**Bounded Model Checking, Answer Set Programming, and Fixed Points**

Ilkka Niemelä  

Laboratory for Theoretical Computer Science  
Helsinki University of Technology  
Finland

---

**Answer Set Programming**

- Term coined by Vladimir Lifschitz
- Roots: KR, logic programming, nonmonotonic reasoning
- Based on some formal system with semantics that assigns a theory a collection of answer sets (models).
- An **ASP solver**: computes answer sets for a theory
- Solving a problem in ASP: Encode the problem as a theory such that solutions to the problem are given by answer sets of the theory.

---

**Contents**

- Answer set programming
- ASP solvers and applications
- BMC using ASP

---

**ASP—cont’d**

- Solving a problem using ASP

\[
\begin{align*}
\text{Problem} & \quad \rightarrow \quad \text{Encoding} \quad \rightarrow \quad \text{Theory} \quad \rightarrow \quad \text{ASP solver} \quad \rightarrow \quad \text{Models} \\
\text{instance} & \quad \rightarrow \quad \text{Solutions} \\
\end{align*}
\]

- Possible formal system
  - Propositional logic
  - CSP
  - Logic programs
  - Truth assignments
  - Variable assignments
  - Stable models
Example. Bounded Model Checking

BMC uses a SAT-based ASP approach:

- The behavior of the system is unfolded up to a bounded number \((n)\) of steps (formula \(S\))
- Negation of the requirement \(R\) (formula \(\overline{R}\))
- \(S \land \overline{R}\) is satisfiable iff the system has an execution (of length at most \(n\)) violating the requirement \(R\)

Applying ASP

- Uniform encoding: separate problem specification and data
- Compact, easily maintainable representation
- Integrating KR, DB, and search techniques
- Handling dynamic, knowledge intensive applications: data, frame axioms, exceptions, defaults, closures

What is ASP Good for?

Search problems:
- Constraint satisfaction
- Planning, routing
- Computer-aided verification
- Security analysis
- Product configuration
- Combinatorics
- Diagnosis

ASP Using Logic Programs

- Logic programming: framework for merging KR, DB, and search
- PROLOG style logic programming systems not directly suitable for ASP:
  - search for proofs (not models) and produce answer substitutions
  - not entirely declarative
- In late 80s new semantical basis for “negation-as-failure” in LPs based on nonmonotonic logics: Stable model semantics
- Implementations of stable model semantics led to ASP
Example. 3-coloring

Problem: \( \text{clrd}(V, 1) \leftarrow \text{not clrd}(V, 2), \text{not clrd}(V, 3), \text{vtx}(V) \)
\( \text{clrd}(V, 2) \leftarrow \text{not clrd}(V, 1), \text{not clrd}(V, 3), \text{vtx}(V) \)
\( \text{clrd}(V, 3) \leftarrow \text{not clrd}(V, 1), \text{not clrd}(V, 2), \text{vtx}(V) \)
\( \leftarrow \text{edge}(V, U), \text{clrd}(V, C), \text{clrd}(U, C) \)

Data: \( \text{vtx}(v) \quad \text{vtx}(u) \quad \ldots \)
\( \text{edge}(v, u) \quad \text{edge}(u, w) \quad \ldots \)

3-colorings and stable models of the encoding correspond: \( v \) colored \( i \) if \( \text{clrd}(v, i) \) in the model.

Stable Models — cont’d

Program:
\( b \leftarrow \)
\( f \leftarrow b, \text{not } eb \)
\( eb \leftarrow p \)

Stable model:
\{\( b, f \)\}

Another candidate model: \{\( b, eb \)\} satisfies the rules but is not a proper stable model: \( eb \) is included for no reason.

Justifiability of stable models is captured by the notion of a \textit{reduct} of a program

\( S \) is a stable model of \( P \) if \( S = \text{LM}(P^S) \).
Example. Stable models

<table>
<thead>
<tr>
<th>S</th>
<th>P</th>
<th>P^S</th>
<th>LM(P^S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>{b,f}</td>
<td>b ←</td>
<td>b ←</td>
<td>{b,f}</td>
</tr>
<tr>
<td></td>
<td>f ← b, not eb</td>
<td>f ← b</td>
<td></td>
</tr>
<tr>
<td></td>
<td>eb ← p</td>
<td>eb ← p</td>
<td></td>
</tr>
<tr>
<td>{b,eb}</td>
<td>b ←</td>
<td>b ←</td>
<td>{b}</td>
</tr>
<tr>
<td></td>
<td>f ← b, not eb</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>eb ← p</td>
<td>eb ← p</td>
<td></td>
</tr>
</tbody>
</table>

The set \{b,eb\} is not a stable model of P but \{b,f\} is the (unique) stable model of P.

Example. Stable models — cont’d

- A program can have **none**, one, or **multiple** stable models.
- Program:  
  \[ p ← \text{not } q \]  
  \[ q ← \text{not } p \]

- Program:  
  Stable models:  
  \{p\}  
  \{q\}

- Program:  
  Stable models:  
  None

Variables

- Variables are needed for uniform encodings

  Program:  
  \[ \text{clrd}(V, 1) ← \text{not clrd}(V, 2), \text{not clrd}(V, 3), \text{vtx}(V) \]  
  \[ \text{clrd}(V, 2) ← \text{not clrd}(V, 1), \text{not clrd}(V, 3), \text{vtx}(V) \]  
  \[ \text{clrd}(V, 3) ← \text{not clrd}(V, 1), \text{not clrd}(V, 2), \text{vtx}(V) \]  

  Data:  
  \[ \text{vtx}(v) \quad \text{vtx}(u) \quad \ldots \]  
  \[ \text{edge}(v, u) \quad \text{edge}(u, w) \quad \ldots \]

Variables — cont’d

- Semantics: Herbrand models
- A rule is seen as a shorthand for the set of its ground instantiations.

  Example.  
  \[ \text{clrd}(V, 1) ← \text{not clrd}(V, 2), \text{not clrd}(V, 3), \text{vtx}(V) \]

  is a shorthand for  
  \[ \text{clrd}(v, 1) ← \text{not clrd}(v, 2), \text{not clrd}(v, 3), \text{vtx}(v) \]  
  \[ \text{clrd}(u, 1) ← \text{not clrd}(u, 2), \text{not clrd}(u, 3), \text{vtx}(u) \]  
  \[ \text{clrd}(1, 1) ← \text{not clrd}(1, 2), \text{not clrd}(1, 3), \text{vtx}(1) \]  
  \[ \ldots \]
Stable Models — cont’d

- A stratified program has a unique stable model (canonical model).
- It is linear time to check whether a set of atoms is a stable model of a ground program.
- It is NP-complete to decide whether a ground program has a stable model.
- Normal programs (without function symbols) give a uniform encoding to every NP search problem.

Extensions

For example in the Smodels system:

- Choice rules: { a } :- b, not c.
- Cardinality constraints: 2 {hd_1,...,hd_n } 4
- Weight constraints:
  20 [hd_1 =6,...,hd_n = 13]
  A.k.a. pseudo-Boolean constraints:
  20 \leq 6hd_1 + \cdots + 13hd_n
- Optimization
  minimize [hd_1 = 100,...,hd_n = 600]

Also disjunctions, preferences, weak constraints, …
Example. Propositional Satisfiability

- Consider formula $p_1: (a \lor \neg b) \land (\neg a \leftrightarrow b)$

- Encoding:
  - $\{ a \}. \{ b \}. \% \text{Choices}$
  - $\text{not } p_1. \% \text{Constraint}$
  - $p_1: p_2, p_3. \% \text{Conjunction}$
  - $p_2: a. \% \text{Disjunction}$
  - $p_2: \text{not } b. \% \text{Disjunction}$
  - $p_3: \text{not } a, b. \% \text{Equivalence}$
  - $p_3: a, \text{not } b. \% \text{Equivalence}$

- Satisfying truth assignments for $p_1$ and the stable models of the program correspond

Example. Hamiltonian cycles

A Hamiltonian cycle: a closed path that visits all vertices of the graph exactly once.

**Data**

$vtx(a). \ldots$

$edge(a,b). \ldots$

$\text{init}_vtx(a0). \% \text{for some vertex } a0$

**Problem encoding**

$\{ \text{hc}(X,Y) \} :\text{edge}(X,Y).$

$\text{hc}(X,Y), \text{hc}(X,Z), Y=Z.$

$\text{hc}(Y,X), \text{hc}(Z,X), Y=Z.$

$\text{vtx}(X), \text{not } r(X).$

$r(Y) :\text{hc}(X,Y), \text{init}_vtx(X).$

$r(Y) :\text{hc}(X,Y), r(X).$

Fixed Points

- The stable model semantics captures inherently **minimal fixed points** enabling compact encodings of closures

- Example. Reachability from node $s$.

  $r(s). \% \text{source}$

  $r(v) :\text{r}(w). \% \text{for each edge } (w,v)$

- The program is **linear size and captures reachability**: it has a unique model $S$ s.t. $v$ is reachable from $s$ iff $r(v) \in S$.

- Example. Transitive closure of relation $q(X,Y)$

  $t(X,Y) : q(X,Y).$

  $t(X,Y) : q(X,Z), t(Z,Y).$

ASP vs Other Approaches

- **SAT, CSP, (M)IP**

  - **Similarities**: search for models (assignments to variables) satisfying a set of constraints

  - **Differences**: no logical variables, fixed points, database or DDB techniques available, search space given by variable domains

- **LP, CLP**

  - **Similarities**: database and DDB techniques

  - **Differences**: Search for proofs (not models), non-declarative features
ASP Solvers and Applications

ASP Solvers

- ASP solvers need to handle two challenging tasks
  - complex data
  - search
- The approach has been to use
  - logic programming and deductive data base techniques for the former
  - SAT/CSP related search techniques for the latter
- In the current systems: separation of concerns
  - A two level architecture

Architecture of ASP Solvers

Typically a two level architecture employed

- **Grounding** step handles complex data:
  - Given program $P$ with variables, generate a set of ground instances of the rules which preserves the models.
  - LP and DDB techniques employed
- **Model search** for ground programs:
  - Special-purpose search procedures
  - Translation to SAT
    - propositional models and stable models are closely related via (Clark’s) program completion

Program Completion

- Program completion $\text{comp}(P)$: a simple translation of a logic program $P$ to a propositional formula.
  - **Example**.
    - $P$:
      - $a \leftarrow b, \neg c$
      - $a \leftarrow \neg b, d$
      - $a, d$
    - $\text{comp}(P)$:
      - $a \leftrightarrow ((b \land \neg c) \lor (\neg b \land d))$
      - $\neg b, \neg c, \neg d$
      - $\neg (a \land \neg d)$
- For **tight programs** (no positive recursion) stable models of a logic program and propositional models of its completion coincide.
Program Completion — cont’d

For non-tight programs (with positive recursion) there are differences:

\[
\begin{align*}
    p & \leftarrow q \\
    q & \leftarrow p
\end{align*}
\]

ASP solver: unique model: \{\}

SAT solver: 2 models: \{\}, \{p, q\}

Approaches to extend SAT solvers

- Extend completion with loop formulas dynamically (ASSAT, CMODELS)
- One pass compilation to SAT

\[
O(||P|| \times \log |At(P)|)\]

(Janhunen, ECAI 2004)

ASP Implementations

<table>
<thead>
<tr>
<th>Name</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>dlv</td>
<td><a href="http://www.dbai.tuwien.ac.at/proj/dlv/">http://www.dbai.tuwien.ac.at/proj/dlv/</a></td>
</tr>
<tr>
<td>nomore++</td>
<td><a href="http://www.cs.uni-potsdam.de/nomore/">http://www.cs.uni-potsdam.de/nomore/</a></td>
</tr>
<tr>
<td>XASP</td>
<td>distributed with XSB v2.6</td>
</tr>
<tr>
<td>aspps</td>
<td><a href="http://www.cs.engr.uky.edu/ai/aspps/">http://www.cs.engr.uky.edu/ai/aspps/</a></td>
</tr>
</tbody>
</table>

SAT and ASP

Due to close relationship results carry over

- **Restarting** has been found useful in SAT/CSP
  Used for example in `smodels -restart`

- Modern SAT solvers employ **conflict driven learning and backjumping**
  First ASP attempt (Ward, Schlipf, 2004)

- SAT solvers use **watched literal** data structures to achieve efficient propagation for large clause sets

- ASP solvers have **built-in support for aggregates** (cardinality and weight constraints)
  Efficient techniques for (boolean combinations of) pseudo-Boolean constraints

Applications

- **Planning**
  - USAvisor project at Texas Tech: A decision support system for the flight controllers of space shuttles

- **Product configuration**
  - Intelligent software configurator for Debian/Linux
  - WeCoTin project (Web Configuration Technology)
Applications—cont’d

- VLSI routing, planning, combinatorial problems, network management, network security, security protocol analysis, linguistics...
- WASP Showcase Collection
  http://www.kr.tuwien.ac.at/projects/WASP/showcase.html

Encoding BMC Problems

- BMC problem
  INPUT: A system description $N$ (with some initial conditions $C_0$), a bound $n$, and a requirement $R$.
  QUESTION: Is there an execution of system $N$ of length at most $n$ (starting from some initial state satisfying $C_0$) that violates $R$.

- The encoding of a BMC problem can be divided into two (orthogonal) tasks
  - encoding of executions of $N$ of length $n$
  - encoding of requirement $R$

Encoding BMC problems—cont’d

- Given a BMC problem we need to construct two programs (sets of formulas)
- $\text{Exe}(N,n)$:
  a model of $\text{Exe}(N,n)$ corresponds to an execution of $N$ in $n$ steps (starting from some initial state satisfying $C_0$).
- $\text{Req}(\neg R,n)$:
  a model of $\text{Req}(\neg R,n)$ corresponding to an execution of length $n$ satisfies $\neg R$.

BMC Using ASP
Encoding BMC problems—cont’d

- **Soundness:**
  If \( \text{Exec}(N, n) \cup \text{Req}(\neg R, n) \) has a model, then there is an execution of \( N \) with at most \( n \) steps where \( R \) does not hold.

- **Completeness:**
  If there is an execution of \( N \) with at most \( n \) steps where \( R \) does not hold, then \( \text{Exec}(N, n) \cup \text{Req}(\neg R, n) \) has a model.

### Encoding the executions

We assume that executions are encoded such that

- each model \( I \) of \( \text{Exec}(N, n) \) corresponds to an execution of \( N \) in \( n \) steps with

\[
M_0 \xrightarrow{t_0} M_1 \xrightarrow{t_1} \ldots M_{n-1} \xrightarrow{t_{n-1}} M_n
\]

where

state variable \( p \) holds in state \( M_i \) iff \( p(i) \) is true in \( I \)

### Requirements—LTL

- **LTL:** prop. logic + temporal operators \( (U, F, G, X, \ldots) \)
- **LTL formula is evaluated over an infinite sequence of states** \( w = M_0, M_1, M_2, \ldots \)
- \( w \models p \ U q \) iff \( p \) holds until \( q \) holds in some state in \( w \).
- \( w \models F p \) iff for some state in \( w \), \( p \) holds \( (\top U p) \)
- \( w \models G p \) iff for all states in \( w \), \( p \) holds \( (\neg (\top U \neg p)) \)

**Examples:**
- **Safety:** \( \neg (\neg \text{reqUack}) \)
- **Liveness:** \( G(\text{req} \rightarrow \text{Fack}) \)
- **Fairness:** \( GFen \rightarrow GFex \)
Encoding LTL Requirements

- For an LTL formula \( \varphi \) (negation of the requirement), \( \text{Req}(\varphi, n) \) eliminates models not satisfying \( \varphi \).
- \( \text{Req}(\varphi, n) \):
  1. rules capturing the conditions under which a model corresponds to an execution satisfying \( \varphi \)
  2. rule

\[ \leftarrow \text{not } \varphi(0) \]

to eliminate models not satisfying \( \varphi \) in an initial state.

LTL encoding—cont’d

Guess a loop point: \( \{l(0), l(1), \ldots, l(n-1)\} \)
Check it:
\[
\leftarrow l(i), p(i), \text{not } p(n)
\]
\[
\leftarrow l(i), p(n), \text{not } p(i)
\]
Next of the last state: \( nl(i+1) \leftarrow l(i) \)

\[
 M_0 \quad M_{i} \quad M_{n}
\]

LTL requirements—cont’d

- Consider looping bounded executions
- Treating non-looping ones is a straightforward extension

\[
 M_{i} \equiv M_{n}
\]

LTL encoding

- \( \text{Req}(\varphi, n) \): Formula \( \varphi \) is translated recursively starting from its subformulas
- Translation of \( \varphi = \varphi_1 U \varphi_2 \) based on the fixed point characterization \( \varphi_1 U \varphi_2 \equiv \varphi_2 \lor (\varphi_1 \land X(\varphi_1 U \varphi_2)) \)

\[
\varphi(i) \leftarrow \varphi_2(i)
\]
\[
\varphi(i) \leftarrow \varphi_1(i), \varphi(i+1)
\]
\[
\varphi(n+1) \leftarrow nl(i), \varphi(i)
\]

Example.

\[
f = p0 U \left( \neg p1 \land p2 \right) : \]
\[
f_1(i) \leftarrow \text{not } p1(i)
\]
\[
f_2(i) \leftarrow f_1(i), p2(i)
\]
\[
f(i) \leftarrow f_2(i)
\]
\[
f(i) \leftarrow p0(i), f(i+1)
\]
\[
f(n+1) \leftarrow nl(i), f(i)
\]
Comparison

- SAT based encoding [Biere et al./Cimatti et al.]:
  - size is at least quadratic in the bound
- Logic program encoding
  - size is linear in the bound, system description, and LTL formula

Exploiting Concurrency

- Inherent concurrency of an asynchronous system can be exploited by allowing multiple independent actions to occur together (step semantics):
  - Change $\text{Exec}(N, n)$ to allow steps.
  - $\text{Req}(\phi, n)$: For step semantics, allow at most one visible action in a step by adding:
    
    $$\leftarrow 2\{t_1(i), \ldots, t_k(i)\}$$

    where $\{t_1, \ldots, t_k\}$ is the set of visible actions, i.e., the actions whose firing changes the truth value of an atom $p$ appearing in the formula $\phi$.

  - $(X$ cannot be used)

Example

$$\text{Exec}(N, n)$$:

- Free initial marking
  - $\{p1(0)\} \leftarrow \text{not } p1(0)$
  - $\{p2(0)\} \leftarrow \text{not } p2(0)$
- Initial conditions
  - $\{p3(0)\} \leftarrow \text{not } p3(0)$
  - $\{p4(0)\} \leftarrow \text{not } p4(0)$
  - $\{p5(0)\} \leftarrow \text{not } p5(0)$

Preconditions

- $\{t1(i)\} \leftarrow p3(i)$
- $\{t2(i)\} \leftarrow p1(i), p2(i)$
- $\{t3(i)\} \leftarrow p2(i)$
- $\{t4(i)\} \leftarrow p4(i)$
- $\{t5(i)\} \leftarrow p2(i)$

Effects

- $p1(i + 1) \leftarrow t1(i)$
- $p2(i + 1) \leftarrow t4(i)$
- $p3(i + 1) \leftarrow t2(i)$
- $p4(i + 1) \leftarrow t2(i)$
- $p5(i + 1) \leftarrow t5(i)$

Frame axioms

- $\text{idling only at start:}$
  - $\leftarrow 2\{t2(i), t3(i), t5(i)\}$
  - $\leftarrow \text{idle}(i + 1), \text{not } \text{idle}(i)$
  - $\leftarrow \text{idle}(n - 1)$

Conflicts:

- $\text{idle}(i), \ldots, \text{not } t5(i)$

Experiments

- Deadlock checking/LTL checking using a benchmark set proposed by Corbett [1995]
- Experiments using step and interleaving semantics
- ASP solver: Smodels 2.26
- Comparison with NuSMV 2.1.0
  - NuSMV/BMC: NuSMV with optimized Biere et al.
  - translation and zChaff
  - NuSMV/BDD: NuSMV with tableau-based LTL using BDDs


...
Experiments—cont’d

<table>
<thead>
<tr>
<th>Problem</th>
<th>n</th>
<th>St</th>
<th>Int</th>
<th>Bmc</th>
<th>Bdd</th>
<th>States</th>
</tr>
</thead>
<tbody>
<tr>
<td>DP(6)</td>
<td>7</td>
<td>0.2</td>
<td>8</td>
<td>4.3</td>
<td>728</td>
<td></td>
</tr>
<tr>
<td>DP(8)</td>
<td>8</td>
<td>1.5</td>
<td>10</td>
<td>64.0</td>
<td>59048</td>
<td></td>
</tr>
<tr>
<td>DP(10)</td>
<td>9</td>
<td>25.9</td>
<td>12</td>
<td>1257.1</td>
<td>&gt;1800</td>
<td></td>
</tr>
<tr>
<td>DP(12)</td>
<td>10</td>
<td>889.4</td>
<td>14</td>
<td>&gt;1800</td>
<td>&gt;1800</td>
<td></td>
</tr>
</tbody>
</table>

For instance for six philosophers:

\[ \neg\mathit{GF}(f_5.\text{up} U (p_5.\text{eat} \land (f_3.\text{up} U (p_3.\text{eat} \land (f_1.\text{up} U p_1.\text{eat}))))]) \]

http://www.tcs.hut.fi/~kepa/experiments/boundsmodels/

Conclusions

- **ASP = KR + DB + search**
- ASP emerging as a viable KR tool
- Efficient implementations under development
  (Smodels, aspps, dlv, XASP, CMODELS, ASSAT, nomore++, clasp, ...)
- Logic programming based ASP supports directly (least) **fixed points** useful in many applications: encoding temporal properties, configurations, planning, ...
- Exploiting concurrency in asynchronous models computationally advantageous

Further Work

- Exploiting concurrency
- Linear size encoding in SAT
  [Latvala, Biere, Heljanko, Junntila; FMCAD’2004]
- Incrementality and Past LTL
  [Heljanko et al., CAV’2005]
  Implemented in NuSMV 2.4.0

For instance for six philosophers:

\[ \neg\mathit{GF}(f_5.\text{up} U (p_5.\text{eat} \land (f_3.\text{up} U (p_3.\text{eat} \land (f_1.\text{up} U p_1.\text{eat}))))]) \]