Incremental Computation of Warranted Arguments in Dynamic Defeasible Argumentation: The Rule Addition Case

Gianvincenzo Alfano[1], Sergio Greco[1], Francesco Parisi[1], Gerardo Ignacio Simari[2], Guillermo Ricardo Simari[2]

[1] Department of Informatics, Modeling, Electronics and System Engineering
University of Calabria, Italy
{g.alfano, greco, fparisi}@dimes.unical.it
[2] Departamento de Ciencias e Ing. de la Computación
Universidad Nacional del Sur (UNS)
Instituto de Ciencias e Ing. de la Computación (ICIC UNS−CONICET) Argentina
{gis,grs}@cs.uns.edu.ar

33rd ACM/SIGAPP Symposium On Applied Computing
Track on Knowledge Representation and Reasoning

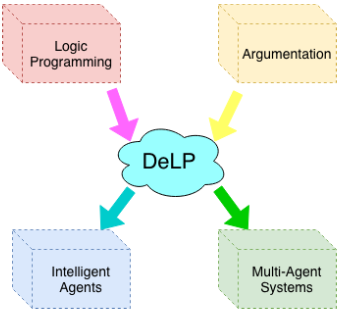
April 9-13, 2018
Pau, France

Dynamic Structured Argumentation

- A general way for representing arguments and relationships (defeats) between them
- It allows representing dialogues, making decisions, and handling inconsistency and uncertainty
- Several kinds of Argumentation Frameworks (e.g. Abstract Argumentation, **Structured Argumentation**)
- A well-known formalism for structured argumentation is DeLP: **De**feasible **L**ogic **P**rogramming
- Argumentation frameworks are often dynamic (change over time) as a consequence of the fact that argumentation is inherently dynamic (change mind/opinion, new available knowledge)
- We devise an incremental technique for computing conclusions in structured argumentation frameworks (avoiding wasted effort due to recomputation from scratch)

DeLP

- We focus on **De**feasible **L**ogic **P**rogramming, a formalism that combines results of Logic Programming and Defeasible Argumentation.
- DeLP is a knowledge representation language, where defeasible and non-defeasible rules can be expressed.
- The language has two different negations: classical negation, used for representing contradictory knowledge and negation as failure, used for representing incomplete information.
- A defeasible argumentation inference mechanism for warranting the conclusions that are entailed.



Outline

A DeLP Program

DeLP considers two kinds of program rules: **defeasible** rules to represent tentative information, and **strict** rules used to represent strict knowledge.

Example

Consider the DeLP-program $\mathcal{P}_1 = (\Pi_1, \Delta_1)$, where:

$$\Pi_1 = \{w,\ t,\ z,\ (p \leftarrow t)\}$$

$$\Delta_1 = \left\{ \begin{array}{llll} (\sim a \prec y), & (y \prec x), & (x \prec z), & (\sim x \prec p, t) \\ (a \prec w, y), & (\sim w \prec t), & (y \prec p), & (x \prec p) \end{array} \right\}$$

The (non-contradictory) set of literals that can be derived from Π_1 is $\{w, t, z, p\}$. However, both a and $\sim a$ can be derived from $\mathcal{P}_1$ using the following sets of rules and facts:

$$(a \prec w, y), (y \prec p), (p \leftarrow t), (t) \;\text{ and }\; (\sim a \prec y), (y \prec p), (p \leftarrow t), (t),$$

respectively.

Argument

Given a DeLP program $\mathcal{P} = (\Pi, \Delta)$ and a literal α, we say that $\langle \mathcal{A}, \alpha \rangle$ is an argument for α if $\mathcal{A}$ is a set of defeasible rules of Δ such that:

(i) there is a derivation for α from $\Pi \cup \mathcal{A}$,

(ii) the set $\Pi \cup \mathcal{A}$ is not contradictory, and

(iii) $\mathcal{A}$ is minimal (i.e., there is no proper subset $\mathcal{A}'$ of $\mathcal{A}$ satisfying both (i) and (ii)).

Example

Given $\mathcal{P}_1$, we have the following arguments (among others):

- $\langle \mathcal{A}_1, \sim a \rangle = \langle \{(\sim a \prec y),\ (y \prec x),\ (x \prec z)\}, \sim a \rangle$
- $\langle \mathcal{A}_2, \sim a \rangle = \langle \{(\sim a \prec y),\ (y \prec p)\}, \sim a \rangle$
- $\langle \mathcal{A}_3, a \rangle = \langle \{(a \prec w, y),\ (y \prec p), \}, a \rangle$
- $\langle \mathcal{A}_4, \sim x \rangle = \langle \{(\sim x \prec t, p)\}, \sim x \rangle$
- $\langle \mathcal{A}_5, x \rangle = \langle \{(x \prec p)\}, x \rangle$

Argument

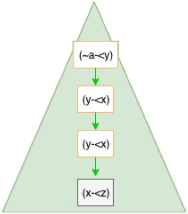
Given a DeLP program $\mathcal{P} = (\Pi, \Delta)$ and a literal α, we say that $\langle \mathcal{A}, \alpha \rangle$ is an argument for α if $\mathcal{A}$ is a set of defeasible rules of Δ such that:

(i) there is a derivation for α from $\Pi \cup \mathcal{A}$,
(ii) the set $\Pi \cup \mathcal{A}$ is not contradictory, and
(iii) $\mathcal{A}$ is minimal (i.e., there is no proper subset $\mathcal{A}'$ of $\mathcal{A}$ satisfying both (i) and (ii)).

Example (An argument for $\sim a$ built from the previous program)

Given $\mathcal{P}_1$, we have the following arguments (among others):

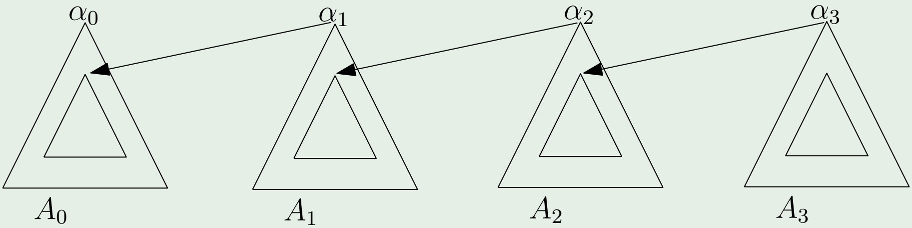
- $\langle \mathcal{A}_1, \sim a \rangle = \langle \{(\sim a \prec y), (y \prec x), (x \prec z)\}, \sim a \rangle$



Argumentation Line

- A sequence of arguments obtained from a DeLP program, where each element of the sequence is a **defeater** of its predecessor.

Example (Argumentation Line)



Example (An argumentation line from the previous program)

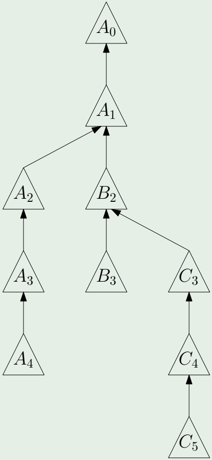
Given $\mathcal{P}$, an argumentation line is the following:

$$[\mathcal{A}_1, \mathcal{A}_3]$$

Dialectical Process

- Given an argument *A* for a literal *L*, the dialectical tree contains all acceptable argumentation lines that start with that argument.
- It allows to determine the status for a given argument.

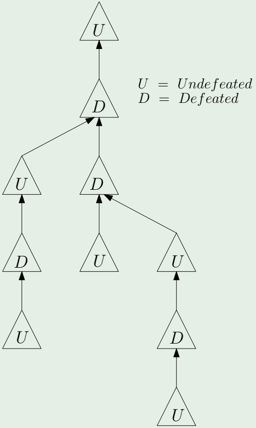
Example (Dialectical Tree)



Dialectical Process: Marking Procedure

- All leaves are marked as **Undefeated**.
- An argument in the tree is marked as **Defeated** if and only if it has at least a child marked as **Undefeated**.

Example (Dialectical Tree)



Status of literals

The status of literals allow us to determine the conclusions we can draw from a DeLP-program.
$S_{\mathcal{P}} : \mathit{Lit} \rightarrow \{\text{IN}, \text{OUT}, \text{UNDECIDED}, \text{UNKNOWN}\}$ assigning a *status* to each literal w.r.t. $\mathcal{P}$ as follows:

- $S_{\mathcal{P}}(\alpha) = \text{IN}$ if there exists a (marked) dialectical tree whose root α is *Undefeated*
- $S_{\mathcal{P}}(\alpha) = \text{OUT}$ if $S_{\mathcal{P}}(\sim\alpha) = \text{IN}$
- $S_{\mathcal{P}}(\alpha) = \text{UNDECIDED}$ if neither $S_{\mathcal{P}}(\alpha) = \text{IN}$ nor $S_{\mathcal{P}}(\alpha) = \text{OUT}$
- $S_{\mathcal{P}}(\alpha) = \text{UNKNOWN}$ if $\alpha \notin \mathit{Lit}_{\mathcal{P}}$, i.e., α is not in the language of the program

Example (Arguments from the previous program)

Given $\mathcal{P}$, then $S_{\mathcal{P}}(t) = \text{IN}$, $S_{\mathcal{P}}(\sim w) = \text{OUT}$, $S_{\mathcal{P}}(a) = \text{UNDECIDED}$.

Outline

Adding a rule to a DeLP program

An update consists of modifying a DeLP-program $\mathcal{P}$ into a new DeLP-program $\mathcal{P}^{'}$ by adding a strict or a defeasible rule. In particular, $\mathcal{P}'$ is as follows:

- If r is a **strict** rule, then if $(\Pi \cup r)$ is not contradictory, then $\mathcal{P}' = ((\Pi \cup r), \Delta)$ otherwise $\mathcal{P}' = \mathcal{P}$ (i.e., the update has no effect if it would yield a contradictory program).
- If r is a **defeasible** rule, then $\mathcal{P}' = (\Pi, (\Delta \cup r))$, that is, adding a defeasible rule is always permitted.

Example (Perform $r = (a \prec x)$ on $\mathcal{P}_1$)

The updated DeLP-program $\mathcal{P}_1' = (\Pi_1', \Delta_1')$, is as follows:

$$\Pi_1' = \Pi_1 = \{w,\ t,\ z,\ (p \leftarrow t)\}$$

$$\Delta_1' = \Delta_1 = \left\{ \begin{array}{llll} (\sim a \prec y), & (y \prec x), & (x \prec z), & (\sim x \prec p, t) \\ (a \prec w, y), & (\sim w \prec t), & (y \prec p), & (x \prec p) \end{array} \right\} \cup \{(a \prec x)\}$$

Question

After performing an update the conclusion that can be derived may change: the status of a is IN (was UNDECIDED) after performing the update $r = (y \leftarrow t)$ in our example.

Should we recompute the status of literals from scratch?

Overview of our incremental approach

Two main steps.

1) First, we check if the update is *irrelevant* (the status of all literals are preserved).

   In such a case we simply return the initial status $S_{\mathcal{P}}$.

2) To efficiently deal with *relevant* updates, we identify the subset of literals whose status needs to be recomputed after performing an update, and only recompute their status.

Irrelevant defeasible-rule update

Adding a rule whose head consists of a literal that already appears as a fact does not affect the status of any other literal of the program.

Proposition Let $\mathcal{P}$ be a DeLP-program, and $r = \alpha_0 \prec \alpha_1, \ldots, \alpha_n$ an update for $\mathcal{P}$. If $S_{\mathcal{P}}(\alpha_0) = \text{IN}$ then r is irrelevant for $\mathcal{P}$ (i.e., $S_{\mathcal{P}'} = S_{\mathcal{P}}$).

However, many updates are *relevant*.

Example (Relevant update)

Consider the DeLP-program $\mathcal{P}_1$ from our running example, where we have that $S_{\mathcal{P}}(y) = S_{\mathcal{P}}(t) = \text{IN}$. If the rule addition update is $r = (y \leftarrow t)$, then $S_{\mathcal{P}'}(a)$ becomes IN, though it was UNDECIDED before performing the update.

Hyper-graph for a DeLP-Program

Given a program $\mathcal{P}$, $G(\mathcal{P}) = \langle N, H \rangle$ is defined as follows:

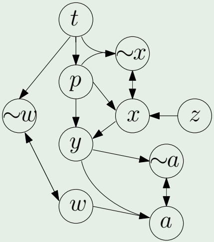
- If there is a strict derivation in Π for literal α, then $\alpha \in N$;
- For each strict rule $\alpha_0 \leftarrow \alpha_1, \ldots, \alpha_n$ (resp., defeasible rule $\alpha_0 \prec \alpha_1, \ldots, \alpha_n$) such that $\alpha_1, \ldots, \alpha_n \in N$, then $\alpha_0 \in N$ and $(\{\alpha_1, \ldots, \alpha_n\}, \alpha_0) \in H$;
- For each pair of nodes in N representing complementary literals α and $\sim\alpha$, both $(\{\alpha\}, \sim\alpha) \in H$ and $(\{\sim\alpha\}, \alpha) \in H$.

Example (Hyper-graph $G(\mathcal{P})$ for $\mathcal{P}$)

Consider the DeLP-program $\mathcal{P}_1 = (\Pi_1, \Delta_1)$, where:

$$\Pi_1 = \{w,\ t,\ z,\ (p \leftarrow t)\}$$

$$\Delta_1 = \left\{ \begin{array}{llll} (\sim a \prec y), & (y \prec x), & (x \prec z), & (\sim x \prec p, t) \\ (a \prec w, y), & (\sim w \prec t), & (y \prec p), & (x \prec p) \end{array} \right\}$$

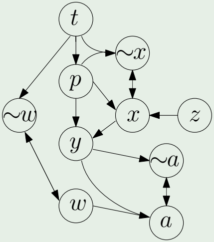


Preserved Literals

We say that a node y is *reachable* from a set X of nodes if there exists a hyper-path from X to y. We use $Reach_{G(\mathcal{P})}(X)$ to denote the set of all nodes that are reachable from X in $G(\mathcal{P})$.

Example

$Reach_{G(\mathcal{P})}(\{y\}) = \{y, a, \sim a\}$



Theorem (Preserved literals)

Given a DeLP program $\mathcal{P}$ and an update r, a literal α is preserved (i.e., $S_{\mathcal{P}}(\alpha) = S_{\mathcal{P}'}(\alpha)$) if $\alpha \notin Reach_{G(\mathcal{P})}(\{head(r)\})$.

Incremental Algorithm

Algorithm Dynamic DeLP-Solver

Input: DeLP-program $\mathcal{P}$, Initial status $S_{\mathcal{P}}$, Update r.
Output: Status $S_{\mathcal{P}'}$ of the updated program $\mathcal{P}'$.

```
if the update is irrelevant then
    return S_P;
if head(r) is a fresh literal then
    return S_P ∪ {(head(r), DELP-SOLVER(P', head(r)))}
Let G(P) = ⟨N, H⟩;// Build Hyper-graph
Let R = Reach_G(P)({head(r)}); // Literals not preserved
Let R̄ = {α ∈ (N \ R)};// Preserved literals
for α in R̄ do
    S_P'(α) ← S_P(α)// Copy the status of preserved literals
for α in R do
    S_P'(α) ← DELP-SOLVER(P', α)// Recompute the status of literals
return S_P'.
```

Outline

Dataset & Metodology

Datasets:
We randomly generated a dataset of 30 DeLP-programs, each of them having 200 positive and 20 negative literals, 8 facts, a number of strict rules in $\{5, 15, \ldots, 35\}$, and a number d of defeasible rules in $\{100, 120, \ldots, 200\}$, where each (strict or defeasible) rule has a number of literals in the body in $\{1, 2, 3\}$.

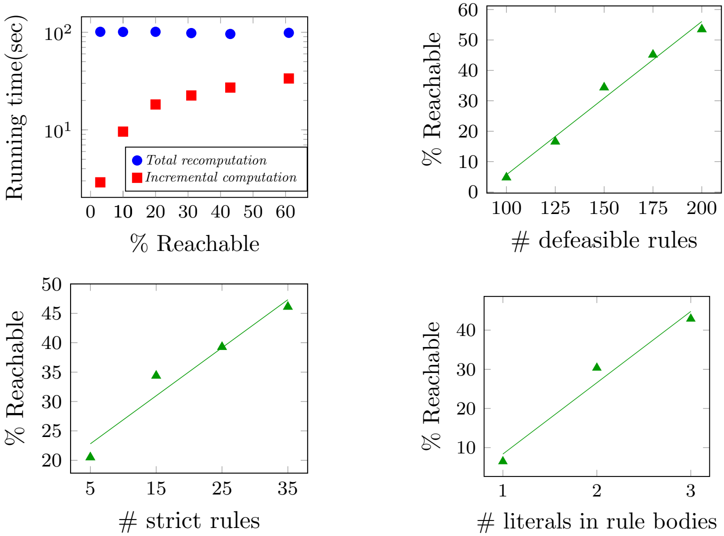
Methodology
For each DeLP-program $\mathcal{P}$ in the dataset, we compared the average running time of Algorithm 1 with that of the approach from scratch, which computes the status in the updated program by directly calling the DeLP-Solver for each literal of $\mathcal{P}$.

Dataset & Metodology

Datasets:
We randomly generated a dataset of 30 DeLP-programs, each of them having 200 positive and 20 negative literals, 8 facts, a number of strict rules in $\{5, 15, \ldots, 35\}$, and a number *d* of defeasible rules in $\{100, 120, \ldots, 200\}$, where each (strict or defeasible) rule has a number of literals in the body in $\{1, 2, 3\}$.

Methodology
For each DeLP-program $\mathcal{P}$ in the dataset, we compared the average running time of Algorithm 1 with that of the approach from scratch, which computes the status in the updated program by directly calling the DeLP-Solver for each literal of $\mathcal{P}$.

Experimental Results



Results

1) We compared our technique with the computation from scratch.
2) Our incremental algorithm outperforms the computation from scratch.
3) Our technique is sensitive to the percentage of literals reachable from the update: the higher it is the lower the benefits are.
4) A study to determine which parameter impacts on reachability is reported.

Outline

* We have taken the first steps in tackling the problem of avoiding wasted effort when determining the warrant status of literals in a DeLP program after that a (defeasible or strict) rule is added

* Our incremental approach outperforms the computation from scratch (especially if the average number of literals reachable from an update is less than 30%).

FW1) Further developing these techniques, as well as developing similar ones for rule deletion, fact addition and deletion, and the more general case of simultaneously adding or deleting a *set* of rules and facts.

FW2) Investigating how our technique can be also extended to cope with other structured argumentation frameworks.

Thank you!

... any ~~question~~ argument?