Bounded Model Checking, Answer Set Programming, and Fixed Points

Ilkka Niemelä
Ilkka.Niemela@tkk.fi, http://www.tcs.hut.fi/~ini/

Laboratory for Theoretical Computer Science
Helsinki University of Technology
Finland

Contents

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

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.

ASP—cont'd

- Solving a problem using ASP

Problem instance						Models (Solutions)
$\longrightarrow$	Encoding	Theory $\longrightarrow$	ASP solver	$\longrightarrow$		

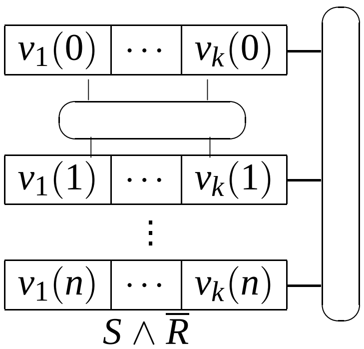
-

Possible formal system	Models
Propositional logic	Truth assignments
CSP	Variable assignments
Logic programs	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 $\overline{R}$)
- $S \wedge \overline{R}$ is satisfiable iff the system has an execution (of length at most n) violating the requirement R



What is ASP Good for?

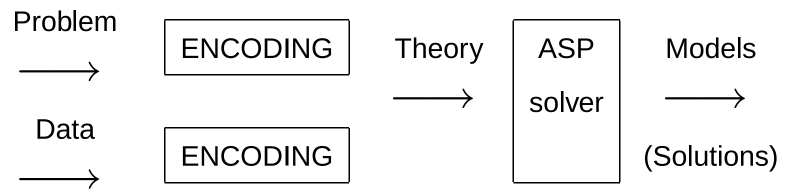
Search problems:

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

☞ Declarative problem solving

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



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:

$$
\begin{aligned}
& clrd(V,1) \leftarrow \text{not } clrd(V,2), \text{not } clrd(V,3), vtx(V) \\
& clrd(V,2) \leftarrow \text{not } clrd(V,1), \text{not } clrd(V,3), vtx(V) \\
& clrd(V,3) \leftarrow \text{not } clrd(V,1), \text{not } clrd(V,2), vtx(V) \\
& \leftarrow edge(V,U), clrd(V,C), clrd(U,C)
\end{aligned}
$$

Data:

$$
\begin{array}{lll}
vtx(v) & vtx(u) & \ldots \\
edge(v,u) & edge(u,w) & \ldots
\end{array}
$$

☞ 3-colorings and stable models of the encoding correspond: v colored i iff $clrd(v,i)$ in the model.

LPs with Stable Models Semantics

- Consider normal logic program rules

$$A \leftarrow B_1, \ldots, B_m, \text{not } C_1, \ldots, \text{not } C_n$$

- Seen as constraints on an answer set (stable model):
  - if $B_1, \ldots, B_m$ are in the set and
  - none of $C_1, \ldots, C_n$ is included,

  then A must be included in the set
- A stable model is a set of atoms
  (i) which satisfies the rules and
  (ii) where each atom is **justified** by the rules.

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 **reduct** of a program

  ☞ The stable model semantics [Gelfond/Lifschitz,1988].

Stable Models — cont'd

- Consider the propositional (variable free) case:
  P — ground program
  S — set of ground atoms
- Reduct P^S (Gelfond-Lifschitz)
  - delete each rule having a body literal not C with $C \in S$
  - remove all negative body literals from the remaining rules
- P^S is a definite program with unique least model $LM(P^S)$
- **S is a stable model of P iff $S = LM(P^S)$.**

Example. Stable models

S	P	P^S	$LM(P^S)$
$\{b,f\}$	$b \leftarrow$	$b \leftarrow$	$\{b,f\}$
	$f \leftarrow b, \text{not } eb$	$f \leftarrow b$	
	$eb \leftarrow p$	$eb \leftarrow p$	
$\{b,eb\}$	$b \leftarrow$	$b \leftarrow$	$\{b\}$
	$f \leftarrow b, \text{not } eb$		
	$eb \leftarrow p$	$eb \leftarrow p$	

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

Example. Stable models

- A program can have **none**, one, or **multiple** stable models.
- Program: Stable models:

$p \leftarrow \text{not } q$ $\{p\}$

$q \leftarrow \text{not } p$ $\{q\}$

- Program: Stable models:

$p \leftarrow \text{not } q$ None

$q \leftarrow \text{not } p$

$\leftarrow \text{not } p$

$\leftarrow \text{not } q$

Variables

- Variables are needed for uniform encodings

Program:

$clrd(V,1) \leftarrow \text{not } clrd(V,2), \text{not } clrd(V,3), vtx(V)$

$clrd(V,2) \leftarrow \text{not } clrd(V,1), \text{not } clrd(V,3), vtx(V)$

$clrd(V,3) \leftarrow \text{not } clrd(V,1), \text{not } clrd(V,2), vtx(V)$

$\leftarrow edge(V,U), clrd(V,C), clrd(U,C)$

Data:

$vtx(v)$ $vtx(u)$ $\ldots$

$edge(v,u)$ $edge(u,w)$ $\ldots$

Variables — cont'd

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

Example.

$$clrd(V,1) \leftarrow \text{not } clrd(V,2), \text{not } clrd(V,3), vtx(V)$$

is a shorthand for

$$clrd(v,1) \leftarrow \text{not } clrd(v,2), \text{not } clrd(v,3), vtx(v)$$
$$clrd(u,1) \leftarrow \text{not } clrd(u,2), \text{not } clrd(u,3), vtx(u)$$
$$clrd(1,1) \leftarrow \text{not } clrd(1,2), \text{not } clrd(1,3), vtx(1)$$
$$\ldots$$

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, . . .

Problem Encoding with ASP

Generate-and-test programming

- Basic methodology:
  - **Generator rules**: provide candidate answer sets (typically encoded using choice constructs)
  - **Tester rules**: eliminate non-valid candidates (typically encoded using integrity constraints)
  - **Optimization statements**: Criteria for preferred answer sets (typically encoded using cost functions)

Example. Propositional Satisfiability

- Consider formula p_1: $\underbrace{(a \vee \neg b)}_{p_2} \wedge \underbrace{(\neg a \leftrightarrow b)}_{p_3}$
- Encoding:

```
{ a }. { b }.  % Choices
:- not p1.     % Constraint
p1:- p2, p3.   % Conjunction
p2:- a.        % Disjunction
p2:- not b.    % Disjunction
p3:- not a, b. % Equivalence
p3:- a, not b. % Equivalence
```

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

Fixed Points

- The stable model semantics captures inherently **minimal fixed points** enabling compact encodings of **closures**
- Example. Reachability from node s.

```
r(s).              % source
r(v) :- r(w).      % 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).
```


Example. Hamiltonian cycles

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

```
% Data
vtx(a). ...
edge(a,b). ...
init_vtx(a0).   %for some vertex a0
% Problem encoding
{ hc(X,Y) } :- edge(X,Y).
:- hc(X,Y), hc(X,Z), Y!=Z.
:- hc(Y,X), hc(Z,X), Y!=Z.
:- vtx(X), not r(X).
r(Y) :- hc(X,Y), init_vtx(X).
r(Y) :- hc(X,Y), r(X).
```


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 $\mathrm{comp}(P)$: a simple translation of a logic program P to a propositional formula.
  Example.

$P:$	$\mathrm{comp}(P):$
$a \leftarrow b, \mathrm{not}\ c$	$a \leftrightarrow ((b \land \neg c) \lor (\neg b \land d))$
$a \leftarrow \mathrm{not}\ b, d$	$\neg b, \neg c, \neg d$
$\leftarrow a, \mathrm{not}\ d$	$\neg(a \land \neg d)$

- 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

$p \leftarrow q$		$p \leftrightarrow q$
$q \leftarrow p$	vs	$q \leftrightarrow p$
ASP solver:		SAT solver:
unique model: $\{\}$		2 models: $\{\}, \{p, q\}$

- Approaches to extend SAT solvers
  - Extend completion with **loop formulas dynamically** (`ASSAT`, `CMODELS`)
  - One pass compilation to SAT $O(\|P\| \times \log |At(P)|)$ translation (Janhunen, ECAI 2004)

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

ASP Implementations

`Smodels`	`http://www.tcs.hut.fi/Software/smodels/`
`dlv`	`http://www.dbai.tuwien.ac.at/proj/dlv/`
`GnT`	`http://www.tcs.hut.fi/Software/gnt/`
`CMODELS`	`http://www.cs.utexas.edu/users/tag/cmodels.html`
`ASSAT`	`http://assat.cs.ust.hk/`
`nomore++`	`http://www.cs.uni-potsdam.de/nomore/`
`XASP`	distributed with XSB v2.6
	`http://xsb.sourceforge.net`
`aspps`	`http://www.cs.engr.uky.edu/ai/aspps/`
`ccalc`	`http://www.cs.utexas.edu/users/tag/cc/`

Applications

- Planning
  USAdvisor 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)
  –Spin-off (`http://www.variantum.com/`)

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`
- C. Baral. Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press, 2003.

BMC Using ASP

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)
  - $\mathrm{Exe}(N,n)$:
    a **model** of $\mathrm{Exe}(N,n)$ corresponds to an **execution** of N in n steps (starting from some initial state satisfying C_0).
  - $\mathrm{Req}(\neg R,n)$:
    a model of $\mathrm{Req}(\neg R,n)$ corresponding to an execution of length n **satisfies** $\neg R$.

Encoding BMC problems—cont'd

- Soundness:
  If $\mathrm{Exe}(N,n) \cup \mathrm{Req}(\neg R,n)$ 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 $\mathrm{Exe}(N,n) \cup \mathrm{Req}(\neg R,n)$ has a **model**.

Encoding the executions

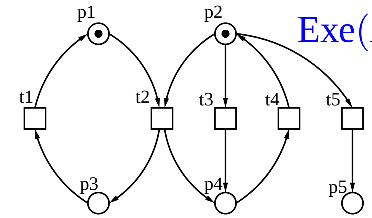
We assume that executions are encoded such that

- each **model** I of $\mathrm{Exe}(N,n)$ 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

Example



$\mathrm{Exe}(N,n)$:

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

Preconditions	Effects	Frame axioms
$\{t1(i)\} \leftarrow p3(i)$	$p1(i+1) \leftarrow t1(i)$	$p1(i+1) \leftarrow p1(i), \mathrm{not}\, t2(i)$
$\{t2(i)\} \leftarrow p1(i), p2(i)$	$p2(i+1) \leftarrow t4(i)$	$p2(i+1) \leftarrow p2(i), \mathrm{not}\, t2(i),$
$\{t3(i)\} \leftarrow p2(i)$	$p3(i+1) \leftarrow t2(i)$	$\mathrm{not}\, t3(i), \mathrm{not}\, t5(i)$
$\{t4(i)\} \leftarrow p4(i)$	$p4(i+1) \leftarrow t2(i)$	$p3(i+1) \leftarrow p3(i), \mathrm{not}\, t1(i)$
$\{t5(i)\} \leftarrow p2(i)$	$p4(i+1) \leftarrow t3(i)$	$p4(i+1) \leftarrow p4(i), \mathrm{not}\, t4(i)$
	$p5(i+1) \leftarrow t5(i)$	$p5(i+1) \leftarrow p5(i)$

Interleaving:	Idling only at start:	
$\leftarrow 2\, \{t1(i), t2(i), t3(i), t4(i), t5(i)\}$	$\leftarrow idle(i+1), \mathrm{not}\, idle(i)$	$idle(i) \leftarrow \mathrm{not}\, t1(i), \ldots, \mathrm{not}\, t5(i)$
	$\leftarrow idle(n-1)$	

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: $\neg(\neg req U ack)$
  Liveness: $G(req \rightarrow F ack)$
  Fairness: $GF en \rightarrow GF ex$

Encoding LTL Requirements

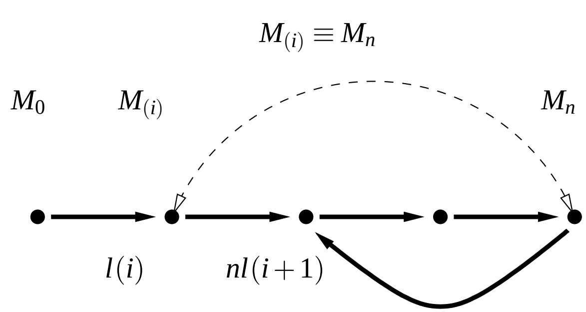
- For an LTL formula φ (negation of the requirement), $\mathrm{Req}(\varphi, n)$ **eliminates** models **not satisfying φ**.
- $\mathrm{Req}(\varphi, n)$:
  (i) rules capturing the conditions under which a model corresponds to an execution satisfying φ
  (ii) rule

$$\leftarrow \text{not } \varphi(0)$$

  to eliminate models not satisfying φ in an initial state.

LTL requirements—cont'd

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



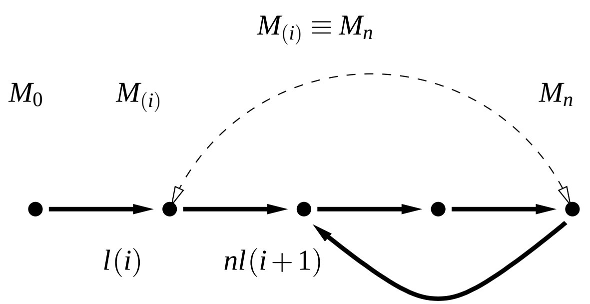
LTL encoding—cont'd

Guess a loop point: $1\{l(0), l(1), \ldots, l(n-1)\}1$

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: $nl(i+1) \leftarrow l(i)$



LTL encoding

- $\mathrm{Req}(\varphi, n)$: Formula φ is translated recursively starting from its subformulas
- Translation of $\varphi = \varphi_1 U \varphi_2$ based on the fixed point characterization $\varphi_1 U \varphi_2 \equiv \varphi_2 \vee (\varphi_1 \wedge X(\varphi_1 U \varphi_2))$
  $\varphi(i) \leftarrow \varphi_2(i)$
  $\varphi(i) \leftarrow \varphi_1(i), \varphi(i+1)$
  $\varphi(n+1) \leftarrow nl(i), \varphi(i)$
- Example.

$$f = p0\, U\, \underbrace{(\overbrace{\neg p1}^{f_1} \wedge p2)}_{f_2}:$$

$f_1(i) \leftarrow \text{not } p1(i)$
$f_2(i) \leftarrow f_1(i), p2(i)$
$f(i) \leftarrow f_2(i)$
$f(i) \leftarrow p0(i), f(i+1)$
$f(n+1) \leftarrow nl(i), f(i)$

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

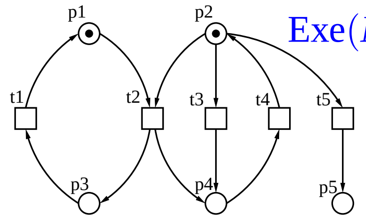
Exploiting Concurrency

- Inherent concurrency of an asynchronous system can be exploited by allowing multiple independent actions to occur together (step semantics):
  - Change $\mathrm{Exe}(N,n)$ to allow steps.
  - $\mathrm{Req}(\varphi,n)$: 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 φ.
  - (X cannot be used)

Example



$\mathrm{Exe}(N,n)$:

Free initial marking

$\{p1(0)\} \leftarrow$
$\{p2(0)\} \leftarrow$
$\{p3(0)\} \leftarrow$
$\{p4(0)\} \leftarrow$
$\{p5(0)\} \leftarrow$

Initial conditions

$\leftarrow \mathit{not}\, p1(0)$
$\leftarrow \mathit{not}\, p2(0)$

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

$p1(i+1) \leftarrow t1(i)$
$p2(i+1) \leftarrow t4(i)$
$p3(i+1) \leftarrow t2(i)$
$p4(i+1) \leftarrow t2(i)$
$p4(i+1) \leftarrow t3(i)$
$p5(i+1) \leftarrow t5(i)$

Frame axioms

$p1(i+1) \leftarrow p1(i), \mathrm{not}\, t2(i)$
$p2(i+1) \leftarrow p2(i), \mathrm{not}\, t2(i), \mathrm{not}\, t3(i), \mathrm{not}\, t5(i)$
$p3(i+1) \leftarrow p3(i), \mathrm{not}\, t1(i)$
$p4(i+1) \leftarrow p4(i), \mathrm{not}\, t4(i)$
$p5(i+1) \leftarrow p5(i)$

Conflicts:

$\leftarrow 2\ \{t2(i), t3(i), t5(i)\}$

Idling only at start:

$\leftarrow \mathit{idle}(i+1), \mathrm{not}\, \mathit{idle}(i)$
$\leftarrow \mathit{idle}(n-1)$

$\mathit{idle}(i) \leftarrow \mathrm{not}\, t1(i), \ldots, \mathrm{not}\, t5(i)$

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

[K. Heljanko and I. Niemelä. Bounded LTL Model Checking with Stable Models. Theory and Practice of Logic Programming, 3 (4&5): 519–550, 2003.]

Experiments—cont'd

LTL Model Checking Experiments

Problem	St n	St s	Int n	Int s	Bmc n	Bmc s	Bdd s	States
DP(6)	7	0.2	8	0.5	8	4.3	64.8	728
DP(8)	8	1.5	10	5.7	10	64.0	$>$1800	6560
DP(10)	9	25.9	12	140.1	12	1257.1	$>$1800	59048
DP(12)	10	889.4	14	$>$1800	14	$>$1800	$>$1800	531440

For instance for six philosophers:

$$\neg GF(f_5.up\ U\ (p_5.eat \wedge (f_3.up\ U\ (p_3.eat \wedge (f_1.up\ U\ p_1.eat)))))$$

`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
  [T. Jussila, K. Heljanko, and I. Niemelä. BMC via On-the-Fly Determinization. International Journal on Software Tools for Technology Transfer, 7(2), 89-101, 2005.]
- Linear size encoding in SAT
  [Latvala, Biere, Heljanko, Junttila; FMCAD'2004]
- Incrementality and Past LTL
  [Heljanko et al., CAV'2005]
  Implemented in `NuSMV` 2.4.0