Defeasible Logic Programming in Satisfiability Modulo CHR

Francesco Santini

Dipartimento di Matematica e Informatica,

Università di Perugia, Italy

AIxIA 2015

Motivations

- Merging **Argumentation-based Logic Programming** and **Satisfiability-modulo Theories** together

- Using **Satisfiability-modulo Constraint Handling Rules**
  - To encode Argumentation-based Logic Programming
- Towards **Argumentation-based Constraint Logic Programming**
  - (X,Y) : −X > Y, B(X), C(Y): where X > Y is a constraint, and A(X, Y), B(X), C(Y) are literals, as in regular Logic Programming

- Some (current and future) goals
  1. Have a unifying framework where to solve different ALP proposals
  2. Design propagators (in CHR) on top of different built-in solver (SAT, bounds, linear arithmetic)
  3. Deal with weights (belief degree)
  4. Argument-based reasoning in an efficient way

CHR in a nutshell

```
reflexivity   @ X leq X <=> true.
antisymmetry  @ X leq Y, Y leq X <=> X = Y.
transitivity  @ X leq Y, Y leq Z ==> X leq Z.
idempotence   @ X leq Y \ X leq Y <=> true.
```

- Execution of a CHR program starts with an initial constraint store. The program then proceeds by matching rules against the store and applying them, until either no more rules match (success)

- Given a query $A \leq B, B \leq C, C \leq A$

Current constraints	Rule applicable to constraints	Conclusion from rule application
`A leq B, B leq C, C leq A`	transitivity	`A leq C`
`A leq B, B leq C, C leq A, A leq C`	antisymmetry	`A = C`
`A leq B, B leq A, A = C`	antisymmetry	`A = B`
`A = B, A = C`	none	

Argumentation-based LP

- Several proposals in Logic Programming, almost all defined on the concept of **weak** and **strict** rules
  - Logic for Defeasible Reasoning (LDR) by Donald Nute ’88
  - Rules with priorities by Prakken and Sartor ‘97
  - Defeasible Logic Programming by Garcia and Simari ‘04

- *Facts* are ground literals representing atomic information or its negation.
- *Strict rules* represent non-defeasible rules, and they are represented as $L_0 \leftarrow L_1, \ldots, L_n$.
- *Defeasible rules* represent tentative information, in the form of rules like $L_0 \prec L_1, \ldots, L_n$.

All birds fly except when they don't

Defeasible logic programming

$$\Pi \begin{cases} night. & switch\_on(a). \\ \sim day \leftarrow night. & switch\_on(b). \\ \sim dark(Y) \leftarrow illuminated(X). & switch\_on(c). \\ sunday. & \sim electricity(b). \\ deadline. & \sim electricity(c). \\ & emergency\_lights(c). \end{cases}$$

$$\Delta \begin{cases} light\_on(X) \prec switch\_on(X). \\ \sim lights\_on(X) \prec \sim electricity(X). \\ lights\_on(X) \prec \sim electricity(X), emergency\_lights(X). \\ dark(X) \prec \sim day. \\ illuminated(X) \prec lights\_on(X), \sim day. \\ working\_at(X) \prec illuminated(X). \\ \sim working\_at(X) \prec sunday. \\ working\_at(X) \prec sunday, deadline. \end{cases}$$

In SMCHR

```
/* Strict rules */
night(x) ⟹ not day(x);
illuminated(x) ⟹ not dark(x);

/* Defeasible rules */
switchOn(x) ⟹ lightsOn(x);
not electricity(x) ⟹ not lightsOn(x);
not electricity(x) ∧ emergencyLights(x) ⟹ lightsOn(x);
not day(x) ⟹ dark(x);
not day(x) ∧ lightsOn(x) ⟹ illuminated(x);
illuminated(x) ⟹ workingAt(x);
sunday(x) ⟹ not workingAt(x);
sunday(x) ∧ deadline(x) ⟹ workingAt(x);
```

Facts as constraints:

switchOn(a), switchOn(b), switchOn(c), not electricity(b), not electricity(c), emergencyLights(c), night(a), night(b), night(c), sunday(a), sunday(b), sunday(c), deadline(a), deadline(b), deadline(c).

In SMCHR

```
/* Strict rules */
night(x) ==> not day(x);
illuminated(x) ==> not dark(x);

/* Defeasible rules */
switchOn(x) ==> lightsOn(x);
not electricity(x) ==> not lightsOn(x);
not electricity(x) ∧ emergencyLights(x) ==> lightsOn(x);
not day(x) ==> dark(x);
not day(x) ∧ lightsOn(x) ==> illuminated(x);
illuminated(x) ==> workingAt(x);
sunday(x) ==> not workingAt(x);
sunday(x) ∧ deadline(x) ==> workingAt(x);
```

$\Pi \cup \Delta$ |~ Q

A query: Q = *illuminated(a) ∧ switchOn(a)*

The answer is UNKNOWN (i.e., SAT using sat) and 3 new constraints:

lightsOn(a), not dark(a), workingAt(a)

```
/* Defeasible rules */
not electricity(x) ==> not lightsOn(x) ∧ defeasibleNotLightsOn(x);

/*1*/ defeasibleNotLightsOn(x) ∧ lightsOn(x) ==> strictLightsOn(x);
```

In the paper, how to mark defeasible information

Possibilistic DeLP

- Defeasible Logic Programming (P-DeLP) is an extension of DeLP in which
  - rules are attached with weights, belonging to the real unit interval [0..1] (here [0..100]
  - weights express the relative <u>**belief or preference strength** </u>of arguments. Each fact p_i is associated with a certainty value that expresses how much the relative fuzzy-statement is believed in terms of necessity measures.
  - Weights are aggregated in accordance to $(p_1 \wedge \cdots \wedge p_k \rightarrow q, \alpha)$ iff $(p_1, \beta_1), \ldots, (p_k, \beta_k)$ and $(q, \min(\alpha, \beta_1, \ldots, \beta_k))$

P-DeLP example

```
type sw1(num); type sw2(num); type sw3(num); type pumpClog(num); type
    pumpFuel(num); type pumpOil(num); type oilOk(num); type fuelOk(num);
     type engineOk(num); type heat(num); type lowSpeed(num);

/* Strict */
pumpClog(x) ⟹ not fuelOk(x);

/* Defeasible */
sw1(x) ⟹ x $<= 60 | pumpFuel(x);
sw1(x) ⟹ x $> 60 | pumpFuel(60);
pumpFuel(x) ⟹ x $<= 30 | fuelOk(x);
pumpFuel(x) ⟹ x $> 30 | fuelOk(30);
sw2(x) ⟹ x $<= 80 | pumpOil(x);
sw2(x) ⟹ x $> 80 | pumpOil(80);
pumpOil(x) ⟹ x $<= 80 | oilOk(x);
pumpOil(x) ⟹ x $> 80 | oilOk(80);
oilOk(x) ∧ fuelOk(y) ⟹ x $<= y ∧ x $<= 30 | engineOk(x);
oilOk(x) ∧ fuelOk(y) ⟹ y $<= x ∧ y $<= 30 | engineOk(y);
oilOk(x) ∧ fuelOk(y) ⟹ 30 $<= x ∧ 30 $<= y | engineOk(30);
heat(x) ⟹ x $<= 95 | not engineOk(x);
heat(x) ⟹ x $> 95 | not engineOk(95);
heat(x) ⟹ x $<= 90 | not oilOk(x);
heat(x) ⟹ x $> 90 | not oilOk(x);
lowSpeed(x) ∧ pumpFuel(y) ⟹ x $<= y ∧ x $<= 70 | pumpClog(x);
lowSpeed(x) ∧ pumpFuel(y) ⟹ y $<= x ∧ y $<= 70 | pumpClog(y);
lowSpeed(x) ∧ pumpFuel(y) ⟹ 70 $<= x ∧ 70 $<= y | pumpClog(70);
sw2(x) ⟹ x $<= 80 | lowSpeed(x);
sw2(x) ⟹ x $> 80 | lowSpeed(80);
sw3(x) ∧ sw2(y) ⟹ x $<= y ∧ x $<= 80 | not lowSpeed(x);
sw3(x) ∧ sw2(y) ⟹ y $<= x ∧ y $<= 80 | not lowSpeed(y);
sw3(x) ∧ sw2(y) ⟹ 80 $<= x ∧ 80 $<= y | not lowSpeed(80);
sw3(x) ⟹ x $<= 60 | fuelOk(x);
sw3(x) ⟹ x $> 60 | fuelOk(80);
```

Query: sw1 (100) ∧ sw2 (100)

Store: engineOk(30) ∧ fuelOk(30) ∧ lowSpeed(80) ∧ not fuelOk (60) ∧ oilOk(80) ∧ pumpClog (60) ∧ pumpFuel (60) ∧ pumpOil (80) ∧ sw1 (100) ∧ sw2 (100)

Thank you for your time!

Acquario Room (G-GF)

Contacts:

francesco.santini@dmi.unipg.it