Panorama de Constraint Answer Set Programming

Bryan Garreau, Martín Diéguez, Eric Monfroy, Igor Stéphan

▶ **To cite this version:**

Bryan Garreau, Martín Diéguez, Eric Monfroy, Igor Stéphan. Panorama de Constraint Answer Set Programming. 20èmes Rencontres des Jeunes Chercheurs en Intelligence Artificielle, Jun 2022, Saint-Etienne, France. hal-03765417

HAL Id: hal-03765417
https://hal.science/hal-03765417v1

Submitted on 31 Aug 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Panorama de Constraint Answer Set Programming

B. Garreau[1], M. Dieguez Lodeiro[1], E. Monfroy[1], I. Stéphan[1]

[1] Université d'Angers, LERIA

{bryan.garreau, martin.dieguezlodeiro, eric.monfroy, igor.stephan}@univ-angers.fr

Résumé

Constraint Answer Set Programming (CASP) est un paradigme fusionnant Answer Set Programming (ASP) et la programmation par contraintes. CASP a gagné en popularité ces dernières années et beaucoup de techniques de résolutions ont émergé. Dans ce court article nous allons vous présenter quelques une des méthodes de résolution les plus renommées et nous allons voir les avantages et désavantages de celles-ci avant d'aborder les futures pistes de recherches que nous aimerions explorer.

Mots-clés

Constraint Answer Set Programming, Answer Set Programming, Constraint Programming, Knowlege Representation, NonMonotonic Reasoning

1 Introduction

Le paradigme *Constraint Answer Set Programming* (CASP) consiste en un langage déclaratif à la fois riche et simple pour modéliser des problèmes complexes et des mécanismes pour les résoudre. Ce nouveau paradigme est issu de la fusion de deux paradigmes d'intelligence artificielle, *Answer Set Programming* et *Constraint Programming* dont les origines remontent aux travaux de Mellarkod et al. [10]. Cette fusion permet de tirer partie des avantages des deux paradigmes pour obtenir un paradigme plus déclaratif et des solveurs plus performants.

Answer Set Programming (ASP) [5] est un langage de programmation utilisé en représentation des connaissances, en raisonnement non monotone mais aussi pour résoudre des problèmes combinatoires difficiles. Les solveurs ASP acceptent des programmes avec des variables qui doivent être substituées par des symboles propositionnels avant la phase de résolution. Le processus qui consiste à remplacer ces variables s'appelle le *grounding*. Bien qu'il existe des approches qui combinent la phase de *grounding* avec la phase de résolution [9], les solveurs les plus utilisés le font séparément [8].

Tandis que les approches avec *grounding* sont devenues très efficaces ces dernières années, les programmes propositionnels obtenus à l'issue de cette phase peuvent être très volumineux à cause de l'explosion combinatoire des substitutions, particulièrement lorsque les variables s'étendent sur les domaines $\mathbb{N}$, $\mathbb{R}$, etc... Le *grounding* est souvent un frein au processus de résolution. Dans le but d'améliorer et de raffiner le *grounding*, des techniques de programmation par contraintes peuvent être appliquées lors du *grounding* [10] ce qui permet d'éviter la génération de clauses inutiles. Afin d'illustrer le problème du *grounding*, considérons l'exemple suivant :

Exemple 1 *Considérons un circuit électrique composé de deux appareils : un interrupteur et une lampe. L'interrupteur peut être* allumé *(*`switchOn`*) ou* éteint *(*`swicthOff`*) de manière non-déterministe. L'état de la lampe (*on*/* off*) est directement déduit de l'état de l'interrupteur. Le prédicat* `light` *représente la présence de lumière dans l'environnement d'après l'état de l'interrupteur et l'heure de la journée. La proposition* `night` *représente la partie de la journée ou il n'y a pas de lumière naturelle (entre 22h00 et 7h00) et la proposition* `sleep` *est une conséquence directe de* `switchOff` *et* `night`*.*

Un programme ASP de cet exemple est présenté dans la Liste 1. Le programme *ground* proposé par le grounder `Gringo` contient 40 règles (en extension ou intension).

Liste 1 – Exemple de programme ASP

```
time(0..23).
1 {ctime(X) : time(X)} 1.
switchOn :- not switchOff .
switchOff :- not switchOn .
light :- switchOn.
light :- not night.
night :- X<7, ctime(X).
night :- X>=22, ctime(X).
sleep :- switchOff,night.
```

Constraint Programming (CP) [2] est un autre paradigme de programmation qui permet de modéliser des problèmes sous la forme de triplets (X, D, C) où X est l'ensemble des variables, D représente les domaines associés aux variables et C est l'ensemble des contraintes qui représentent les relations entre les variables. La particularité de CP est l'utilisation d'algorithmes performants de propagation des contraintes permettant de réduire considérablement l'espace de recherche à explorer lors de la résolution d'un problème. CP profite de plus d'un catalogue de contraintes globales qui permettent d'utiliser des algorithmes encore plus efficaces sur des problèmes précis. Les contraintes globales sont un atout majeur de CP puisqu'elles permettent

de rendre un programme plus concis et plus efficace.

Le principal problème des solveurs ASP actuellement est la phase de *grounding* qui peut être lente et produit des modèles qui sont inutiles. Ainsi ASP est souvent peu performant sur les problèmes comprenant un grand nombre de données. L'un des intérêts de CASP est d'apporter des techniques de filtrage et de propagation de contraintes pour optimiser cette phase de *grounding* et rendre possible la résolution de problèmes ASP contraints. C'est notamment le cas de problèmes industriels d'ordonnancement [1][3].

Dans ce papier nous allons vous présenter deux des principales méthodes de résolution de problèmes CASP ainsi que leurs avantages et inconvénients.

2 Approches pour CASP

Parmi les différents solveurs CASP, le plus populaire est Clingcon 3 [4]. Ce solveur n'effectue pas la substitution des variables utilisées dans les contraintes et les traite pendant la phase de résolution. Cette simplification du *grounding* permet de réduire le temps de calcul mais également d'économiser de la mémoire. La phase de résolution bénéficie donc d'algorithmes de CP pour traiter plus efficacement les contraintes. Toutefois, la propagation ne s'applique pas aux contraintes natives à ASP mais seulement aux contraintes sur des variables à domaines finis.

Par exemple, le problème dans la Liste 1 peut être représenté dans le langage de Clingcon 3 par le programme en Liste 2. Dans le premier programme il faut utiliser une règle de choix pour représenter le temps actuel alors que dans le deuxième exemple une variable `x` avec un domaine fini représente le temps actuel. Le *grounding* en est simplifié et ne produit que 8 règles, ce qui est beaucoup moins que les 40 initiales.

Liste 2 – Exemple de programme CASP

```
&dom{0..23} = x.
switchOn :- not switchOff .
switchOff :- not switchOn .
light :- switchOn.
light :- not night.
night :- &sum{x} < 7.
night :- &sum{x} >= 22.
sleep :- switchOff,night.
```

Il est également possible d'écrire des programmes CASP contenant des contraintes globales ce qui les rend plus concis et plus lisibles. De plus, cela permet d'utiliser des algorithmes adaptés pour rendre le traitement de ces contraintes plus efficient. Cependant, cette méthode de résolution, bien qu'efficace, fait une distinction entre les contraintes issue de CP et les contraintes ASP ce qui limite le traitement des variables des contraintes et la coopération entre les méthodes de résolution.

D'autres approches visent à traduire des programmes CASP en problème de *Satisfiability Modulo Theories* (SMT) [12]. Les solveurs SMT traitent des problèmes de satisfiabilité qui contiennent des formules logiques plus complexes comprenant des entiers, des réels ou des structures de données. Ces formules plus complexes appartiennent à des théories et sont traitées par des solveurs de théories pendant la résolution. Les solveurs de SMT, devenus très performants, permettent de résoudre des problèmes CASP de manière efficace. De plus, SMT est flexible et il est possible de rajouter des théories pour traiter des contraintes plus efficacement.

Plusieurs approches devenues récemment populaires ont essayé de donner une sémantique logique aux programmes CASP. Basées sur l'*equilibrium logic* [11], Cabalar et al [6] l'ont étendu pour traiter les contraintes et les agrégats. De plus, Eiter et al [7] combinent l'*equilibrium logic* avec la *weighted logic* pour représenter des contraintes algébriques. Toutes ces approches permettent de traiter les contraintes d'un point de vue logique.

3 Conclusions et ligne de travail

La résolution ASP est une approche de *Generate and Test* alors que la résolution CP utilise une approche *Constraint and Generate*. Cette différence entre les méthodes de résolution rend leur fusion prometteuse mais complexe. CASP est un paradigme assez récent et plusieurs méthodes de résolution ont été proposées. Cependant, la plupart des solveurs sont construits autour de solveurs ASP ou traduisent les programmes CASP dans d'autres langages (en problème SMT par exemple).

L'utilisation de CASP est principalement motivée par les limitations des solveurs ASP actuels tel que le *grounding*. Il est néanmoins possible d'étendre ASP pour traiter par exemples des nombres flottants qui ne sont pas traités nativement par ASP. Cette augmentation du langage permettrait de rendre ASP plus déclaratif qu'auparavant.

Dans le futur il pourrait être utile de s'intéresser à une coopération plus forte entre les solveurs ASP et de CP de façon à réellement les fusionner, et ce de manière plus transparente afin de mieux tirer parti des deux approches.

Références

[1] Dirk Abels, Julian Jordi, Max Ostrowski, Torsten Schaub, Ambra Toletti, and Philipp Wanko. Train scheduling with hybrid ASP. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 3–17. Springer, 2019.

[2] Krzysztof Apt. *Principles of constraint programming*. Cambridge university press, 2003.

[3] Marcello Balduccini. Industrial-size scheduling with ASP + CP. In *International conference on logic programming and nonmonotonic reasoning*, pages 284–296. Springer, 2011.

[4] Mutsunori Banbara, Benjamin Kaufmann, Max Ostrowski, and Torsten Schaub. Clingcon : The next generation. *TPLP*, 17(4) :408–461, 2017.

[5] G. Brewka, T. Eiter, and M. Truszczyński. Answer set programming at a glance. *Communications of the ACM*, 54(12) :92–103, 2011.

[6] Pedro Cabalar, Roland Kaminski, Max Ostrowski, and Torsten Schaub. An ASP semantics for default reasoning with constraints. In Subbarao Kambhampati, editor, *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 1015–1021. IJCAI/AAAI Press, 2016.

[7] Thomas Eiter and Rafael Kiesel. ASP($\mathcal{AC}$) : Answer Set Programming with Algebraic Constraints. *Theory Pract. Log. Program.*, 20(6) :895–910, 2020.

[8] Roland Kaminski, Javier Romero, Torsten Schaub, and Philipp Wanko. How to build your own ASP-based system ?! *CoRR*, abs/2008.06692, 2020.

[9] Claire Lefèvre, Christopher Béatrix, Igor Stéphan, and Laurent Garcia. Asperix, a first-order forward chaining approach for answer set computing. *Theory and Practice of Logic Programming*, 17(3) :266–310, 2017.

[10] Veena S. Mellarkod, Michael Gelfond, and Yuanlin Zhang. Integrating answer set programming and constraint logic programming. *Ann. Math. Artif. Intell.*, 53(1-4) :251–287, 2008.

[11] D. Pearce. A New Logical Characterisation of Stable Models and Answer Sets. In *Proc. of Non-Monotonic Extensions of Logic Programming (NMELP'96)*, pages 57–70, Bad Honnef, Germany, 1996.

[12] Benjamin Susman and Yuliya Lierler. SMT-based constraint answer set solver EZSMT (system description). In *Technical Communications of the 32nd International Conference on Logic Programming (ICLP 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.