

Using a Theorem Prover for Reasoning on Constraint Problems

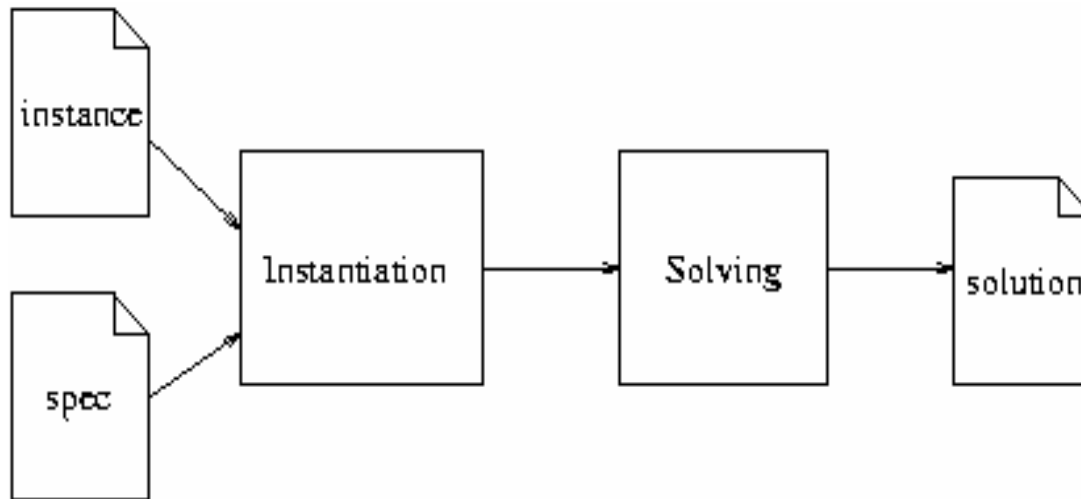
Marco Cadoli and Toni Mancini

Dipartimento di Informatica e Sistemistica
Università degli Studi di Roma “La Sapienza”
{cadoli|tmancini}@dis.uniroma1.it

Outline

- State-of-the-art systems for constraint modelling and programming
- Our goals
- Use of ATP technology for:
 - Detecting & breaking value symmetries
 - Detecting & handling functional dependencies
- Experiments
- Conclusions & future work

State-of-the-art systems architecture



- Clear separate problem spec and instances
- Advantages:
 - Decoupling of the specification from the solver
 - Focus on declarative aspects of specifications

Goals of our research

- Make a more complex reasoning on the spec, in order to:
 - Detect properties of the spec, interesting for the modeller (also for **verification** purposes)
 - Reformulate the spec to improve solver **efficiency**
- Reasoning: done on the problem model (spec), **independently on the instance**
- Undecidable in the general case

Note: spec level optimizations can be composed, after instantiation, with existing techniques

Why reasoning at the spec level

- **Verification:**
 - Problem specs can be viewed as **sw artefacts**. Reasoning helps the modeller during design
- **Optimization:**
 - Many properties amenable to optimizations derive from problem structure (hold for all instances).
 - Additional properties may arise from the instance
 - Current approaches try to infer *all* properties after instantiation (e.g., symmetries, dependencies)
 - But “structural” properties can be much less recognizable there, since problem structure has been destroyed!

Example: Reasoning at the spec level

- **Symmetry detection**

Graph 3-coloring (a problem):

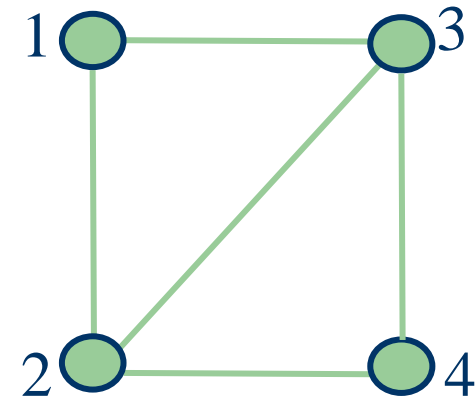
Is symmetric: color names can be exchanged

A graph (an instance):

May be symmetric: further symmetries

- **Functional dependencies**

Blue nodes can be computed from red and green ones (as complement)



In this work

- Specifications viewed as logical formulae
- Main question:
Is it possible to infer properties of specs by using Automated Theorem Proving technology?
- In this work we focus on:
 - Value symmetries
 - Functional dependencies

ESO specifications

In general, an ESO spec is a formula $\psi: \exists \mathbf{S} \phi(\mathbf{S}, \mathbf{R})$ where:

- \mathbf{S} is the set of **guessed predicates**
- \mathbf{R} is the **input schema** (set of relations)
- $\phi(\mathbf{S}, \mathbf{R})$ is a first-order formula on the relational vocabulary $\mathbf{S} \cup \mathbf{R} \cup \{=\}$

The **input instance** is given **as a relational DB** on the schema \mathbf{R}

Example: Graph 3-Col in ESO

- EDB schema: binary relation *edge*
- Herbrand domain: graph nodes

Choose 3 subsets of the graph nodes...

$$\exists RGB \quad \forall X \quad R(X) \vee G(X) \vee B(X) \quad \wedge$$

...disjoint...

$$\forall X \quad R(X) \rightarrow \neg G(X) \quad \wedge$$

$$\forall X \quad R(X) \rightarrow \neg B(X) \quad \wedge$$

$$\forall X \quad B(X) \rightarrow \neg G(X) \quad \wedge$$

s.t. they are a covering...

...and every edge links nodes with diff. colors

$$\forall XY \quad X \neq Y \wedge R(X) \wedge R(Y) \rightarrow \neg edge(X, Y) \quad \wedge$$

$$\forall XY \quad X \neq Y \wedge G(X) \wedge G(Y) \rightarrow \neg edge(X, Y) \quad \wedge$$

$$\forall XY \quad X \neq Y \wedge B(X) \wedge B(Y) \rightarrow \neg edge(X, Y),$$

Detecting and breaking symmetries



Why it is useful?

- **Efficiency:** It is well known that adding additional constraints that break symmetries can significantly reduce solving time
- **Verification:**
 - Does the model have symmetries?
(Does it really have an expected symmetry?)
 - Does a given formula correctly break a symmetry? (correctness & completeness)

Why at the specification level?

- To help user during modelling
- Easier to detect “structural” symmetries

Detecting value symmetries [Cadoli & M., 2003]

Given $\psi: \exists \mathbf{S} \phi(\mathbf{S}, \mathbf{R})$ (*Hyp: S_i all monadic*):

Definition: Unif. value transformation (UVT)

A mapping $\sigma: \mathbf{S} \rightarrow \mathbf{S}$ which is total and onto (i.e. a permutation)

Definition: Unif. value symmetry (UVS)

A u.v. transformation σ s.t.:

$$\forall \mathbf{S} \forall \mathbf{I} \phi(\mathbf{S}, \mathbf{I}) \leftrightarrow \phi^\sigma(\mathbf{S}, \mathbf{I})$$

with ϕ^σ defined as

$$\phi[S_1/\sigma(S_1), \dots, S_n/\sigma(S_n)]$$

Example: Graph 3-Col (cont.)

- Function σ s.t. $\sigma(R)=R$, $\sigma(G)=B$, $\sigma(B)=G$ is a *u.v. transformation* for the graph 3-Col spec
- Function σ is also a *u.v. symmetry* for it, since it is true that

$$\forall S \forall I \phi(S, I) \leftrightarrow \phi^\sigma(S, I)$$

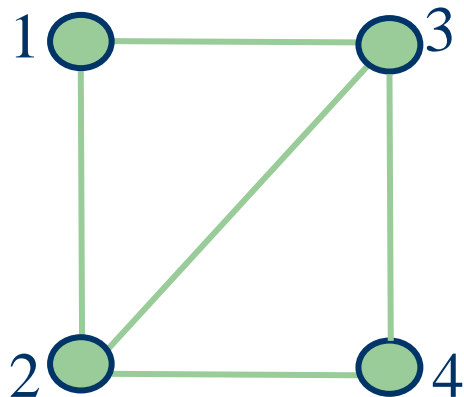
(i.e., for every input instance, solutions are mapped into solutions and vice versa)

Why monadic guessed predicates?

- *Monadic predicates* \Leftrightarrow domain values in CSPs
- *Binary predicates* handled by unfolding

Example: CSP u.v. transformation

Graph 3-Coloring on:



- $V = \{X_1, \dots, X_4\}$
- $D = \{D_1, \dots, D_4\}$, $D_i = \{R, G, B\}$ for each i
- $C = \dots$

The set of functions $\{\sigma_0, \sigma_1, \dots, \sigma_4\}$ s.t.:

- $\sigma_0 = Id,$
- $\sigma_1(R) = \dots = \sigma_4(R) = R$
- $\sigma_1(G) = \dots = \sigma_4(G) = B$
- $\sigma_1(B) = \dots = \sigma_4(B) = G$

is a u.v. transformation for this CSP
(cf., e.g., [Meseguer&Torras, 1999])

Some results [Cadoli & M., 2003]

Given:

- $\psi: \exists \mathbf{S} \phi(\mathbf{S}, \mathbf{R})$ (S_i all monadic) and
- u.v. transformation σ for ψ

Theorem: σ is a u.v. symmetry iff $\phi \equiv \phi^\sigma$

→ Can use a first-order theorem prover 😊

Theorem: Checking whether σ is a u.v. symmetry for ψ is **undecidable** 😞

Nonetheless, experimental analyses show that first-order theorem provers usually perform quite well in these cases

Experiments with ATP (OTTER)

Given:

- $\psi: \exists \mathbf{S} \phi(\mathbf{S}, \mathbf{R})$ (S_i all monadic) and
- u.v. transformation σ for ψ

We checked with a first-order thm prover (OTTER v. 3.3)

whether: $\phi \equiv \phi^\sigma$

Specification	u.v.transf.	Is symm?	Time (s)
3-Coloring	$R \leftrightarrow G$	Yes	0.27
Not-All-Eq SAT	$T \leftrightarrow F$	Yes	0.22
Not-All-Eq 3-SAT	$T \leftrightarrow F$	Yes	4.71
Social golfer (unfolded)	Swap 2 groups in the same week	Yes	0.96

What about non-UVSs?

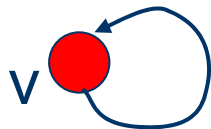
What if the given σ is not a UVS for ψ ?

- Finding a **counter-example** suffices
- Can use a **finite model finder**

Example: 3-Coloring with red self-loops

$$\psi: \exists \mathbf{S} \phi_{3\text{-Col}}(\mathbf{S}, \mathbf{R}) \wedge \forall \mathbf{X} \text{edge}(\mathbf{X}, \mathbf{X}) \rightarrow \mathbf{R}(\mathbf{X})$$

MACE v2.2 was able to show that $\sigma: R \leftrightarrow G$ is not a UVS in **< 1 sec**, by finding the following counter-example:



edge		R	G	B
From	To	v	-	-
v	v			

Breaking symmetries [Cadoli & M., 2003]

Goal: Given symmetry σ , find $\beta(\mathbf{S})$ such that

$$\psi: \exists \mathbf{S} \phi(\mathbf{S}, \mathbf{R}) \text{ (} S_i \text{ all monadic)}$$

can be safely transformed into:

$$\exists \mathbf{S} (\phi(\mathbf{S}, \mathbf{R}) \wedge \beta(\mathbf{S}))$$

$\beta(\mathbf{S})$ is called **symmetry-breaking formula**

Breaking symmetries: SB formulae

Given $\psi: \exists \mathbf{S} \phi(\mathbf{S}, \mathbf{R})$ (S_i monadic) and u.v. symmetry σ for ψ :

Definition: Symmetry-breaking formula

Formula $\beta(\mathbf{S})$ s.t.:

1. σ is no longer a symmetry for $\exists \mathbf{S} (\phi \wedge \beta(\mathbf{S}))$:

$$(\phi(\mathbf{S}, \mathbf{R}) \wedge \beta(\mathbf{S})) \not\equiv (\phi(\mathbf{S}, \mathbf{R}) \wedge \beta(\mathbf{S}))^\sigma$$

2. No solutions, except symmetric ones, are lost:

$$\phi(\mathbf{S}, \mathbf{R}) \models \bigvee_{\sigma \in \sigma^*} (\phi(\mathbf{S}, \mathbf{R}) \wedge \beta(\mathbf{S}))^\sigma$$

Can use a
finite model
finder!

Experiments: SB with ATP

Given:

- $\psi : \exists \mathbf{S} \phi(\mathbf{S}, \mathbf{R})$ (Si all monadic), UVS σ for ψ
- formula $\beta(\mathbf{S})$

Question: Is $\beta(\mathbf{S})$ a SB formula for ψ wrt σ ?

Spec	Symm	Formula $\beta(\mathbf{S})$	SB?	CPU time (sec)	
				Cond 1 (MACE)	Cond 2 (OTTER)
3-Col	$R \leftrightarrow G$	$R(v_1) \vee B(v_1)$	Yes	1.47	1.89
3-Col	$R \leftrightarrow G$	$\forall w G(w) \rightarrow$ $\exists v R(v) \wedge v \leq w$	Yes	2.67	?
SocGolf	swap	as above	Yes	0.04	?

2nd order SB formulae

Example: 3-Coloring, $\sigma: R \leftrightarrow G$,

$$\beta(R, G, B) : |R| \leq |G|$$

(ESO formula, guesses a total inj. function T from R to G)

- Conditions 1 & 2 become 2nd order (non-)equivalences. For $\beta(\mathbf{S})$ of the form $\exists \mathbf{T} \gamma(\mathbf{R}, \mathbf{G}, \mathbf{B})$
 1. $\exists \mathbf{T} (\phi(\mathbf{S}, \mathbf{R}) \wedge \gamma(\mathbf{S}, \mathbf{T})) \not\equiv \exists \mathbf{T} (\phi(\mathbf{S}, \mathbf{R}) \wedge \gamma(\mathbf{S}, \mathbf{T}))^\sigma$;
 2. $\phi(\mathbf{S}, \mathbf{R}) \models \bigvee_{\sigma \in \sigma^*} (\exists \mathbf{T} \gamma(\mathbf{S}, \mathbf{T}))^\sigma$.
- How to use ATP technology?
 - Use a 2nd order thm prover (under investigation)
 - Find 1st order conditions that imply 2nd order equiv

ESO SB formulae: Checking cond 1

Example: 3-Coloring, $\sigma: R \leftrightarrow G$, $\beta(R, G, B) : |R| \leq |G|$

In ESO: guess total inj. funct. T from R to G:

$$\beta(R, G, B) : \exists T \gamma(R, G, B, T)$$

Cond. 1: $\exists T (\phi(S, R) \wedge \gamma(S, T)) \not\equiv \exists T (\phi(S, R) \wedge \neg \gamma(S, T))^\sigma$

1. Find

$$\langle \overline{edge}, \overline{R}, \overline{G}, \overline{B}, \overline{T} \rangle \models \phi(\overline{edge}, \overline{R}, \overline{G}, \overline{B}) \wedge \gamma(\overline{R}, \overline{G}, \overline{B}, \overline{T}) \wedge \neg \gamma(\overline{R}, \overline{G}, \overline{B}, \overline{T})^\sigma$$

A graph plus a coloring s.t.

$$|\overline{R}| < |\overline{G}|, \text{ e.g.: } v \bullet$$

...then, this interpretation

shows the non-equivalence. Overall computation: < 0.3 sec

Finite model
finder

Theorem
prover

2. If $\gamma(\overline{R}, \overline{G}, \overline{B}, \overline{T})^\sigma \equiv \perp$

i.e., $|\overline{G}| \leq |\overline{R}|$
Contraddiction!

Detecting functional dependencies among guessed predicates



Why it is useful?

- **Efficency:** A search strategy can be (automatically) synthesized in order to exploit such a dependence (e.g., by **avoiding branches on dependent predicates**)
- **Verification:**
 - **Dependencies by design:** Do they really hold?
 - **Unexpected dependencies:** Do they exist?
 - Do they evidence a **model bug**?
 - Do they evidence a **bad design choice** (e.g., redundancy)?

Why at the specification level?

- They are intrinsically “structural” properties!

Dependencies: existing approaches

Only at the instance level, e.g.:

- Detecting “defined” vars in SAT instances due to:
 - Clausification process
 - Equivalence clauses


and using modified versions of the DP procedure

- SAT solvers that avoid branches on these vars, e.g., **EQSATZ**

Dependencies: our approach

Detect dependent guessed predicates at the spec level

Definition: Given spec $\psi : \exists \mathbf{SP} \phi(\mathbf{S}, \mathbf{P}, \mathbf{R})$,
set \mathbf{P} functionally depends on \mathbf{S} if, for each instance I
and each pair of interpretations M and N of (\mathbf{S}, \mathbf{P}) , s.t:

1. $M \neq N$,
 2. $M, I \models \phi$, and
 3. $N, I \models \phi$
- 
- then: $M|_{\mathbf{S}} \neq N|_{\mathbf{S}}$

i.e., no two distinct solutions to ϕ exist that coincide on extension for predicates in \mathbf{P} .

Dependencies (cont)

Theorem: Given spec $\psi : \exists SP \phi(S, P, R)$,
set P functionally depends on S iff the following
formula is valid

$$[\phi(S, P, R) \wedge \phi(S', P', R) \wedge \neg(SP \equiv S'P')] \rightarrow \neg(S \equiv S')$$

→ Can use a first-order theorem prover 😊

Theorem: Checking whether set P functionally depends on
 S is **undecidable** 😞

Nonetheless, experimental analyses show that first-order
theorem provers usually perform quite well in these cases

Experiments with ATP (OTTER)

Given $\psi: \exists \mathbf{SP} \phi(\mathbf{S}, \mathbf{P}, \mathbf{R})$

We checked with a first-order thm prover whether

$$[\phi(\mathbf{S}, \mathbf{P}, \mathbf{R}) \wedge \phi(\mathbf{S}', \mathbf{P}', \mathbf{R}) \wedge \neg(\mathbf{SP} \equiv \mathbf{S}'\mathbf{P}')] \rightarrow \neg(\mathbf{S} \equiv \mathbf{S}')$$

was a valid formula.

Specification	\mathbf{S}	\mathbf{P}	Time (s)
3-Coloring	R,G	B	0.25
NAE-3-SAT	T	F	0.38
Sailco	regBoat, extraBoat	inventory	0.21
Protein Folding	Move	X,Y,Hits	569.55 (*)

(*) Simplified version, with Hits omitted.

Automated synthesis of a search proc

Question: Is explicitly avoiding branches on dependent predicates effective in practice?

Experiments with Ilog OPL Studio:

1. **2D HP-Protein folding** on benchmark instances
2. **Blocks world** on random instances
(results not shown, but similar to those of 2D HP-Protein folding)

2D HP-Protein folding with OPL

Length	Time no search proc	Time with search proc	Saving %
30	–	6	99.9
36	3078	13	85.1
37	3386	17	93.6
45	–	99	>97.3
48	–	156	>95.7
50	–	215	>94.0
60	–	128	>96.4
64	–	233	>93.5
102	1242	1007	18.9
123	914	913	0.0
136	761	760	0.0

Conclusions

- We have proven that reasoning at the spec level is effective for:
 - Detecting and breaking “structural” symmetries
 - Detecting and handling functional dependencies in the problem model

Positive consequences both for the modelling (verification) and solving stages (efficiency)

- Reasoning automatable by using ATP technology
- The wide availability of problem specs constitutes a **brand new set of benchmarks for ATP systems**

Current work

- Experiments with other thm provers and on diff. specs
- Investigate the applicability of the approach for **other forms of reasoning**, e.g.
 - Symmetries on variables

For symmetry reasons, can be reduced to "r1 < r2", i.e. "ordered"

Example: N-Queens

```
int+ N = ...; range Row = 1..N; range Col = 1..N;
var Col Queen[Row];
solve {
  forall (r1, r2 in Row : r1 <> r2) {
    Queen[r1] <> Queen[r2];
    Queen[r1]+r1 <> Queen[r2]+r2;
    Queen[r1]-r1 <> Queen[r2]-r2;
  }
};
```

Proved by OTTER in < 1 sec

Thank you! 😊

Questions?

