction of stable model semantics to completion semantics and SAT. Compared to the one in [Ben-Eliyahu and Dechter, 1996] which needs n^2 extra atoms and n^3 new rules, and the one in [Lin and Zhao, 2002] which does not introduce any new atom but in the worst case may add an exponential number of new rules (clauses), our transformation introduces $O(n^2 + m)$ extra atoms and $O(n \times m)$ extra rules in the worst case, where n is the number of atoms in the original program and m the number of rules. We have observed that in all of the benchmark logic programs, the number of rules in a program is smaller than n^2. One useful feature of our transformation is that it is modular w.r.t adding new facts.

This paper is organized as follows. Section 2 introduces some logical preliminaries. Section 3 studies an encoding of the HC problem that is tight on every completion model of the program, and reports some experimental results using SMODELS and ASSAT(Chaff2). Section 4 proposes a notion of inherent tightness and a translation from a logic program to one that is inherently tight. Section 5 concludes this paper. Due to space limitations, proofs, if given, are only sketched.

2 Logical Preliminaries

Logic programs A normal logic program is a set of rules of the following form:

$$a \leftarrow b_1, \cdots, b_m, \texttt{not } c_1, \cdots, \texttt{not } c_n, \qquad (1)$$

where a is either an atom or empty, b's, and c's are atoms. When a is empty, then the rule is a constraint, and when $m = n = 0$ and a is an atom, then the rule is a fact, asserting that a is true. Given a rule r, in the following, we denote by $Head(r)$ its head atom, and $Pos(r)$ and $Neg(r)$ the set of positive body literals and the set of atoms that occur under not in the body of rule r, respectively. For any set X of atoms, we denote the set $\{\texttt{not } L : L \in X\}$ by $not(X)$. So a rule r can be represented as

$$Head(r) \leftarrow Pos(r) \cup not(Neg(r)). \qquad (2)$$

Given a program P, in the following, we let $Atom(P)$ be the set of atoms in P.

Program completion Given a logic program P, its completion, $Comp(P)$ is the union of the constraints in P and the Clark completion [Clark,] of P with constraints deleted. In other words, it consists of the following sentences:

- For any atom a, if $a \leftarrow G_1, \cdots, a \leftarrow G_n$ are all the rules about a in P, then $a \equiv G_1 \vee \cdots \vee G_n$ is in $Comp(P)$. Here we abuse the notation and write G_i both as the body of a rule and a disjunct in a formula. Its intended meaning is that when we write a body G in a formula, it stands for $true$ if G is empty, it is the conjunction of literals in G with not replaced by $\neg$ otherwise. Notice here that when $n = 0$, the equivalence for a becomes $a \equiv false$;
- If $\leftarrow G$ is a constraint in P, then $\neg G$ is in $Comp(P)$.

The stable model semantics Given a logic program P without constraints, and a set S of atoms, the Gelfond-Lifschitz transformation of P on S, written as P_S, is obtained from P as follows:

- For each atom q and any rule r in P, if $q \in Neg(r)$ and $q \notin S$, then delete the literal not q from the body of rule r.
- In the resulted set of rules, delete all those rules that still contain a negative literal in their bodies, i.e., for any rule $r \in P$, if there is a $q \in Neg(r)$ such that $q \in S$, then delete this rule r.

It is clear that for any set of atoms S, P_S is a set of rules without any negative literals. Thus P_S has a unique minimal model, which is the same as the set of atoms that can be derived from the program by resolution when rules are interpreted as implications. We denote this set as $Cons(P_S)$. A set S is a *stable model* [Gelfond and Lifschitz, 1988] of P if and only if $S = Cons(P_S)$.

Now let P be a program that may have constraints, and P' the result of deleting all constraints in P. Then a set of atoms is a stable model of P iff it is a stable model of P' and satisfies all the constraints in P.

Dependency graph The *predicate dependency graph* [Apt *et al.*,] of a logic program P is a directed graph with signed edges. The vertices are atoms mentioned in P. There is a directed positive (resp. negative) edge from vertex p to q if there is a rule r in P such that $Head(r) = q$ and $p \in Pos(r)$ (resp. $Head(r) = q$ and $p \in Neg(r)$). Informally, a positive (resp. negative) edge from p to q means that q depends positively (resp. negatively) on p.

For any two atoms α and β in P, there is a positive path from α to β if and only if there is a path from vertex α to vertex β in the dependence graph G_P and the path has no negative edges.

3 Tight logic programs and Hamiltonian Circuit problem

A program P is *tight* [Lifschitz, 1996] if there exists a level mapping λ such that, for every rule

$$a \leftarrow b_1, \cdots, b_m, \texttt{not } c_1, \cdots, \texttt{not } c_n \qquad (3)$$

in P,

$$\lambda(b_1), \cdots, \lambda(b_m) < \lambda(a). \qquad (4)$$

Here a *level mapping* is a function from literals to ordinals.

It is not hard to see that a program is tight iff its dependency graph has no positive cycles. For these programs, the program completion semantics under classical logic coincides with the stable model semantics. That is, if a program is tight, then a set of atoms is its stable model iff it is a classical propositional model of its completion [Fages, 1994]. Recently, Babovich, Erdem, & Lifschitz [2000] generalized this result to "tight on a set of literals" by modifying the mapping λ to a *partial level mapping* with respect to a set of literals. Formally, a program P is tight on a set S of atoms if there exists a partial level mapping λ with the domain S such that, for every rule (3) in P, if a and $b_1, \cdots, b_k \in S$, then

$$\lambda(b_1), \cdots, \lambda(b_k) < \lambda(a). \quad (5)$$

They showed that if a normal logic program is tight on a model of its completion, then this model is also a stable model. As a result, if the logic program is tight or tight on all models of its completion, to find stable models of this logic program, we can reduce it to SAT by computing its completion. As it turned out, this is quite a general result as many interesting logic programs, including all constraint satisfaction problems and planning problems are indeed tight on every model of their completions. In fact, among the commonly known benchmark domains, only Niemelä's normal logic program encoding of the Hamiltonian Circuit (HC) problem [Niemelä, 1999] is not always tight on its completion models.

As the experimental results reported in [Babovich *et al.*, 2000; Huang *et al.*, 2002; Lin and Zhao, 2002] showed, for problems like graph coloring and blocks world planning whose encodings in logic programs are either tight or tight on their completion models, computing a stable model of these programs using the state of art SAT solvers on their completions is a lot faster than SMODELS, which is currently the state of art specialized stable model generator for normal logic programs. For HC problems, while ASSAT still outperforms SMODELS, the improvements are not as great as in other problems. One of the reasons is that when a completion model may not be a stable model, ASSAT needs to repeatedly call a SAT solver, and the total computation time is directly proportional to the number of times a SAT solver is called. So to see if this was due to the inherent difficulty of the HC problem or because of the particularity of Niemelä's encoding [Niemelä, 1999], we came up with an encoding of the HC problem that is always tight on the completion models.

To motivate our new encoding, let's first look at Niemelä's given below:

$$hc(V,U) \leftarrow arc(V,U), \texttt{not}\ otherroute(V,U). \quad (6)$$
$$otherroute(V,U) \leftarrow arc(V,U), arc(V,W), hc(V,W), U \neq W. \quad (7)$$
$$otherroute(V,U) \leftarrow arc(V,U), arc(W,U), hc(W,U), V \neq W. \quad (8)$$
$$reached(U) \leftarrow arc(V,U), hc(V,U), reached(V), \texttt{not}\ initialnode(V). \quad (9)$$
$$reached(U) \leftarrow arc(V,U), hc(V,U), initialnode(V). \quad (10)$$
$$\leftarrow vertex(V), \texttt{not}\ reached(V). \quad (11)$$

As one can see, rules (6) - (8) guess some paths with the property that for every vertex X, if X has at least one outgoing arcs, then exactly one of these arcs is chosen to be in the paths, and similarly if X has at least one incoming arcs, then exactly one is chosen to be in the paths. The rest of the rules then make sure, especially by asserting that $reached(X)$ must be true for every vertex X, that these paths are in fact a single simple cycle that covers the whole graph. When grounded, this program is not tight because rule (9) would create cycles among $reached(X)$ atoms.

Another way of solving the problem is to first guess some paths that must go through every vertex exactly once, and then make sure that when we do not count the outgoing arc from the starting vertex, these paths are in fact a single path that does not contain any cycles. This is the idea behind our program in Figure 1 for solving the HC problem:

$$outgoing(V) \leftarrow arc(V,U), hc(V,U). \quad (12)$$
$$incoming(U) \leftarrow arc(V,U), hc(V,U). \quad (13)$$
$$\leftarrow vertex(V), \texttt{not}\ outgoing(V). \quad (14)$$
$$\leftarrow vertex(V), \texttt{not}\ incoming(V). \quad (15)$$
$$hc(V,U) \leftarrow arc(V,U), \texttt{not}\ otherroute(V,U). \quad (16)$$
$$otherroute(V,U) \leftarrow arc(V,U), arc(V,W), hc(V,W), U \neq W. \quad (17)$$
$$otherroute(V,U) \leftarrow arc(V,U), arc(W,U), hc(W,U), V \neq W. \quad (18)$$
$$reached(V,U) \leftarrow arc(V,U), hc(V,U), \texttt{not}\ initialnode(V). \quad (19)$$
$$reached(V,U) \leftarrow arc(W,U), hc(W,U), vertex(V), \texttt{not}\ initialnode(W), reached(V,W). \quad (20)$$
$$\leftarrow vertex(V), reached(V,V). \quad (21)$$

Figure 1: An encoding of the HC problem that is always tight on every model of its completion

Notice that rules (16) - (18) are the same as (6) - (8); rules (12) - (15) are needed to make sure that the chosen path must go through each vertex exactly once for the cases when a vertex has no outgoing (incoming) arcs in the graph; rules (19) - (21) use the binary predicate $reached(X,Y)$ to make sure that when we delete the chosen outgoing arc from the starting vertex, the chosen path has no cycles, thus must be a single path. Formally, it can be shown that the program correctly solves the HC problem and is always tight on every completion model.

Proposition 1 *Let $G = (V,E)$ be a graph. Let P be the union of the rules in Figure 1 and facts represented by the graph in terms of $vertex(X)$ and $arc(X,Y)$. If M is a stable model of P, then atoms of the form $hc(u,v)$ in M correspond to an HC of G. Furthermore, P is tight on every model of its completion.*

Given a graph, compared with Niemelä's encoding, ours yields a larger program because $reached$ in our program is a binary predicate. To evaluate the performance of this program, we tried SMODELS (version 2.27) and ASSAT(Chaff2) (version 1.3 with its simplification feature activated) on the same set of graphs[2] used in [Lin and Zhao,

[2]These graphs can be downloaded from ASSAT's web site http://www.cs.ust.hk/assat.

2002], which divided it into three categories: randomly generated graphs, hand-coded hard instances, and complete graphs. Our experiments were done on Sun Ultra 5 machines with 256M memory running Solaris. The times given below are in CPU seconds as reported by Unix "time" command, and include the times for running lparse for grounding the input programs. For randomly generated graphs (see Table 1), ASSAT clearly performs much better using our new encoding; SMODELS also seemed to behave better using the new encoding as the run times are more uniform now. However, for hand-coded graphs (see Table 2), which are formed by first taking several copies of a small graph and then selectively adding some arcs to connect these components, the old encoding was clearly better for SMODELS. For ASSAT, some of the graphs became easier with the new encoding, but many became harder. Finally, for complete graphs, which are hard partly because they yield huge programs both for the new and old encodings, the new encoding turned out to be better for ASSAT but a little worse for SMODELS. For instance, for the complete graph with 80 vertices, using the old encoding, ASSAT returned with a solution in 3297 seconds, and SMODELS in 5072 seconds, but with the new encoding, ASSAT took only 170 seconds, but SMODELS needed 12485 seconds.

From these mixed results, it seems that both encodings have merits, especially if one uses SMODELS. But the experiments do suggest that the new encoding, being tight on every completion model, is better for SAT-based stable model generators such as ASSAT and Cmodels[3]. We would not be surprised that with better and faster SAT solvers, the new encoding would outperform in all cases.

4 Inherently tight programs

Given the potential benefit of tight logic programs for SAT-based stable model generators, we now investigate a more general translation from an arbitrary logic program to one whose completion models are always stable models. To this end, we first define a notion of *inherently tight on a set of atoms*, and show that it is a sufficient and necessary condition for a program's completion model to be a stable model.

Definition 1 *A program P is inherently tight on a set of atoms S if there is a program Q such that $Q \subseteq P$, S is a stable model of Q and Q is tight on S.*

Proposition 2 *A normal logic program is inherently tight on all of its stable models.*

Proof: Suppose S is a stable model of P. Notice first that each rule in P_S has a corresponding one in P under Gelfond-Lifschitz transformation. Now define W, a set of rules, as follows:

1. W is initially empty, and T initially P_S;

[3]We also ran experiments for Cmodels (http://www.cs.utexas.edu/users/tag/cmodels.html). The results were very similar to the ones for ASSAT. We did not include its data here because it is not guaranteed to work when the input program is not tight, like Niemelä's encoding of the HC problem here.

Problem	SMODELS		ASSAT(Chaff2)	
	old	new	old	new
nv50a380	1.1	30.2	7.1	3.3
nv50a400	1.2	33.5	29.2	3.4
nv50a420	>2h	24.3	20.5	3.5
nv50a440	1.2	29.9	1.9	3.8
nv50a460	>2h	33.2	13.5	4.1
nv50a480	1.8	35.1	18	4.2
nv50a500	2.0	34.9	40.3	4.2
nv50a520	2.1	41.0	1.3	4.6
nv50a540	2.4	41.2	2.3	4.7
nv50a560	2.8	48.3	3.6	4.9
nv50a580	2.9	45.2	78.6	5
nv60a320	>2h	35.1	0.5	3.4
nv60a360	>2h	33.0	10.3	3.7
nv60a420	1.5	63.1	169.4	4.4
nv60a440	22.5	61.5	41.1	4.8
nv60a460	1.5	70.1	203.9	4.6
nv60a480	1.4	63.6	42	4.7
nv60a500	1.9	73.1	1.2	5.2
nv60a520	>2h	73.1	4.4	4
nv60a540	2.1	79.8	3664.8	5.6
nv60a560	1779.6	86.0	3.8	5.8
nv60a580	2.3	89.5	9.7	6.2
nv70a300	0.4	37.5	20.2	3.4
nv70a320	0.6	40.3	26.6	4
nv70a340	0.8	56.1	103.4	4.3
nv70a360	0.9	69.2	3.7	4.4
nv70a380	1.0	76.4	32.1	4.6
nv70a400	1.3	63.8	5.5	5.1
nv70a420	1.4	73.9	12.7	5.3
nv70a440	265.8	77.9	20.3	5.5
nv70a460	>2h	104	19.6	5.8
nv70a480	1.7	102.4	102.7	6.2
nv70a500	14.3	115.2	0.8	6.2
nv70a520	>2h	97.2	232.1	6.5
nv70a540	2.2	110.5	18.8	7
nv70a560	2.2	456.9	87.9	7.1
nv70a580	>2h	146.1	211.4	7.2

Table 1: Randomly generated graphs: nvXaY is a graph with X vertices and Y arcs.

Problem	SMODELS		ASSAT(Chaff2)	
	old	new	old	new
2xp30	0.1	>2h	0	>2h
2xp30.2	0.9	>2h	205.5	19.3
2xp30.2	>2h	>2h	155.9	12.9
2xp30.3	>2h	>2h	159.8	12.8
2xp30.4	>2h	>2h	3824.1	>2h
4xp20	0.1	>2h	0	38.4
4xp20.1	>2h	>2h	13.1	101.9
4xp20.2	1.3	>2h	8.2	8.6
4xp20.3	0.1	>2h	4.8	107.2

Table 2: Hand-coded hard graphs: NxG.M is a graph with N copies of graph G and some arcs connecting the copies.

2. Choose a fact $\alpha \leftarrow$ in T. Suppose r is the corresponding rule in P. Now let W be $W \cup r$, and remove all rules about α from T, and delete α in the bodies of all other rules in T;
3. Go back to Step 2 until T is empty.

It is clear that W is tight and S is one of its stable models. Thus P is inherently tight on S. ■

Theorem 1 *Let P be a normal logic program, and S a set of atoms. Then S is a stable model of P if and only if S is a model of the completion of P and P is inherently tight on S.*

Proof: $\Rightarrow$: By Proposition 2 and the fact that a stable model is also a model of a program.

$\Leftarrow$: Since S be a model of the completion of P, so S is closed under P. Since P is inherently tight on S, there is a $Q \subseteq P$ such that S is a stable model of Q and Q is tight on S. Thus S is supported by Q, hence also supported by P. Now S is supported by and closed under P, thus a stable model of P (by a theorem in [Babovich *et al.*, 2000]). ■

We say that a program is *inherently tight* if it is inherently tight on all models of its completion. Clearly, a program that is tight on all its completion models is inherently tight. By Theorem 1, if all models of its completion are also its stable model, then the program is inherently tight, and vice versa.

We now give a translation from an arbitrary logic program to one that is inherently tight. In the following, a set S of atoms in a program P is called a *strongly connected component* (SCC) if there is a path in G_P from u to v for any $u, v \in S$, and S is not a subset of any other such set, where G_P is the dependency graph of P.

Algorithm 1 *Let P be a normal logic program. the following algorithm transforms it to an inherently tight program Q.*
Let Q be empty initially.

1. *For any rule $r \in P$ of the form*

$$a \leftarrow b_1, \cdots, b_n, d_1, \cdots, d_m, \texttt{not } c_1, \cdots, \texttt{not } c_s \tag{22}$$

b_i ($1 \leq i \leq n$) and atom a are in the same SCC while d_j ($1 \leq j \leq m$) and a are not. We add the following rules into Q:

$$\begin{aligned} a_r \leftarrow \texttt{not } right(a, b_1), \cdots, \texttt{not } right(a, b_n),& \\ b_1, \cdots, b_n, d_1, \cdots, d_m, \texttt{not } c_1, \cdots, \texttt{not } c_s& \end{aligned} \tag{23}$$

$$a \leftarrow a_r \tag{24}$$

$$right(b_i, a) \leftarrow a_r \tag{25}$$

$$right(X, a) \leftarrow a_r, right(X, b_i) \tag{26}$$

In these rules, $1 \leq i \leq n$ and variable X will be instantiated to all atoms α if α and a are in the same SCC;

2. *If $r \in P$ is a constraint, add rule r it into Q.*

Intuitively, if a set M is a stable model of Q, then $a_r \in M$ iff $a \in M$ and the rule r in P 'contributes' to the calculation of $Cons(Q_M)$, meaning that rule r is actually used to derive a. $right(u, v)$ is used to record that atom u is used to prove v and this is needed to prevent loops. The following example illustrates the idea.

Example 1 *Consider the program $P = \{a \leftarrow b.\ \ b \leftarrow a.\ \ a \leftarrow \mathsf{not}\, c.\ \ c \leftarrow c.\}$. It has two completion models $\{a, b\}$ and $\{c\}$, but only the first one is a stable model. For this program, there are two SCCs: $\{a, b\}$ and $\{c\}$. Algorithm 1 transforms it into the following program Q:*

$$\begin{aligned}
&a_1 \leftarrow \mathsf{not}\, right(a, b), b.\\
&a \leftarrow a_1.\\
&right(b, a) \leftarrow a_1.\\
&right(a, a) \leftarrow a_1, right(a, b).\\
&right(b, a) \leftarrow a_1, right(b, b).\\
&a_2 \leftarrow \mathsf{not}\, c.\\
&a \leftarrow a_2.\\
&b_1 \leftarrow \mathsf{not}\, right(b, a), a.\\
&b \leftarrow b_1.\\
&right(a, b) \leftarrow b_1.\\
&right(a, b) \leftarrow b_1, right(a, a).\\
&right(b, b) \leftarrow b_1, right(b, a).\\
&c_1 \leftarrow \mathsf{not}\, right(c, c), c.\\
&c \leftarrow c_1.\\
&right(c, c) \leftarrow c_1.\\
&right(c, c) \leftarrow c_1, right(c, c).
\end{aligned}$$

The completion of the program contains the following equivalences $c \equiv c_1$, $c_1 \equiv \neg right(c, c) \wedge c$, and $right(c, c) \equiv c_1$. Thus it entails $\neg c \wedge \neg c_1 \wedge \neg right(c, c)$. So there is only one model of the completion where a and b are true and c is false, which is also the only stable model of Q. Notice that in the stable model of P, a is established by the second rule about it. Correspondingly, in the stable model of Q, a_2 is true while a_1 is not, and $right(a, b)$ is true while $right(b, a)$ is not (a is established first, and then b is derived from a).

Theorem 2 *For any normal logic program P, the program Q output by Algorithm 1 is inherently tight. Furthermore P and Q are equivalent on $Atom(P)$, that is, for any subset S of $Atom(P)$, S is a stable model of P iff there is exactly one stable model S' of Q such that such that $S = S' \cap Atom(P)$.*

Proof: We show only that Q is inherently tight. The second part of the theorem is easier.

We only need to prove that all models of $Comp(Q)$ are its stable models. Let N be a model of the completion of Q. We show that N satisfies every loop formula of Q, thus it must be a stable model of Q (Theorem 1 in [Lin and Zhao, 2002]) Suppose L is a positive loop in Q. According to Algorithm 1, either L is a loop that does not contain any atom of the form $right(u, v)$ or L is a set of atoms of the form $right(u, v)$.

If L is a loop that does not contain any atom of the form $right(u, v)$, suppose $v_1, \cdots, v_s$ are all atoms in L. L has even number of atoms. Atoms in $Atom(P)$ and newly added atoms occur along the loop alternatively. Suppose the loop is: x'_1, x_1, x'_2, x_2, $\cdots$, x'_t, x_t where x'_i are newly added atoms and x_i are atoms in $Atom(P)$ ($1 \leq i \leq t$). If all v_j ($1 \leq j \leq s$) are in N, it is known that $right(x_i, x_{i+1})$ are in N by rule 25. Further considering rule 26, we have $right(x_i, x_{i-1}) \in N$ for some i, thus rule 23 can not be satisfied for some x'_i

in the loop. So not all atoms in the loop are in N. Now that N is a model of the completion, so the loop formula [Lin and Zhao, 2002] for loop L is satisfied.

If L is a loop and it contains a set of atoms $right(a_1, a_2)$, $right(a_2, a_3)$, $\cdots$, $right(a_v, a_1)$. Suppose $right(a, b)$ is an atom in L and it is deduced by the rule $right(a, b) \leftarrow b_i, right(a, c)$. Suppose there is a rule r such that $Pos(r) \cap L = \emptyset$, if $Body(r)$ is satisfied by N, the loop formula of L is satisfied. If there is no such a r, then the rule $right(a, b) \leftarrow b_i, right(a, c)$ must be satisfied. Thus $b_i \in N$. But there is only one rule about b_i which requires $right(\alpha, \beta) \notin N$ for some $\alpha, \beta \in L$ for it to be applicable. However from our assumption that $L \subseteq N$, and L is a loop, it must be the case that $right(\alpha, \beta) \in N$ for all $\alpha, \beta \in L$. This is a contradiction, thus there must be an r as above. ■

Algorithm 1 effectively provides yet another reduction of answer set semantics to SAT by calculating the completion of the translated program. Compared with the mapping in [Ben-Eliyahu and Dechter, 1996], which always adds n^2 extra atoms and n^3 extra rules, the number of extra atoms and rules needed for our transformation depends on the number of rules whose heads and some of positive literals in their bodies belong to the same SCC. In the worst case, it needs $O(n^2 + m)$ extra atoms and $O(n \times m)$ extra rules, where n is the number of atoms and m the number of rules in the input program. Usually, m is a lot less than n^2. For instance, for Niemelä's encoding of the HC problem, given a graph with k vertices, there are at most $O(k^2)$ atoms and $O(k^3)$ rules.

One interesting feature of Algorithm 1 is that it is modular w.r.t. new facts. That is, if P' is a set of facts, then for any program P, the program returned by the algorithm on $P \cup P'$ is the union of the programs returned by it on P and P'. This feature will be handy when we extend the algorithm to programs with variables as the instantiation of a program with variables can be thought of as the union of the program and a set of ground facts.

We have implemented the algorithm and tried it on Niemelä's encoding of the HC problem, and found that for SMODELS, the translated program is still better than the original one on randomly generated graphs. But on other types of graphs, the translated program is a lot worse. But for ASSAT, the translated program was slower even on randomly generated programs. Apparently, being (inherently) tight does not always guarantee better performance even for SAT-based stable model generators.

5 Conclusions and future work

We have proposed a new solution to the Hamiltonian Circuit problem. For any given graph, the solution yields a program that is tight on every model of its completion. Compared with Niemelä's encoding of the same problem, our new encoding performs better on both randomly generated graphs and complete graphs, but is slightly worse on a set of hand-coded graphs. While these results hold for both SMODELS, which is a specialized stable model generator, and ASSAT, which makes use of SAT solver Chaff2, the performance gain seems greater for ASSAT than for SMODELS.

We also defined a notion of inherent tightness that captures a sufficient and necessary condition for a program completion model to be its stable model, and proposed an algorithm for translating an arbitrary logic program into an inherently tight one, which provides a reduction of stable model semantics to completion semantics and SAT.

In terms of future work, we are interested in finding out more about the relationships between stable model semantics and completion semantics, especially the kinds of features of an inherently tight logic program that would be particularly good for SAT-based stable model generators like ASSAT.

Acknowledgements

We thank Yuting Zhao for useful discussions related to the topics of this paper, especially for his helps in using ASSAT, and for providing some experimental data on ASSAT.

This work was supported in part by the Research Grants Council of Hong Kong under Competitive Earmarked Research Grant HKUST6205/02E.

References

[Apt *et al.*,] K. R. Apt, H. A. Blair, and A. Walker. In Jack Minker, editor, *Foundations of deductive databases and logic programming*, pages 89–148.

[Babovich *et al.*, 2000] Y. Babovich, E. Erdem, and V. Lifschitz. Fages' theorem and answer set programming. In *NMR-2000*, 2000.

[Ben-Eliyahu and Dechter, 1996] R. Ben-Eliyahu and R. Dechter. Propositional semantics for disjunctive logic programs. *Annals of Mathematics and Artificial Intelligence*, 12:53–87, 1996.

[Clark,] K. L. Clark. In H Gallaire and J. Minker, editors, *Logics and Databases*, pages 293–322.

[Fages, 1994] F. Fages. Consistency of clark's completion and existence of stable models. In *Journal of Methods of Logic in Computer Science*, volume 1, pages 51–60, 1994.

[Gelfond and Lifschitz, 1988] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *ICLP'88*, August 1988.

[Huang *et al.*, 2002] G.-S. Huang, X. Jia, C.-J. Liau, and J.-H. You. Two-literal logic programs and satisfiability representation of stable models: A comparison. In *Proc. 15th Canadian Conference on AI, LNCS, Springer*, 2002.

[Lifschitz, 1996] V. Lifschitz. Foundations of logic progrmming. In *Priciples of Knowledge Representation*, 1996.

[Lin and Zhao, 2002] F. Lin and Y. Zhao. ASSAT: Computing answer sets of a logic program by sat solvers. In *AAAI'02*, 2002.

[Niemelä, 1999] I. Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25(3-4):241–273, 1999.

[Simons, 2003] P. Simons. *Smodels: a system for computing the stable models of logic programs, version 2.27*, 2003.