Answer Set Programming and Configuration Knowledge Representation

Andreas Falkner
Siemens AG, Vienna, Austria
Corporate Technology, RTC BAM, Configuration Technologies
(andreas.a.falkner@siemens.com)
Contents

A practical overview of ASP
- SAT solving
- ASP modeling and solving
- Exercise 1: Simplified PC configuration

Product configuration with ASP
- OOASP
- Working example: Bicycle configuration
- Exercise 2: Check (consistency of a configuration)
- Exercise 3: Solve (i.e. complete a partial configuration)

Reconfiguration
- Knowledge evolution
- Exercise 4: Repair (an inconsistent or old configuration)
SAT = Propositional Satisfiability

Problem:
- Given a set $V$ of Boolean variables and a set $C$ of clauses over $V$ (i.e. a Boolean formula in conjunctive normal form, as clauses are disjunctions of – possible negated – variables)
- Is there a truth assignment for $V$ satisfying $C$?

Example:
- $(x_1 \lor \neg x_2 \lor \neg x_3) \land (x_1 \lor x_2 \lor x_4)$

Why is this useful for configuration?
- There are efficient SAT solvers available which can be used after transforming a given problem to SAT with millions of variables and constraints
- But be aware: SAT is proved NP-complete, as is 3-SAT (clauses have at most 3 vars)
The Power of SAT Solvers

From: Sabharwal, IBM Watson Research Center, 2011

- SAT encoding for bounded model checking problem from SATLIB

```
p cnf 51639 368352
-1 7 0
-1 6 0
-1 5 0
-1 -4 0
-1 3 0
-1 2 0
-1 -8 0
-9 15 0
-9 14 0
-9 13 0
-9 -12 0
-9 11 0
-9 10 0
-9 -16 0
-17 23 0
-17 22 0
```

i.e., ((not x₁) or x₇)
((not x₁) or x₆)
etc.

x₁, x₂, x₃, etc. are our Boolean variables
(to be set to True or False)

Should x₁ be set to False??
The Power of SAT Solvers

– 10 pages later:

\[
\begin{align*}
185 & \quad -9 \ 0 \\
185 & \quad -1 \ 0 \\
177 & \quad 169 \quad 161 \quad 153 \quad 145 \quad 137 \quad 129 \quad 121 \quad 113 \quad 105 \quad 97 \\
89 & \quad 81 \quad 73 \quad 65 \quad 57 \quad 49 \quad 41 \\
33 & \quad 25 \quad 17 \quad 9 \quad 1 \quad -185 \ 0 \\
186 & \quad -187 \ 0 \\
186 & \quad -188 \ 0 \\
\ldots
\end{align*}
\]

\[\text{i.e., } (x_{177} \text{ or } x_{169} \text{ or } x_{161} \text{ or } x_{153} \ldots \\
\quad x_{33} \text{ or } x_{25} \text{ or } x_{17} \text{ or } x_9 \text{ or } x_1 \text{ or } \neg x_{185})\]

\[\text{clauses / constraints are getting more interesting...}\]

\[\text{Note } x_1 \ldots\]
The Power of SAT Solvers

– Finally, 15,000 pages later

\[
\begin{align*}
-7 & 260 0 \\
7 & -260 0 \\
1072 & 1070 0 \\
-15 & -14 -13 -12 -11 -10 0 \\
-15 & -14 -13 -12 -11 10 0 \\
-15 & -14 -13 -12 11 -10 0 \\
-15 & -14 -13 -12 11 10 0 \\
-7 & -6 -5 -4 -3 -2 0 \\
-7 & -6 -5 -4 -3 2 0 \\
-7 & -6 -5 -4 3 -2 0 \\
-7 & -6 -5 -4 3 2 0 \\
185 & 0 \\
\end{align*}
\]

Search space of truth assignments: \(2^{50000} \approx 3.160699437 \cdot 10^{15051}\)

Current SAT solvers solve this instance in just a few seconds!
Simple Example: PC Configuration

Minimal fragment of PC model and adapted constraints:
- PC has a usage type (internet, multimedia, scientific)
- Its CPU has a clock rate (slow, medium, fast)
- Scientific usage requires fast clock rate
- Media usage is incompatible with slow clock rate

SAT representation as Boolean formula:
\[(\neg \text{ui} \land \neg \text{um} \land \neg \text{us}) \lor (\text{ui} \land \text{um} \land \neg \text{us}) \lor (\neg \text{ui} \land \neg \text{um} \land \text{us})\]
\[\land (\neg \text{cs} \land \neg \text{cm} \land \neg \text{cf}) \lor (\neg \text{cs} \land \text{cm} \land \neg \text{cf}) \lor (\neg \text{cs} \land \neg \text{cm} \land \text{cf})\]
\[\land (\neg \text{um} \rightarrow \text{cm} \lor \text{cf}) \land (\neg \text{us} \rightarrow \text{cf})\]

CNF:
\[(\text{ui} \lor \text{um} \lor \text{us}) \land (\neg \text{ui} \lor \neg \text{um}) \land (\neg \text{ui} \lor \neg \text{us}) \land (\neg \text{um} \lor \neg \text{us}) \land (\neg \text{cs} \lor \text{cm} \lor \text{cf}) \land (\neg \text{cs} \lor \neg \text{cm}) \land (\neg \text{cs} \lor \neg \text{cf}) \land (\neg \text{cm} \lor \neg \text{cf}) \land (\neg \text{um} \lor \text{cm} \lor \text{cs}) \land (\neg \text{us} \lor \text{cf})\]
Simple Example: PC Configuration

Minimal fragment of PC model and adapted constraints:
- PC has a usage type (internet, multimedia, scientific)
- Its CPU has a clockrate (slow, medium, fast)
- Scientific usage requires fast clockrate
- Media usage is incompatible with slow clockrate

ASP representation:
\[
\text{pc(internet)} \lor \text{pc(scientific)} \lor \text{pc(multimedia)}. \quad \% \text{GUESS}
\]
\[
\text{cpu(slow)} \lor \text{cpu(medium)} \lor \text{cpu(fast)}. 
\]
\[
\text{:- pc(scientific), not cpu(fast)}. \quad \% \text{CHECK}
\]
\[
\text{:- pc(multimedia), cpu(slow)}. 
\]

Solutions = answer sets:
Answer 1: pc(multimedia), cpu(fast)
Answer 2: pc(scientific), cpu(medium)
Answer 3: pc(scientific), cpu(fast)
Answer 4: pc(internet), cpu(slow)
Answer 5: pc(internet), cpu(medium)
Answer 6: pc(internet), cpu(fast)
Declarative Problem Solving

Algorithm = Logic + Control (Kowalski, 1979, CACM):

- What is the problem? Clear specification!
- How to solve? Powerful generic solver(s)!

- Answer Set Programming (Potassco)
ASP = Answer Set Programming

Standard specification language
- ASP-Core-2, since 2012
- Syntax: terms, atoms, literals, variables, function symbols, predicates, facts, rules, constraints, optimization, choice rules, aggregations, weak constraints
- Semantics: stable models, minimal models, negation as failure = default negation, non-monotonic

Prominent systems:
- smodels (Finland: Soininen, Niemelä)
- DLV (Vienna, Calabria: Gottlob, Eiter, Leone)
- Potassco (Potsdam: Schaub, Gebser): clingo = gringo + clasp

Applications in real-world (e.g. DLV, 2010):
- Shift planning in harbour Gioa-Tauro for 130 employees
- Daily shift: 25 sec, monthly shift: 490 sec
How to Use Clingo

Edit a program in a text editor, e.g. pc.lp

- **Facts**: e.g. `usage(internet). usage(multimedia).`
- **Choices (aggregates)**: e.g. `1 { pcUsage(U) : usage(U) } 1.``
- **Rules** consist of head and body (aggregates are allowed): e.g. `pcCPU(fast) :- pcUsage(scientific).``
- **Constraints** state what is **not** allowed, e.g. `:- pcCPU(slow), pcUsage(multimedia).``

Execute directly in a command shell, e.g.

- `clingo -n1 pc.lp` (for one answer set, `-n1` may be omitted)
- `clingo -n0 pc.lp` (for all answer sets)
- optimization directives (`#minimize`) will be used automatically
- to list union/intersect of answer sets: `--enum-mode=brave / cautious`
- to hand over constant (`#const`) values: `-c constname=constvalue`
- to show grounded version in readable format: `--text`
Coding Style for Clingo

from: Answer Set Solving in Practice
  – Gebser, Kaminski, Kaufmann, Schaub, 2012

For solving a problem (class) in ASP, provide
  – Data: facts describing an instance and
  – Program: a (uniform) encoding of solutions

Encodings are often structured by defining
  – Domain information (by deduction from facts)
  – Generator providing solution candidates (choice rules)
  – Propagation rules specifying properties of candidates (normal rules)
  – Testers eliminating invalid candidates (integrity constraints)
  – Display statements projecting answer sets (#show)
  – Optimization directives evaluating answer sets (#minimize/#maximize)
Exercise 1 (30 min)

(1) Change the PC configurator model (pc.lp)
   - A PC can have up to 4 CPUs
   - Add a constraint that requires all CPUs to have the same clockrate

(2) How many configurations are possible?
   - For 2, 3, 4 CPUs
   - With and without the additional constraint
Product Configuration seen Object-oriented

Specification of the product range

- Generic part-of structure (partonomy, aggregation): product consists of parts or modules with given properties and possible values
- Generic kind-of structure (taxonomy, generalization/specialization): parts appear as different variants or types
- Number of sub-parts not fixed (dependent on other decisions)
- Cross-tree dependencies

Some representations

- UML / OCL (Standard?)
- Product Variant Master (Hvam)
- Cardinality-based Feature Modeling (Czarnecki)
- CSL (Siemens CT)
- OOASP (Siemens CT)
Product Configuration with OOASP

Answer Set Programming and Configuration Knowledge Representation
Bikeshop UML Model (CSL)
Bikeshop Model (OOASP)

```prolog
ooasp_class("v1","BikeshopConfigObject").
ooasp_class("v1","Bicycle").
ooasp_subclass("v1","Bicycle","BikeshopConfigObject").
ooasp_attribute("v1","Bicycle","Bicycle_type","string").
ooasp_attribute_enum("v1","Bicycle","Bicycle_type","City").
ooasp_attribute_enum("v1","Bicycle","Bicycle_type","MTB").
ooasp_attribute_enum("v1","Bicycle","Bicycle_type","Racing").
ooasp_attribute("v1","Bicycle","Bicycle_hasLights","boolean").
ooasp_class("v1","Frame").
ooasp_subclass("v1","Frame","BikeshopConfigObject").
ooasp_attribute("v1","Frame","Frame_size","integer").
ooasp_attribute_minInclusive("v1","Frame","Frame_size",20).
ooasp_attribute_maxInclusive("v1","Frame","Frame_size",28).
% associations  (-1 means *, i.e. unrestricted)
ooasp_assoc("v1","Bicycle_frame","Bicycle",1,1,"Frame",1,1).
... (see bikeshop_kb.lp)
```
OOASP Specification of Models

\texttt{ooasp\_class(Id}_M, Id_C)\texttt{)}

\texttt{ooasp\_subclass(Id}_M, Id_C, Id_{SC})\texttt{)}

\texttt{ooasp\_assoc(Id}_M, Id_A, Id_{C_1}, Min_{C_1}, Max_{C_1}, Id_{C_2}, Min_{C_2}, Max_{C_2})\texttt{)}

\texttt{ooasp\_attribute(Id}_M, Id_C, Id_{AT}, \{"string", "integer", "boolean"\})

\texttt{ooasp\_attribute\_minInclusive(Id}_M, Id_C, Id_{AT}, MinV)\texttt{)}

\texttt{ooasp\_attribute\_maxInclusive(Id}_M, Id_C, Id_{AT}, MaxV)\texttt{)}

\texttt{ooasp\_attribute\_enum(Id}_M, Id_C, Id_{AT}, Val)\texttt{)}

\hspace{1cm} a class \( C \) is defined in a model \( M \)
\hspace{1cm} defines a subclass relation between a class \( C \) and a super class \( CS \) in a model \( M \)
\hspace{1cm} defines an association relation \( A \) between classes \( C_1 \) and \( C_2 \) with the given cardinalities, e.g. for every instance of the class \( C_1 \) at least \( Min_{C_2} \) and at most \( Max_{C_2} \) instances of the class \( C_2 \) must be associated
\hspace{1cm} an attribute \( AT \) of a class \( C \) is defined to have one of the three possible types
\hspace{1cm} provides an optional minimum value \( MinV \) for an integer attribute \( AT \)
\hspace{1cm} provides an optional maximum \( MaxV \) for an integer attribute \( AT \)
\hspace{1cm} defines a possible value \( Val \) for a string attribute \( AT \)

OOASP Specification of Instances

\[
\begin{align*}
\text{ooasp\_instantiation}(Id_M, Id_I) & \quad \text{defines an instantiation } I \text{ of a model } M \\
\text{ooasp\_isa}(Id_I, Id_C, Id_O) & \quad \text{declares that an object } O \text{ is an instance of the class } C \\
\text{ooasp\_associated}(Id_I, Id_A, & \quad \text{objects } O_1 \text{ and } O_2 \text{ are associated by the association relation } A \\
Id_{O_1}, Id_{O_2}) & \quad \text{assigns a value } Val \text{ to an attribute } AT \text{ of an object } O \\
\text{ooasp\_attribute\_value}(Id_I, Id_{AT}, & \quad \\
Id_O, Val) & \quad \\
\end{align*}
\]

Advantages of "generic" representation

- easier automated generation and validation
- explicit versions of model and instances
- easier upgrades and knowledge evolution
- reasoning over several KB versions
OOASP Specification of Constraints

Built-in integrity constraints

- For cardinality restrictions, typing, etc.
- e.g. ooasp_cv(I, mincardviolated(01, A)) :-
  {ooasp_associated(I, A, 01, 02): ooasp_isa(I, C2, 02)} C2MIN-1, C2MIN>0,
  ooasp_assoc(M, A, C1, C1MIN, C1MAX, C2, C2MIN, C2MAX),
  ooasp_instantiation(M, I),
  ooasp_isa(I, C1, 01).

- The rule for ooasp_cv(...) defines the constraint code
- Constraint violation forced during solving (by :- ooasp_cv(...))

Domain-specific constraints

- To be written by the knowledge engineer
- All ASP and OOASP functionality can be used
- Same procedure as for built-in constraints
- e.g. constraint wheelsMustHaveSameSize in bikeshop_kb.lp
OOASP Reasoning

Check correctness of the configuration (model checking)
- Exercise 2: Answer set consists of violated constraints (ooasp_cv)

Find the / a solution (model finding)
- Complete current partial configuration to a valid solution
- Exercise 3: Answer sets consist of the complete solution (bikeinfo)
- Not yet supported: Find the best solution according to a minimization or maximization criterion (optimization)

Reconcile an inconsistent instantiation (reconfiguration)
- Create a new version of the instantiation which is consistent
- Minimize the costs of the necessary actions (change, add, remove)
- Cause of inconsistency may be user requirement or knowledge evolution (new attributes, new constraints, etc.)
- Exercise 4: Answer sets consists of necessary actions and new values
Exercise 2 (15 min)

(1) Correct the OOASP representation of the Bikeshop
   - Compare `bikeshop_kb.lp` to the Bikeshop UML model
   - There is one error (Hint: have a look at the associations)

(2) Check the configuration i1 w.r.t. correct bikeshop_kb
   - Check: In a command shell, execute `clingo i1_check.lp`
   - How many constraints are violated? Built-in? Domain-specific?
   - Configure: Edit `bikeshop_i1.lp` to remove constraint violations
   - Repeat check/configure until `bikeshop_i1.lp` is a valid configuration

(3) Cross-check with solving (reconciliation)
   - Execute `clingo -n0 i1_repair.lp` on the original `bikeshop_i1.lp`
   - Compare the resulting (last) answer set with your changes
   - Are there any differences?
Exercise 3 (60 min)

(1) Solving with bikeshop_kb.lp
   - Execute `clingo -n0 i1_config.lp` to enumerate all solutions
   - How many? 4374? How many variants for City bikes? 1458?

(2) Change constraint wheelsMustHaveSameSize
   - Change to constraintWheelSize as specified on next slide
   - Execute clingo -n0 i1_config.lp again - How many variants? 486?

(3) Add constraintFrameType (see next slide)
   - Execute clingo -n0 i1_config.lp again - How many variants? 270?

(4) Add constraintBicycleLights (see next slide)
   - Execute clingo -n0 i1_config.lp again - How many variants? 189?

(5) Add constraint requiresLightSystem (see rule on next slide)
   - Execute clingo -n0 i1_config.lp again - How many variants? 99?
Bikeshop Constraints (CSL)

Wheel sizes must correspond to frame size

```java
class Wheel extends BikeshopConfigObject {
    constraint constraintWheelSize {
        assert (wheelSize == bicycle.frame.frameSize);
    }
}
```

Some frame types are incompatible with bicycle type

```java
class Frame extends BikeshopConfigObject {
    constraint constraintFrameType {
        var b: bikeshop.Bicycle = this.bicycle;
        assert ((b.type == bikeshop.BicycleType.City) ->
            ((this.type == bikeshop.FrameType.diamond) || (this.type == bikeshop.FrameType.stepthrough)));
        assert ((b.type == bikeshop.BicycleType.MTB) -> (this.type == bikeshop.FrameType.diamond));
        assert ((b.type == bikeshop.BicycleType.Racing) -> (this.type != bikeshop.FrameType.stepthrough));
    }
}
```

City bikes must have lights, mountain bikes must not

- If the Boolean attribute is set, a LightSystem instance must exist

```java
class Bicycle extends BikeshopConfigObject {
    constraint constraintBicycleLights {
        assert ((this.type == bikeshop.BicycleType.City) -> this.hasLights);
        assert ((this.type == bikeshop.BicycleType.MTB) -> !(this.hasLights));
    }
    rule ruleSetHasLights {
        this.hasLights = (this.lightSystem != null);
    }
```
Exercise 4 (15 min)

(1) Reconciliation
- Make sure that bikeshop_i1.lp contains your valid configuration (Exercise 2.2)
- Make sure that bikeshop_kb.lp contains your extended model (Exercise 3.2 to 3.5)
- Execute clingo -n0 i1_repair.lp to reconcile i1 to the new KB

(2) Analyse the results
- What did the system change?
- Why?
- Force the system to keep type "City"
   (Hint: add a fact in bikeshop_i1.lp)
- Is the result as you expected?
bikeinfo("i2","City","true","diamond",26,26,26,"battery")?
References


