

# Propagators and Solvers for the Algebra of Modular Systems

Bart Bogaerts<sup>1,2</sup>, David Mitchell<sup>3</sup>, Eugenia Ternovska<sup>3</sup>

1 Aalto University, Finland, bart.bogaerts@aalto.fi

2 KU Leuven, Belgium, bart.bogaerts@cs.kuleuven.be

3 Simon Fraser University, Burnaby, Canada, {ter,mitchell}@cs.sfu.ca

October 17th, 2016

# Overview

- ▶ Modular Systems
- ▶ Propagators and Solvers
- ▶ Explanations and Learning
- ▶ Modular Patterns
- ▶ Conclusion

# Modular Systems

- ▶ Modular expressions:

$$E ::= \perp \mid M_i \mid E \times E \mid \neg E \mid \pi_\delta E \mid \sigma_{Q \equiv R} E.$$

- ▶ An atomic module is interpreted as a class of structures.
- ▶ Compound modules also represent classes of structures.

# Modular Systems: Conventions

- ▶ We are concerned with the *model expansion task* for modular systems.

## Definition

The *model expansion task* for modular systems is: given a (compound) module  $E$  and a partial structure  $\mathcal{I}$  with finite domain, find a structure  $I$  (or: find all structures  $I$ ) such that  $I \geq_p \mathcal{I}$  and  $I \models E$  (if one such exists).

# Modular Systems: Conventions

- ▶ We are concerned with the *model expansion task* for modular systems.

## Definition

The *model expansion task* for modular systems is: given a (compound) module  $E$  and a partial structure  $\mathcal{I}$  with finite domain, find a structure  $I$  (or: find all structures  $I$ ) such that  $I \geq_p \mathcal{I}$  and  $I \models E$  (if one such exists).

- ▶ We assume a finite domain is given and fixed.
- ▶ Without loss of generality, we assume all vocabularies are *relational*
- ▶ A *domain atom* is an expression of the form  $P(\bar{d})$  ( $P$  predicate,  $\bar{d}$  domain elements)
- ▶ A *four-valued*  $\Sigma$ -structure  $\mathcal{I}$  is an assignment  $P(\bar{d})^{\mathcal{I}}$  of a four-valued truth value (true **t**, false **f**, unknown **u** or inconsistent **i**) to each domain atom over  $\Sigma$ .

# Propagators and Solvers for Modular Systems: Goals

- ▶ Modular systems: integration on the semantic level (how to combine *information* from different domains).
- ▶ This paper: integration on the solving level (how to combine *solving techniques* from different domains).
- ▶ We will assume pieces of software (propagators/solvers) are given for atomic modules, and research how to obtain software for combined modules.

# Propagators

## Definition

A *propagator* is a mapping  $P$  from partial structures to partial structures such that the following hold:

- ▶  $P$  is  $\leq_p$ -monotone: whenever  $\mathcal{I} \geq_p \mathcal{I}'$ , also  $P(\mathcal{I}) \geq_p P(\mathcal{I}')$ .
- ▶  $P$  is information-preserving:  $P(\mathcal{I}) \geq_p \mathcal{I}$  for each  $\mathcal{I}$ .

## Definition

Given a module  $E$ , a propagator  $P$  is an *E-propagator* if on two-valued structures, it coincides with  $E$ , i.e., whenever  $\mathcal{I}$  is two-valued,  $P(\mathcal{I}) = \mathcal{I}$  if  $\mathcal{I} \in E$ , and  $P(\mathcal{I})$  is inconsistent otherwise.

# Propagators

## Lemma

*Let  $P$  be an  $E$ -propagator. If  $I$  is a model of  $E$  and  $I \geq_p \mathcal{I}$ , then also  $I \geq_p P(\mathcal{I})$ .*



# Propagators: Example

## Example

Modern ASP solvers typically contain (at least) two propagators. One, which we call  $P_{UP}^{\mathcal{P}}$ , performs unit propagation on the completion of the program  $\mathcal{P}$ . The other, which we call  $P_{UFS}^{\mathcal{P}}$  performs unfounded set propagation; that is: it maps a partial structure  $\mathcal{I}$  to  $\mathcal{I} \cup \{\neg p \mid p \in IUFS(\mathcal{P}, \mathcal{I})\}$ , where  $IUFS(\mathcal{P}, \mathcal{I})$  is the largest unfounded set of  $\mathcal{P}$  with respect to  $\mathcal{I}$  [1].

It is easy to see that these two propagators are information-preserving and  $\leq_p$ -monotone.

# Propagators: Checkers

- ▶ How can we create propagators for modules? Back-up plan: checkers.

## Definition

If  $E$  is a module, the  $E$ -checker is the propagator  $P_{check}^E$  defined by:<sup>1</sup>

$$P_{check}^E : \mathcal{I} \mapsto \begin{cases} \mathcal{I} & \text{if } \mathcal{I} \text{ is consistent but not two-valued} \\ \mathcal{I} & \text{if } \mathcal{I} \text{ is two-valued and } \mathcal{I} \models E \\ \tilde{\mathcal{I}} & \text{otherwise} \end{cases}$$

- ▶ In the paper: how to create checkers for compound modules (straightforward).

---

<sup>1</sup>Here,  $\tilde{\mathcal{I}}$  denotes the most precise (inconsistent) structure.

# Solvers

## Definition

Let  $E$  be a module. An  $E$ -solver is a procedure that takes as input a four-valued structure  $\mathcal{I}$  and whose output is the set  $\mathcal{S}$  of all two-valued structures  $I$  with  $I \models E$  and  $I \geq_p \mathcal{I}$ .

# Propagators and Solvers

Simple way to create a solver  $S_p^P$  from a propagator  $P$ :

- ▶ State is a partial structure
- ▶ Depth-first search (choices on domain atoms)
- ▶ Before each choice: apply the propagator

# Propagators for Compound Modules

## Proposition

Let  $P$  be an  $E$ -propagator,  $P'$  an  $E'$ -propagator and  $\delta$  a sub-vocabulary of  $\tau$ . We define the following operations:

- ▶  $P \times P' : \mathcal{I} \mapsto \text{lub}_{\leq_p} \{P(\mathcal{I}), P'(\mathcal{I})\}$ .
- ▶  $\pi_\delta P : \mathcal{I} \mapsto$

$$\begin{cases} \mathfrak{J} & \text{if } \mathcal{I} \text{ is inconsistent} \\ \mathfrak{J} & \text{if } \mathcal{I} \text{ is two-valued on } \delta \text{ and } S_p^P(\mathcal{I}|_\delta) = \emptyset \\ \text{lub}_{\leq_p} (P(\mathcal{I}|_\delta)|_\delta, \mathcal{I}|_{\tau \setminus \delta}) & \text{otherwise.} \end{cases}$$

- ▶  $\sigma_{Q \equiv R} P : \mathcal{I} \mapsto (P(\mathcal{I}))[Q : L, R : L]$  where  $L = \text{lub}_{\leq_p} (Q^{P(\mathcal{I})}, R^{P(\mathcal{I})})$ .

It then holds that  $P \times P'$  is an  $E \times E'$ -propagator,  $\pi_\delta P$  is a  $\pi_\delta E$  propagator and  $\sigma_{Q \equiv R}$  is a  $\sigma_{Q \equiv R} E$ -propagator.

## Explanations (informal)

- ▶ Idea: propagators not only change the partial interpretation
- ▶ They also explain *why* they do it
- ▶ Explanation is itself a propagator again
- ▶ The explanation is “simpler” (in a certain sense)
- ▶ “Simplest” propagators do not need to explain themselves

## Explanations (informal)

- ▶ Idea: propagators not only change the partial interpretation
- ▶ They also explain *why* they do it
- ▶ Explanation is itself a propagator again
- ▶ The explanation is “simpler” (in a certain sense)
- ▶ “Simplest” propagators do not need to explain themselves
- ▶ Generalizes lazy clause generation (constraint programming)
- ▶ Generalizes cutting plane generation (linear programming)

# Explanations (formal)

## Definition

An *explaining propagator* is tuple  $(P, C)$  where  $P$  is a propagator and  $C$  maps each partial structure either to UNEXPLAINED (notation  $\diamond$ ) or to an explaining propagator  $C(\mathcal{I}) = (P', C')$  such that the following hold:

- ▶ (explains propagation)  $P(\mathcal{I}) \leq_p P'(\mathcal{I})$ .
- ▶ (soundness):  $module(P') \subseteq module(P)$



# Explanations

## Example

- ▶ Example from constraint programming
- ▶ Constraint of the form  $T \Leftrightarrow C > D$ .
- ▶ Partial interpretation with  $T = \mathbf{t}, D = 3$ .
- ▶ Propagate  $C > 3$ .
- ▶ Explanation:  $T \wedge D \geq 3 \Rightarrow C > 3$
- ▶ Simpler in the sense: each atom contains at most one integer variable

# Explaining Propagators

- ▶ For some of the operations, we define how to combine explanations,
- ▶ We give a general search algorithm that uses such explaining propagators to build solvers

# Conflict-Driven Learning

- ▶ We provide a generalisation of CDCL for explaining propagators
- ▶ Abstract conditions on “simplest” propagators that ensure a generalisation of resolution is possible
- ▶ Work in progress

# Modular Patterns

- ▶ For some modular expressions: compound propagators are suboptimal
- ▶ Better propagators can be created.

# Modular Patterns

## Example

$$-(-M_1 \times -M_2)$$

If  $P_1$  is a  $M_1$  propagator and  $P_2$  is an  $M_2$  propagator,

$$-(-P_1 \times -P_2)$$

is simply a checker. A more precise propagator is:

$$P_1 + P_2 : \mathcal{I} \mapsto \text{glb}_{\leq_p} (P_1(\mathcal{I}), P_2(\mathcal{I})).$$

# Modular Patterns

- ▶ More examples in the paper:
- ▶ Propagator for selection (with non-atomary selection formula)
- ▶ Propagator for negation of projection

# Conclusion

We study:

- ▶ Solvers and propagators for Modular Systems
- ▶ Explanation mechanism
- ▶ Conflict-driven learning
- ▶ Modular patterns

Why?

- ▶ Generalisation of many existing solving techniques
- ▶ Can serve to prove correctness of future techniques
- ▶ Integration of different paradigms