Scuola Superiore
dell'Università degli Studi di Udine

Classe Scientifico-Economica

Colloquio di fine anno

AUTOMATED REASONING METHODS IN HYBRID SYSTEMS

Relatore:
Prof. Agostino Dovier

Allieva:
Sara Biavaschi

Anno Accademico 2016-2017

Contents

Introduction **2**

1 Background **3**
1.1 Answer Set Programming . . . 3
1.2 Logic programs with constraint atoms . . . 5

2 EZCSP hybrid system **9**
2.1 EZCSP language . . . 9
2.1.1 Syntax . . . 9
2.1.2 Semantics . . . 10
2.1.3 General EZ programs . . . 11
2.2 System Architecture . . . 12
2.3 EZCSP solver . . . 14
2.4 Integration schemas . . . 21
2.4.1 Black-box integration . . . 21
2.4.2 Grey-box integration . . . 21
2.4.3 Clear-box integration . . . 22

3 Application Domains **23**
3.1 Blending Problem . . . 23
3.2 Scheduling Tasks . . . 26

Bibliography **28**

Introduction

Answer set programming (ASP) is a declarative language tailored towards modeling and solving combinatorial optimization problems. It extends computational methods of SAT using ideas from knowledge representation, logic programming, and nonmonotonic reasoning. ASP proved to be useful for finding solutions to a variety of programming tasks, ranging from building decision support systems for the Space Shuttle (see [2]) to developing reasoning tools in biology (see [5]).

However, ASP is not directly suitable for modeling problems containing variables ranging over large domains. The process of grounding, i.e. translating a program containing variables in its propositional equivalent, is often the bottleneck of the solving algorithm. To overcome this problem researches have spent significant efforts in the development of systems that integrate ASP with Constraint Logic Programming (CLP), enriching ASP with the powerful support for numerical computations of CLP. This new area, called *Constraint Answer Set Programming*, has already demonstrated promising results, such as the development of CASP solvers CLINGCON ([7]), ACSOLVER ([10]), EZCSP ([2]) and EZSMT ([12]).

We begin with a review of ASP and CASP formalisms, then, in Chapter 2, we focus our attention on the EZCSP system, introducing its language and solver. We prove formal claims about its solving algorithm by presenting it in a graph-based framework, as pioneered in [11] with regard to backtrack search procedures for SAT and Satisfiability Modulo Theories. This approach will allow us to easily highlight key differences between various configuration of the EZCSP system and in general of hybrid systems. Indeed these systems have different methods to integrate ASP and CSP solvers, and there is no single choice of integration schema that achieves best performance in all cases (see [1]).

Finally, in Chapter 3, we propose our encodings, in the EZCSP language, of the Job Scheduling Problem and of the Blending Problem, two classical problems in operations research that will allow us to highlight the advantages of hybrid systems over ASP.

Chapter 1

Background

1.1 Answer Set Programming

We begin with a review of the syntax and semantics of ASP. Let Σ be a signature containing a set of constants, a set of function and a set of predicate symbols. Terms and atoms are formed as usual in the first order logic. A *literal* is either an atom or its negation.

Definition 1.1.1. A *rule* over Σ is a pair (a_0, Γ), where a_0 is an atom or $\bot$ and $\Gamma = (\Gamma_1, \Gamma_2, \Gamma_3)$ is a triplet of sets of atoms.

If $\Gamma_1 = \{a_1, \ldots, a_l\}$, $\Gamma_1 = \{a_{l+1}, \ldots, a_m\}$, $\Gamma_3 = \{a_{m+1}, \ldots, a_n\}$ we see the rule (a_0, Γ) as a statement of the form

$$a_0 \leftarrow a_1, \ldots, a_l, \text{not}\, a_{l+1}, \ldots, \text{not}\, a_m, \text{not not}\, a_{m+1}, \ldots, \text{not not}\, a_n \tag{1.1}$$

neglecting the order of elements in the right hand side of the arrow. The set of these elements is called the *body* of the rule, denoted by B, while a_0 is called the *head.* By B^{pos} we denote the positive part of the body of a rule 1.1, i.e. $B^{pos} = \Gamma_1$. When $a_0 = \bot$, we omit $\bot$ from the notation and call such rule *denial.* When the body is empty, we omit $\leftarrow$ from the notation and call such rule *fact.*

Intuitively, a rule states that a reasoner who believes $\{a_1, \ldots, a_l\}$ and has no reason to believe $\{a_{l+1}, \ldots, a_m\}$ and no reason not to believe $\{a_{m+1}, \ldots, a_n\}$ has to believe a_0. A rule can also be seen as an implication in first order logic in which *not* is replaced by the classical negation $\neg$ and the body is seen as the conjunction of its elements. Thus we often interpret 1.1 as a clause of the form

$$a_0 \vee \neg a_1 \vee \cdots \vee \neg a_l \vee a_{l+1} \vee \cdots \vee a_m \vee \neg a_{m+1} \vee \cdots \vee \neg a_n \tag{1.2}$$

We say that a set X of atoms satisfies clause 1.2 if $a_0 \in X$, or there exists $i \in \{1, \ldots, l, m+1, \ldots, n\}$ such that $a_i \notin X$, or there exists $i \in \{l+1, \ldots, m\}$ such that $a_i \in X$.

Definition 1.1.2. A *regular (logic) program* is a pair (Σ, Π), where Σ is a signature and Π is a finite set of rules over Σ.

Given a program Π, by $At(\Pi)$ we denote the set of all atoms that occur in the program, by $Lit(\Pi)$ the set of all literals over $At(\Pi)$ and by $(\Pi)^{cl}$ the set of clauses that correspond to the rules of Π. Answer sets of a program Π are intuitively subsets of $At(\Pi)$ that satisfy $(\Pi)^{cl}$ without assuming unnecessary atoms. In order to formalize this intuitive idea, we introduce the following.

Definition 1.1.3. A set M of literals is *consistent* if no two complementary literals, a and $\neg a$, belong to M. It is *complete* over a set of atoms σ if for any atoms a in σ, $a \in M$ or $\neg a \in M$.

It is easy to see how a set of atoms X over some set of atoms σ can be identified with a complete and consistent set of literals over σ:

$$\{a|a \in X\} \cup \{\neg a|a \in \sigma \setminus X\}$$

Vice versa given a set M of literals over σ we can define a set of atoms:

$$M^+ = \{a|a \in M \cap \sigma\}$$

Definition 1.1.4. The reduct of a regular program Π with respect to set X of atoms over $At(\Pi)$, denoted by Π^X, is obtained from Π deleting

- Each rule such that X does not satisfy its body.
- All expressions of the form "$\text{not}\, a$" or "$\text{not}\,\text{not}\, a$" from the body of the remaining rules.

Definition 1.1.5. A set X of atoms over $At(\Pi)$ is an answer set of the program Π if X satisfy $(\Pi^X)^{cl}$ and is subset minimal among the sets of atoms satisfying $(\Pi^X)^{cl}$.

We now present two ways to characterize answer sets of a program.

Theorem 1.1.6. *For a program Π, a set Γ of denials, and a consistent and complete set M of literals over $At(\Pi)$, M^+ is an answer set of $\Pi \cup \Gamma$ if and only if M^+ is an answer set of Π and M is a model of Γ^{cl}.*

Proof. We need to show that M is an answer set for $\Pi^{M^+} \cup \Gamma^{M^+}$ iff it is an answer set for Π^{M^+} and is a model of Γ^{cl}.

Left-to-right: Assume that M^+ is subset minimal among the sets of atoms satisfying $(\Pi^{M^+} \cup \Gamma^{M^+})^{cl}$. Then M^+ satisfies both $(\Pi^{M^+})^{cl}$ and $(\Gamma^{M^+})^{cl}$. The second condition means that M is a model of Γ^{cl}. It remains to check the minimality of M^+. Let N^+ be a subset of M^+ that satisfies $(\Pi^{M^+})^{cl}$. Since M^+ satisfies $(\Gamma^{M^+})^{cl}$, also its subset N^+ satisfies $(\Gamma^{M^+})^{cl}$ because Γ^{M^+} is a set of denials with positive bodies. Since M^+ is minimal among the sets satisfying $(\Pi^{M^+} \cup \Gamma^{M^+})^{cl}$, it follows that $N^+ = M^+$.

Right-to-left: Assume that M^+ is an answer set for Π^{M^+} and is a model of Γ^{cl}. The second condition means that M^+ satisfies $(\Gamma^{M^+})^{cl}$. Consequently, M^+ satisfies both $(\Pi^{M^+})^{cl}$ and $(\Gamma^{M^+})^{cl}$. It remains to check the minimality of M^+. Let N^+ be a subset of M^+ that satisfies $(\Pi^{M^+} \cup \Gamma^{M^+})^{cl}$. Then, in particular, N^+ satisfies $(\Pi^{M^+})^{cl}$. Since M^+ is minimal among such sets, it follows that $N^+ = M^+$. $\square$

By $Bodies(\Pi, a)$ we denote the set of the bodies of all rules of program Π with the head a. We now formalize the intuitive idea that a set of atoms has no reason to be true:

Definition 1.1.7. A set U of atoms occurring in a program Π is *unfounded* on a consistent set M of literals with respect to Π if

$$\forall a \in U \quad \forall B \in Bodies(\Pi, a) \quad M \cap \overline{B} \neq \emptyset \vee U \cap B^{pos} \neq \emptyset$$

Theorem 1.1.8. *For a program Π and a consistent and complete set M of literals over $At(\Pi)$, M^+ is an answer set of Π if and only if M is a model of Π and M^+ contains no non-empty subsets unfounded on M with respect to Π.*

Proof. Left-to-right: Assume that M^+ is an answer set of Π. Since M^+ satisfies $(\Pi^{M^+})^{cl}$, M^+ satisfies Π^{cl}, i.e. M is a model of Π. It remains to verify that M^+ contains no non-empty subsets unfounded on M. Let U be an unfounded subset of M^+. We want to show that $M^+ \setminus U$ satisfies $(\Pi^{M^+})^{cl}$. Let $a \in Heads(\Pi^{M^+})$ and $B \in Bodies(\Pi^{M^+}, a)$. If $a \notin U$, since M^+ satisfies $a \vee \neg B^{pos}$ and B^{pos} is composed of positive atoms it follows that $M^+ \setminus U$ satisfies $a \vee \neg B^{pos}$ also. If $a \in U$, it has to be $U \cap B^{pos} \neq \emptyset$, because if it were $M \cap \overline{B} \neq \emptyset$ the rule $a \leftarrow B^{pos}$ would not be in Π^{M^+}. It follows that $M^+ \setminus U$ satisfies $a \vee \neg B^{pos}$. Since M^+ is minimal among the sets of atoms satisfying $(\Pi^{M^+})^{cl}$, it follows that $M^+ = M^+ \setminus U$, i.e. $U = \emptyset$.

Right-to-left: Assume that M is a model of Π and contains no non-empty subsets unfounded on M with respect to Π. It is easy to see that since M^+ satisfies Π, it satisfies also Π^{M^+}. It remains to check the minimality of M^+. Let N^+ be a subset of M^+ that satisfies Π^{M^+}. We want to prove that $U := M^+ \setminus N^+$ is an unfounded subset of M^+. Let $a \in U$ and $B \in Bodies(\Pi, a)$. If $M \cap \overline{B} = \emptyset$, $a \leftarrow B^{pos}$ is a rule of Π^{M^+}. Since $a \notin N^+$ and N^+ satisfies $a \leftarrow B^{pos}$, it has to be $B^{pos} \not\subseteq N^+$, hence $U \cap B^{pos} \neq \emptyset$. Since M^+ contains no non-empty unfounded subsets, we have $U = \emptyset$ and $N^+ = M^+$. □

1.2 Logic programs with constraint atoms

In this section we introduce the CASP formalism and its semantics. We begin recalling the definition of constraint satisfaction problem, the central matter of constraint logic programming.

Definition 1.2.1. A *constraint satisfaction problem (CSP)* is a triple $\langle X, D, C\rangle$ where X is a set of variables, D is a set of values, called domain, and C is a set of constraints. Every constraint is a pair $\langle t, R\rangle$, where t is an n-tuple of variables and R is an n-ary relation on D. A *solution* is a function $\nu : X \to D$ such that for every constraint $\langle (x_1, \ldots, x_n), R\rangle \in C$ we have $(\nu(x_1), \ldots, \nu(x_n)) \in R$.

For a constraint $c = \langle t, R\rangle$, where D is the domain of its variables and k is the arity of t, we call the constraint $\overline{c} = \langle t, D^k \setminus R\rangle$ the *complement* of c.

Definition 1.2.2. A *logic program with constraint atoms (CA program)* is a quadruple $\mathcal{P} = \langle \Pi, \mathcal{C}, \gamma, D\rangle$, where

1. $\mathcal{C}$ is an alphabet,
2. Π is a regular logic program such that
   (a) $a_0 \notin \mathcal{C}$ for every rule 1.1 in Π
   (b) $\mathcal{C} \subset At(\Pi)$,
3. γ is a function from $\mathcal{C}$ to constraints,
4. D is a domain.

Let $\mathcal{P} = \langle \Pi, \mathcal{C}, \gamma, D \rangle$ be a CA program. The elements of $\mathcal{C}$ are referred to as *constraint atoms*, while all atoms in $At(\Pi) \setminus \mathcal{C}$ are *regular*. By $\mathcal{V}_{\mathcal{P}}$ we denote the set of variables occurring in the constraint $\{\gamma(c) | c \in \mathcal{C}\}$. By $\Pi[\mathcal{C}]$ we denote Π extended with choice rules $\{c\}$ for each constraint atom $c \in \mathcal{C}$, where $\{c\}$ is an abbreviation for the rule

$$a \leftarrow \text{not}\,\text{not}\, a.$$

We call program $\Pi[\mathcal{C}]$ an *asp-abstraction* of $\mathcal{P}$.

Let M a consistent and complete set of literals over $At(\Pi)$. By $K_{\mathcal{P},M}$ we denote the following constraint satisfaction problem

$$\langle \mathcal{V}_{\mathcal{P}}, D, \{\gamma(c) | c \in M|_{\mathcal{C}}\} \cup \{\overline{\gamma(c)} | \neg c \in M|_{\mathcal{C}}\} \rangle.$$

We call this constraint satisfaction problem a *csp-abstraction* of $\mathcal{P}$ with respect to M.

We can now define the notion of answer set of a CA program:

Definition 1.2.3. Let $\mathcal{P} = \langle \Pi, \mathcal{C}, \gamma, D \rangle$ be a CA program and M be a consistent and complete set of literals over $At(\Pi)$. We say that M is an *answer set* of $\mathcal{P}$ if

(a1) M^+ is an answer set of $\Pi[\mathcal{C}]$ and

(a2) the constraint satisfaction problem $K_{\mathcal{P},M}$ has a solution.

If M is an answer set of $\mathcal{P}$ and ν is a solution of $K_{\mathcal{P},M}$, we say that the pair $\langle M, \nu \rangle$ is an *extended answer set* of $\mathcal{P}$.

CASP solvers such as CLINGCON process CA programs as described above, computing an answer set of a program or stating that it doesn't exist. However, the EZCSP solver interprets CA programs slightly differently: it computes "weak" answer sets. We now define this notion and discuss the differences.

Let M a consistent and complete set of literals over $At(\Pi)$. By $\tilde{K}_{\mathcal{P},M}$ we denote the following constraint satisfaction problem

$$\langle \mathcal{V}_{\mathcal{P}}, D, \{\gamma(c) | c \in M|_{\mathcal{C}}\} \rangle.$$

Definition 1.2.4. Let $\mathcal{P} = \langle \Pi, \mathcal{C}, \gamma, D \rangle$ be a CA program and M be a consistent and complete set of literals over $At(\Pi)$. We say that M is a *weak answer set* of $\mathcal{P}$ if

(w1) M^+ is an answer set of $\Pi[\mathcal{C}]$ and

(w2) the constraint satisfaction problem $\tilde{K}_{\mathcal{P},M}$ has a solution.

If M is a weak answer set of $\mathcal{P}$ and ν is a solution of $\tilde{K}_{\mathcal{P},M}$, we say that the pair $\langle M, \nu \rangle$ is an *extended weak answer set* of $\mathcal{P}$.

The difference between answer sets and weak answer sets lies in their conditions (a2) and (w2). Obviously answer sets are also weak answer sets, but there can be weak answer sets that are not answer sets and yield to a counter-intuitive, "unwanted" solution to the problem. For example, consider sample CA program, where constraint atoms are denoted with bars:

$$\begin{aligned} &night \leftarrow |x < 6| \\ &am \leftarrow |x < 12| \end{aligned}$$

This program has 3 answer sets (which are also weak answer sets)

$$\{night, am, |x < 6|, |x < 12|\}$$
$$\{\neg night, am, \neg|x < 6|, |x < 12|\}$$
$$\{\neg night, \neg am, \neg|x < 6|, \neg|x < 12|\}$$

and a weak answer set that is not an answer set

$$\{night, \neg am, |x < 6|, \neg|x < 12|\}$$

and we don't want to be a solution, as it states that it is currently night but not am hours.

Nevertheless, when constraint are used only in a certain way, answer set and weak answer set coincide. This is the case of the EZCSP language. In the following chapter, after we have formally defined the language, we will clarify this statement showing that weak answer set found by the solver are in fact answer sets.

The adoption of weak semantics was driven by the belief that it allows for a more flexible integration of solvers, and by different implementation choices with respect to other solver such as CLINGCON, which adopt non-weak semantics and allow for the constraint atoms to occur in arbitrary rules.

Example 1. Here we present a sample CA program $\mathcal{P}_1 = \langle \Pi_1, \mathcal{C}_1, \gamma_1, D_1 \rangle$ defined as follows.

- Π_1 is the regular program

$$\begin{aligned} &\{switch\}. \\ &lightOn \leftarrow switch, \text{not}\, am. \\ &\leftarrow \text{not}\, lightOn. \\ &\{am\}. \\ &\leftarrow \text{not}\, am, |x < 12|. \\ &\leftarrow am, |x \geq 12|. \end{aligned}$$

- $\mathcal{C}_1 = \{|x < 12|, |x \geq 12|\}$;

- γ_1 is the function that maps $|x < 12|$ to an inequality $x < 12$ and $|x \geq 12|$ to an inequality $x \geq 12$;
- D_1 is the range of integers from 0 to 23.

The asp-abstraction $\Pi[\mathcal{C}]$ of $\mathcal{P}$ is the regular program built extending Π with choice rules

$$\begin{array}{c}\{|x < 12|\}.\\ \{|x \geq 12|\}.\end{array}$$

Consistent and complete set

$$M_1 = \{switch, lightOn, \neg am, \neg|x < 12|, |x \geq 12|\}$$

of literals over $At(\Pi_1)$ is an answer set of $\mathcal{P}_1$. Indeed M_1^+ is an answer set of $\Pi_1[\mathcal{C}_1]$ and the constraint satisfaction problem

$$K_{\mathcal{P}_1, M_1} = \langle \{x\}, D_1, \{x \geq 12\} \rangle$$

has a solution. For instance pair

$$\langle M_1, x = 14 \rangle$$

is an extended answer set of program $\mathcal{P}$.

Chapter 2

EZCSP hybrid system

The EZCSP hybrid system has its roots in the development of an approach for integrating ASP and constraint logic programming. EZCSP language extends ASP language by adding some particular atoms that encode the desired satisfaction problems. The solver computes answer sets of the ASP program and solution to the associated constraint satisfaction problem using arbitrary off-the-shelf solvers.

2.1 EZCSP language

We begin the description of the EZCSP language by defining relation *required*, which is used to define the atoms that encode the constraints. Then we introduce the notion of *propositional ez-program* and introduce their semantics via a mapping into CA programs. Finally we define the notion of (non propositional) EZ program and we construct a mapping from EZ programs to propositional ez-programs.

2.1.1 Syntax

An *EZ-atom* is an expression of the form

$$required(\beta)$$

where β is an atom. Given an alphabet $\mathcal{C}$, $\mathcal{C}^{\mathrm{EZ}}$ is the corresponding set of ez-atoms.

A *(propositional) ez-program* is a tuple

$$\langle E, \mathcal{A}, \mathcal{C}, \gamma, D \rangle$$

where

- $\mathcal{A}$ and $\mathcal{C}$ are alphabets so that $\mathcal{A}$, $\mathcal{C}$ and $\mathcal{C}^{\mathrm{EZ}}$ do not share elements;
- E is a regular logic program so that $At(E) = \mathcal{A} \cup \mathcal{C}^{\mathrm{EZ}}$ and atoms from $\mathcal{C}^{\mathrm{EZ}}$ only occur in the head of its rules;
- γ is a function from $\mathcal{C}$ to constraints;
- D is a domain.

2.1.2 Semantics

We define a map from propositional ez-programs to CA programs:

$$\begin{aligned}\mathcal{M}\colon \{\text{propositional ez-programs}\} &\to \{\text{CA programs}\}\\ \langle E, \mathcal{A}, \mathcal{C}, \gamma, D\rangle &\mapsto \langle \Pi, \mathcal{C}, \gamma, D\rangle.\end{aligned}$$

where Π is the regular logic program that extends E by a denial

$$\leftarrow \mathit{required}(\beta), \text{not}\, \beta \tag{2.1}$$

for every ez-atom $\mathit{required}(\beta)$ occurring in E.

If $\mathcal{E}$ is a propositional ez-program, by $\mathcal{P}_{\mathcal{E}}$ we denote the corresponding CA program.

Definition 2.1.1. Let $\mathcal{E} = \langle E, \mathcal{A}, \mathcal{C}, \gamma, D\rangle$ be a propositional ez-program. For a consistent and complete set M of literals over $At(E) \cup \mathcal{C}$ and an evaluation α from the set $\mathcal{V}_{\mathcal{P}_{\mathcal{E}}}$ of variables to the set D of values, we say that

- M is an *answer set* of $\mathcal{E}$ if M is a weak answer set of $\mathcal{P}_{\mathcal{E}}$;
- a pair $\langle M, \alpha\rangle$ is an *extended answer set* of $\mathcal{E}$ if $\langle M, \alpha\rangle$ is an extended weak answer set of $\mathcal{P}_{\mathcal{E}}$.

We are now ready to discuss the difference between answer sets and weak answer sets with respect to CA programs associated to propositional ez-programs.

Let $\mathcal{E}$ be a propositional ez-program. Consider the asp-abstraction of the corresponding CA program $\mathcal{P}_{\mathcal{E}}$. Note that because of rules 2.1, if $\mathit{required}(\beta)$ is in an answer set M, it has to be $\beta \in M$. However, when $\mathit{required}(\beta) \notin M$, we are requiring neither $\beta \in M$ nor $\neg\beta \in M$. Because of choice rule $\{\beta\}.$, both choices are acceptable answer sets.

If the satisfaction of constraint β is not required, we disregard the presence of β in the answer set we are considering. Actually when the EZCSP system computes answer sets for the ez-program, it omits constraint atoms.

Formally, given an answer set M of the ez-program $\mathcal{E}$, we are concerned only with its subset $M \cap \mathit{Lit}(E)$. We can now show that if we consider only this subset of answer sets, weak answer sets of $\mathcal{P}_{\mathcal{E}}$ are also answer sets of $\mathcal{P}_{\mathcal{E}}$.

Lemma 2.1.2. *Let $\mathcal{E} = \langle E, \mathcal{A}, \mathcal{C}, \gamma, D\rangle$ be a propositional ez-program, and let M be a weak answer set of $\mathcal{P}_{\mathcal{E}}$. Then there exists an answer set N of $\mathcal{P}_{\mathcal{E}}$ such that $M \cap \mathit{Lit}(E) = N \cap \mathit{Lit}(E)$.*

Proof. Let $\langle M, \alpha\rangle$ be an extended weak answer set of $\mathcal{P}_{\mathcal{E}}$.

We now define a function f from $\mathcal{B} := \{\beta \in \mathcal{C} | \neg\beta \in M\}$ to $\mathit{Lit}(E)$.

$$\begin{aligned} f : \mathcal{B} &\to \mathit{Lit}(E)\\ \beta &\mapsto \begin{cases} \beta & \text{if } \alpha \text{ satisfies constraint } \beta,\\ \neg\beta & \text{if } \alpha \text{ does not satisfy constraint } \beta.\end{cases}\end{aligned}$$

Hence the set

$$N = (M \cap \mathit{Lit}(E)) \cup \{\beta | \beta \in \mathcal{C}, \beta \in M\} \cup \{f(\beta) | \beta \in \mathcal{C}, \neg\beta \in M\}$$

is an answer set of $\mathcal{P}_{\mathcal{E}}$. Indeed N^+ is an answer set of the asp-abstraction of $\mathcal{P}_{\mathcal{E}}$ and α is a solution of $K_{\mathcal{P}_{\mathcal{E}},N}$. Furthermore $M \cap Lit(E) = N \cap Lit(E)$, thus the theorem is proved. $\square$

2.1.3 General EZ programs

In order to allow for more compact specifications, the EZCSP system supports an extension of the language of propositional ez-programs, which we call EZ. The language supports an explicit specification of domains and variables, the use of non-ground rules and compact representation of lists in constraints. We begin defining EZ programs and then mapping them to propositional ez-programs.

Let $\Sigma_{\text{EZ}} = \langle C_{\text{EZ}}, V_{\text{EZ}}, F_{\text{EZ}}, R_{\text{EZ}} \rangle$ be a signature composed of disjoint set respectively of constants, variable, function and relation symbols, where

- set C_{EZ} includes symbols for integers and pre-defined constants (fd, q, r) denoting CSP domains;
- set F_{EZ} includes pre-defined symbols that correspond to arithmetic operators (e.g. "+"), arithmetic connectives (e.g. "<"), logical connectives, list delimiters and names of global constraints;
- set R_{EZ} contains pre-defined symbols *cspdomain*, *cspvar*, *required*.

The notion of terms over signature Σ_{EZ} is expanded (w.r.t. the traditional definition of first order logic) adding:

- *extensional list*: an expression of the form $[t_1, \ldots, t_k]$ where t_i are traditional terms;
- *intensional list*: an expression of the form $[g/k]$ where $g \in F_{\text{EZ}}$ or $g \in R_{\text{EZ}}$ and k is an integer;
- *global constraint*: an expression of the form $f(\gamma_1, \ldots, \gamma_k)$ where $f \in F_{\text{EZ}}$ and each γ_i is a list.

Definition 2.1.3. An EZ program is a pair $\langle \Sigma_{\text{EZ}}, \Pi \rangle$ where Π is a set of rules over signature Σ_{EZ} that contains exactly one fact whose head is $cspdomain(fd)$, $cspdomain(q)$, or $cspdomain(r)$.

Similarly to ASP, we replace every non-ground rule (rule containing variables) with a set of propositional (ground) rules, obtaining a ground EZ program.

It's possible to define a mapping from ground EZ programs to a propositional ez-programs:

$$\begin{aligned} \mu \colon \{\text{EZ programs}\} &\to \{\text{propositional ez-programs}\} \\ \langle \Sigma_{\text{EZ}}, \Pi \rangle &\mapsto \langle \mu_V(\Pi) \cup \mu_R(\Pi), \mu_{\mathcal{A}}(\Pi), \mu_{\mathcal{C}}(\Pi), \gamma, \mu_D(\Pi) \rangle \end{aligned}$$

where

$$\mu_D(\Pi) = \begin{cases} \mathcal{FD} \text{ (finite domains)} & \text{if } \mathit{cspdomain}(fd). \in \Pi \\ \mathcal{Q} & \text{if } \mathit{cspdomain}(q). \in \Pi \\ \mathcal{R} & \text{if } \mathit{cspdomain}(r). \in \Pi \end{cases}$$

$$\begin{aligned}\mu_V(\Pi) =& \{\mathit{required}(v \geq l).|\mathit{cspvar}(v,l,r). \in \Pi\} \cup \\ & \{\mathit{required}(v \leq r).|\mathit{cspvar}(v,l,r). \in \Pi\}\end{aligned}$$

$$\tilde{\mu}_R(a \leftarrow B) = \begin{cases} \mathit{required}(\lambda(\beta)) \leftarrow B & \text{if } a \text{ is of the form } \mathit{required}(\beta) \\ a \leftarrow B & \text{otherwise} \end{cases}$$

where $\tilde{\mu}$ is a function that maps intensional lists to corresponding extensional lists. We omit details about its definition. Using $\tilde{\mu}$ we can now define

$$\mu_R(\Pi) = \bigcup_{r \in \Pi} \tilde{\mu}_R(r).$$

Besides using μ_V and μ_R we define

$$\mu_{\mathcal{A}}(\Pi) = \{a \in \mathit{At}(\mu_V(\Pi) \cup \mu_R(\Pi))|a \neq \mathit{required}(-)\}$$

$$\mu_{\mathcal{C}}(\Pi) = \{\beta|\mathit{required}(\beta) \in \mu_V(\Pi) \cup \mu_R(\Pi)\}.$$

We assume that γ, a function from $\mathcal{C}$ to constraint, is given. It represents the canonical association between arithmetic connectives and constraints.

Henceforth, when we talk about answer sets of a general EZ program $\langle \Sigma_{\text{EZ}}, \Pi \rangle$ we mean answer sets of the associated propositional ez-program $\mu(\langle \Sigma_{\text{EZ}}, \Pi \rangle)$.

2.2 System Architecture

In this section we give an overview of the architecture of the EZCSP system, depicted in Figure 2.1, focusing on the functioning while employing the *black-box* integration schema.

When the EZCSP system takes as input an EZ program, the *Pre-processor* translates it into a syntactically legal ASP program. Indeed EZ atoms may contain arithmetic function or operators that have to be translated into auxiliary function symbols. Then the *Grounder* transforms the resulting program into its propositional equivalent. This regular program is then passed to the *EZCSP solver* component, that iterates *ASP solver* and *CP solver* computations. Furthermore the EZCSP solver uses the *CLP translator* component for mapping the csp-abstraction corresponding to the computed answer set to a Prolog program. Details about the behaviour of the EZCSP solver will be discussed in the next section.

EZCSP supports different off-the-shelf grounders, such as GRINGO, CP solvers, such as SICSTUS, BPROLOG, SWIPROLOG, and ASP solvers, such as CMODELS or CLASP.

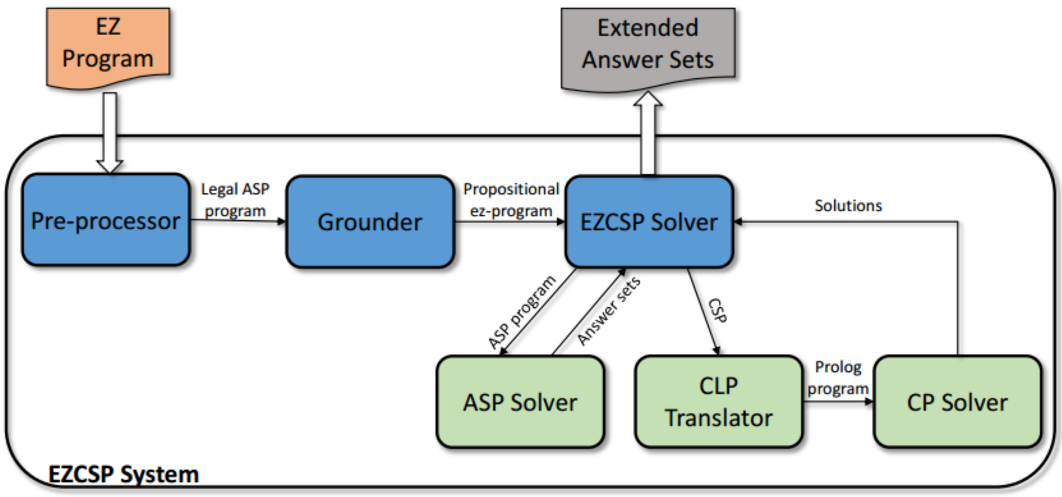

Figure 2.1: Architecture of the EZCSP system (Image from [1])

Example 2. In this example we illustrate the functioning of the system using a problem similar to example 1. In the EZ language, we can encode the problem as follows.

```
    cspdomain(fd).
    cspvar(x,0,23).
    {switch}.
    lightOn :- switch, not am.
    :- not lightOn.
    {am}.
    required(x>=12) :- not am.
    required(x<12) :- am.
    {night}.
    required(x<6) :- night.
    required(x>=6) :- not night.
```

We now analyse the behaviour of the EZCSP system given this program as input. First the pre-processor replaces arithmetic connectives with special function symbols. Its output is

```
    cspdomain(fd).
    cspvar(x,0,23).
    {switch}.
    lightOn :- switch, not am.
    :- not lightOn.
    {am}.
    required(ezcsp__geq(x, 12)) :- not am.
    required(ezcsp__lt(x, 12)) :- am.
    {night}.
    required(ezcsp__lt(x, 6)) :- night.
    required(ezcsp__geq(x, 6)) :- not night.
```

This program is already grounded since there are no non-constraint variables. Thus it is a legal propositional ez-program that the EZCSP solver can take as input. The ASP solver now computes an answer set of this regular program. There are two answer sets (in the sense of regular programs):

$$\{cspdomain(fd), cspvar(x, 0, 23), lightOn, switch, required(ezcsp\_\_\_geq(x, 12)),$$
$$required(ezcsp\_\_\_lt(x, 6)), night\}$$

and

$$\{cspdomain(fd), cspvar(x, 0, 23), lightOn, switch, required(ezcsp\_\_\_geq(x, 12)),$$
$$required(ezcsp\_\_\_geq(x, 6))\}.$$

Then the EZCSP solver sends to the CLP translator the answer set found and after that the CP solver searches a solution of the constraint satisfaction problem. If the ASP solver finds the first solution, the CP solver will find no solution of the csp-abstraction because the constraint variable x cannot be both greater than 12 and less than 6. Hence in this case the EZCSP solver calls the ASP solver a second time in order to find another solution. If the ASP solver finds the second solution, the CP solver will output a solution to the constraint satisfaction problem, for example $x = 13$. In this case the EZCSP solver outputs the extended answer set

$$\langle\{cspdomain(fd), cspvar(x, 0, 23), lightOn, switch, required(ezcsp\_\_\_geq(x, 12)),$$
$$required(ezcsp\_\_\_geq(x, 6))\}, x = 13\rangle$$

and the process stops.

In the next section we will study how exactly the EZCSP solver interacts with the ASP and the CP solver.

2.3 EZCSP solver

We begin introducing some notions and lemmata about CA programs that will be useful in order to prove formal properties of the EZCSP solver.

Definition 2.3.1. Let $\mathcal{P} = \langle \Pi, \mathcal{C}, \gamma, D\rangle$ be a CA program. We say that $\mathcal{P}$ *asp-entails* a denial G over $At(\Pi)$ when for every complete and consistent set M of literals over $At(\Pi)$ such that M^+ is an answer set of $\Pi[\mathcal{C}]$, M satisfies G^{cl}.

Definition 2.3.2. Let $\mathcal{P} = \langle \Pi, \mathcal{C}, \gamma, D\rangle$ be a CA program. We say that $\mathcal{P}$ *cp-entails* a denial G over $At(\Pi)$ when

1. for every answer set M of $\mathcal{P}$, M satisfies G^{cl}, and

2. there is a complete and consistent set N of literals over $At(\Pi)$ such that N^+ is an answer set of $\Pi[\mathcal{C}]$ and N does not satisfy G.

We say that a CA program $\mathcal{P} = \langle \Pi, \mathcal{C}, \gamma, D \rangle$ *entails* a denial G over $At(\Pi)$ when $\mathcal{P}$ either asp-entails or cp-entails G. For a consistent set N of literals over $At(\Pi)$ and a literal l, we say that $\mathcal{P}$ *asp-entails* l with respect to N, if for every complete and consistent set M of literals over $At(\Pi)$ such that M^+ is an answer set of $\Pi[\mathcal{C}]$ and $N \subset M$, $l \in M$.

For a CA program $\mathcal{P} = \langle \Pi, \mathcal{C}, \gamma, D \rangle$ and a set Γ of denials over $At(\Pi)$, by $\mathcal{P}[\Gamma]$ we denote the CA program $\langle \Pi \cup \Gamma, \mathcal{C}, \gamma, D \rangle$.

Lemma 2.3.3. *For a CA program $\mathcal{P} = \langle \Pi, \mathcal{C}, \gamma, D \rangle$ and a set Γ of denials over $At(\Pi)$ if $\mathcal{P}$ entails every denial in Γ then*
(i) every answer set of $(\Pi \cup \Gamma)[\mathcal{C}]$ is also an answer set of $\Pi[\mathcal{C}]$;
(ii) CA programs $\mathcal{P}$ and $\mathcal{P}[\Gamma]$ have the same answer sets.

Proof. (i) This part follows from 1.1.6 applied to regular program $\Pi[\mathcal{C}]$ and set $(\Pi \cup \Gamma)[\mathcal{C}] \setminus \Pi[\mathcal{C}]$ of denials.

(ii) Set Γ is composed of two disjoint sets Γ_1 and Γ_2, where Γ_1 is the set of all denials that are asp-entailed by $\mathcal{P}$ and Γ_2 is the set of all denials cp-entailed by $\mathcal{P}$. From 1.1.6 and definition of asp-entailment, regular programs $\Pi[\mathcal{C}]$ and $(\Pi \cup \Gamma_1)[\mathcal{C}]$ have the same answer sets. Furthermore $K_{\mathcal{P},M} = K_{\mathcal{P}[\Gamma_1],M}$ for any answer set M of $\Pi[\mathcal{C}]$. Thus CA programs $\mathcal{P}$ and $\mathcal{P}[\Gamma_1]$ have the same answer sets.

We now show that also CA programs $\mathcal{P}[\Gamma_1]$ and $\mathcal{P}[\Gamma_1 \cup \Gamma_2]$ have the same answer sets.

Let M be an answer set of $\mathcal{P}[\Gamma_1]$. Since $\mathcal{P}[\Gamma_1]$ cp-entails every denial in Γ_2, we conclude that M is a model of Γ_2^{cl}. By theorem 1.1.6, M^+ is an answer set of $(\Pi \cup \Gamma_1)[\mathcal{C}] \cup \Gamma_2$. But $(\Pi \cup \Gamma_1)[\mathcal{C}] \cup \Gamma_2 = (\Pi \cup \Gamma_1 \cup \Gamma_2)[\mathcal{C}]$ and $K_{\mathcal{P}[\Gamma_1],M} = K_{\mathcal{P}[\Gamma_1 \cup \Gamma_2],M}$ thus M is an answer set of $\Pi[\Gamma_1 \cup \Gamma_2]$.

Take M to be an answer set of $\mathcal{P}[\Gamma_1 \cup \Gamma_2]$. M^+ is an answer set of $(\Pi \cup \Gamma_1)[\mathcal{C}] \cup \Gamma_2 = (\Pi \cup \Gamma_1 \cup \Gamma_2)[\mathcal{C}]$. By theorem 1.1.6 M^+ is an answer set of $(\Pi \cup \Gamma_1)[\mathcal{C}]$ and since $K_{\mathcal{P}[\Gamma_1],M} = K_{\mathcal{P}[\Gamma_1 \cup \Gamma_2],M}$ we derive that M is an answer set of $\mathcal{P}[\Gamma_1]$.

It immediately follows that CA programs $\mathcal{P}$ and $\mathcal{P}[\Gamma_1 \cup \Gamma_2]$ have the same answer sets.

$\square$

Rather than committing ourselves with a pseudocode description, with introduce a graph-based representation of the solver, where nodes represent possible "state of computation" and edge transition from one state to another. This framework allow us to speak of termination and correctness of procedures supported by the EZCSP system and to easily capture differences and similarities of the various configuration supported by EZCSP. This approach allows us to model a complex algorithm by a mathematically simple and elegant object, a graph, rather than a collection of pseudocode statements.

Definition 2.3.4. For an alphabet σ, a *record* relative to σ is a sequence M composed of distinct literals over σ or symbol $\perp$, with some literals possibly annotated with the symbol Δ, such that

1. the set of literals in M is consistent or $M = M'l$, where the set of literals in M' is consistent and contains $\bar{l}$;
2. if $M = M'l^{\Delta}M''$, then neither l nor its dual $\bar{l}$ is in M';
3. if $\perp$ occurs in M, then $M = M'\perp$ and M' does not contain M.

Literals annotated with the symbol Δ are called *decision literals.* Intuitively, decision literals in a record mean that we do not now yet if the literal is true or false, we momentarily assume that they are true, but we are ready to backtrack and negate them as soon as we find a contradiction.

Definition 2.3.5. Let $\mathcal{P} = \langle \Pi, \mathcal{C}, \gamma, D \rangle$ be a CA program. A state relative to $\mathcal{P}$ is either

- a distinguished state *failstate* or
- a triple $M||\Gamma||\Lambda$ where M is a record relative to $At(\Pi)$ and Γ and Λ are each a set of denials that are entailed by $\mathcal{P}$.

We now consider the following transition rules from one state to another.

- *Decide:* $M||\Gamma||\Delta \Rightarrow Ml^{\Delta}||\Gamma||\Delta$, if l is unassigned by M and M is consistent;
- *Fail:* $M||\Gamma||\Delta \Rightarrow Failstate$, if M is inconsistent and M contains no decision literals;
- *Backtrack:* $Pl^{\Delta}Q||\Gamma||\Delta \Rightarrow P\bar{l}||\Gamma||\Delta$, if $Pl^{\Delta}Q$ is inconsistent and Q contains no decision literals;
- *ASP-Propagate:* $M||\Gamma||\Delta \Rightarrow Ml||\Gamma||\Delta$, if $\mathcal{P}[\Gamma \cup \Delta]$ asp-entails l with respect to M;
- *CP-Propagate:* $M||\Gamma||\Delta \Rightarrow M\perp||\Gamma||\Delta$, if $K_{\mathcal{P},M}$ has no solution;
- *Learn:* $M||\Gamma||\Delta \Rightarrow M||\Gamma \cup \{R\}||\Delta$, if $\mathcal{P}[\Gamma \cup \Delta]$ entails denial R and $R \notin \Gamma \cup \Delta$;
- *Learn^t:* $M||\Gamma||\Delta \Rightarrow M||\Gamma||\Delta \cup \{R\}$, if $\mathcal{P}[\Gamma \cup \Delta]$ entails denial R and $R \notin \Gamma \cup \Delta$;
- *Restart:* $M||\Gamma||\Delta \Rightarrow \emptyset||\Gamma||\Delta$, if $M \neq \emptyset$;
- *Restart^t:* $M||\Gamma||\Delta \Rightarrow \emptyset||\Gamma||\emptyset$, if $M \neq \emptyset$.

Definition 2.3.6. For a CA program $\mathcal{P} = \langle \Pi, \mathcal{C}, \gamma, D \rangle$, we define the (oriented) graph $\mathrm{EZ}_{\mathcal{P}}$, whose nodes are the states relative to $\mathcal{P}$ and edges are specified by the nine transition rule presented above.

The transition rule *ASP-Propagate* specifies the conditions under which a new literal l is added to an atomic part. Note that the condition of this rule is semantic, as it refers to the notion of asp-entailment. Propagators used by software system typically use syntactic conditions, which are easier to check. Specifically, the EZCSP solver accounts only for these two special cases of ASP-Propagate:

- *Unit Propagate:* $M||\Gamma||\Delta \Rightarrow Ml||\Gamma||\Delta$, if M is consistent, $C \vee l \in (\Pi[\mathcal{C}] \cup \Gamma \cup \Delta)^{cl}$ and $M \models \overline{C}$;

- *Unfounded:* $M||\Gamma||\Delta \Rightarrow Ml||\Gamma||\Delta$, if M is consistent, and there is a literal l so that $\bar{l} \in U$ for a set U, which is unfounded on M w.r.t. $\Pi[\mathcal{C}] \cup \Gamma \cup \Delta$.

These two transition rules rely on properties that can be checked by efficient procedures. Furthermore *Unit Propagate* or *Unfounded* are applicable only in states where *ASP-Propagate* is applicable. Thus we can define the graph EZSM$_\mathcal{P}$, an edge-induced subgraph of EZ$_\mathcal{P}$, where we drop the edges that correspond to the application of transition rule *ASP-Propagate* not accounted by *Unit Propagate* or *Unfounded.*

Graph EZ$_\mathcal{P}$ and EZSM$_\mathcal{P}$ share important properties that give rise to a class of correct algorithms for computing answer sets of CA programs. Before stating these properties, we introduce some terminology.

For a state $M||\Gamma||\Delta$, we call M,Γ, and Δ the *atomic*, *permanent*, and *temporal* parts of the state. We refer to the transition rules *Decide*, *Fail*, *Backtrack*, *ASP-Propagate*, *CP-Propagate* of graph EZ$_\mathcal{P}$ as *basic.* Similarly, in graph EZSM$_\mathcal{P}$, transition rules *Decide*, *Fail*, *Backtrack*, *Unit Propagate*, *Unfounded* and *CP-Propagate* are called *basic.* They concern only the atomic part of the state. We call the state $\emptyset||\emptyset||\emptyset$ *initial.* We say that a node in EZ$_\mathcal{P}$ is *semi-terminal* if no basic rule is applicable to it.

We say that a path in EZ$_\mathcal{P}$ is *restart-safe* when, prior to any edge e due to an application of *Restart* or *Restart^t* on this path, there is an edge e' due to an application of *Learn* such that

- edge e' precedes e;
- e' does not precede any other edge $e'' \neq e$ due to *Restart* or *Restart^t*.

Lemma 2.3.7. *For any CA program $\mathcal{P}$, and a path from an initial state to $l_1 \dots l_n||\Gamma||\Delta$ in EZ$_\mathcal{P}$, if X is an answer set for $\mathcal{P}$, $i \in \{1, \dots, n\}$ and X satisfies all decision literals l_j^Δ with $j \leq i$ then X satisfies l_i.*

Proof. By induction on the length of a path. The property trivially hold in the initial state, we only need to prove that all transition rules of EZ$_\mathcal{P}$ preserve it. Consider a transition $M||\Gamma||\Delta \Rightarrow S$ where S is a state and $M = l_1 \dots l_n$ is a sequence of literals (or decision literals). Suppose the statement is true for the state $M||\Gamma||\Delta$. We want to prove it for S. We consider different cases depending on the transition rule leading to S.

- *Decide:* $S = Ml_{n+1}^\Delta||\Gamma||\Delta$. By induction hypothesis, the statement is true for $i \in \{1, \dots, n\}$. For $i = l_{n+1}$, take any answer set X for $\mathcal{P}$ that satisfies all decision literals with $i \leq n+1$. X trivially satisfies also l_{n+1}.
- *Fail:* $S = Failstate$. Nothing to prove.
- *CP-Propagate:* $S = M\bot||\Gamma||\Delta$. Nothing to prove.
- *Learn, Learn^t:* Nothing to prove since the atomic part is unchanged.
- *Restart, Restart^t:* Obvious since the atomic part of S is $\emptyset$.
- *ASP-Propagate:* $S = Ml_{n+1}||\Gamma||\Delta$. Take any answer set X of $\mathcal{P}$ such that X satisfies all decision literals l_j^Δ with $j \leq n+1$. From the inductive hypothesis it follows that X satisfies M. Thus $M \subset X$ and from the definition of *ASP-Propagate*,

$\mathcal{P}$ asp-entails l_{n+1} with respect to M. We also know that X^+ is an answer set of $\Pi[\mathcal{C}]$. Thus from the definition of asp-entailment $l_{n+1} \in X$.

- *Backtrack:* $M = Pl_i^{\Delta}Q$ where Q contains no decision literals. $S = P\overline{l_i}||\Gamma||\Delta$. Take any answer set X of $\mathcal{P}$ such that X satisfies all decision literals l_j^{Δ} with $j \leq i$. We need to show that $X \models \overline{l_i}$. Proof by contradiction. Assume that $X \models l_i$. By inductive hypothesis, since Q does not contain decision literals, X satisfies $Pl_i^{\Delta}Q$. This is impossible because M is inconsistent since we have applied *Backtrack* to it. □

We are now ready to prove that we can use paths in EZ$_{\mathcal{P}}$ or in EZSM$_{\mathcal{P}}$ to find answer sets of CA programs. Every restart-safe path leads (in a finite number of transitions) to an answer set or to $Failstate$, depending on the existence of answer sets. The remaining part of this section contains formal proofs of this statement for both EZ$_{\mathcal{P}}$ and EZSM$_{\mathcal{P}}$.

Theorem 2.3.8. *For any CA program $\mathcal{P}$:*

(i) every restart-safe path in EZ$_{\mathcal{P}}$ is finite, and any maximal restart-safe path ends with a state that is semi-terminal;

(ii) for any semi-terminal state $M||\Gamma||\Delta$ of EZ$_{\mathcal{P}}$ reachable from initial state, M is an answer set of $\mathcal{P}$;

(iii) state $Failstate$ is reachable from initial state in EZ$_{\mathcal{P}}$ by a restart-safe path if and only if $\mathcal{P}$ has no answer set.

Proof. Let $\mathcal{P} = \langle \Pi, \mathcal{C}, \gamma, D \rangle$ be a CA program.

(i) We first show that any path in EZ$_{\mathcal{P}}$ that does not contain *Restart* or *Restart^t* edges is finite.

Consider any path t that does not contain these edges. We want to define an anti-symmetric and transitive relation $\prec$ over the set $\mathcal{S}$ of states of EZ$_{\mathcal{P}}$. We begin defining a function $\alpha : \mathcal{S} \setminus \{Failstate\} \to \{\text{Finite sequences of nonnegative integers}\}$. Any state $M||\Gamma||\Delta$ can be written uniquely as $M_0 l_1^{\Delta} M_1 \ldots l_p^{\Delta} M_p||\Gamma||\Delta$, where $M_0, \ldots, M_p$ do not contain decision literals. We define $\alpha(M||\Gamma||\Delta)$ as the sequence of nonnegative integers $|M_0|, |M_1|, \ldots, |M_p|$. Over the set of all finite sequences of nonnegative integers the relation $<$ is interpreted as the lexicographic order.

We now define $\prec$ as follows.

- $\forall S \in \mathcal{S} \setminus \{Failstate\} \quad S \prec Failstate$;
- for every pair of states $M||\Gamma||\Delta \in \mathcal{S}$ and $M'||\Gamma'||\Delta' \in \mathcal{S}$, we say that $M||\Gamma||\Delta \prec M'||\Gamma'||\Delta'$ if and only if
  - $\Gamma \subset \Gamma'$, or
  - $\Gamma = \Gamma'$ and $\Delta \subset \Delta'$, or
  - $\Gamma = \Gamma'$, $\Delta = \Delta'$, and $\alpha(M||\Gamma||\Delta) < \alpha(M'||\Gamma'||\Delta')$.

It's easy to see that $\prec$ is anti-symmetric and transitive. By the definition of the transition rules and the transitiveness, if there is any edge from $M||\Gamma||\Delta$ to $M'||\Gamma'||\Delta'$ in $\mathrm{EZ}_{\mathcal{P}}$ formed by any basic rule or rules *Learn* or *Learn*^{t}, then $M||\Gamma||\Delta \prec M'||\Gamma'||\Delta'$. Observe that there is a finite number of distinct values of α: the length of the atomic part of a states does not exceed $2 * |At(\Pi)| + 1$. Furthermore there is a finite number of denials entailed by $\mathcal{P}$ because the atoms of these denials have to be a subset of $At(\Pi)$. Hence there is a finite number of edges in t.

Consider any restart-safe path r in $\mathrm{EZ}_{\mathcal{P}}$. We call a path from state S to state S' in $\mathrm{EZ}_{\mathcal{P}}$ *restarting* when an edge that follows S is due to the application from the rule *Learn*, an edge leading to S' is due to the application of the rule *Restart* or *Restart*^{t} and on this path there are no other edges due to applications of *Learn*, *Restart* or *Restart*^{t}. Using the previous part of the proof, it follows that any restarting path is finite. We now construct a path r' by replacing each restarting subpath of r by an edge that we call *Artificial*. An edge in r' due to *Artificial* leads from a state of the form $M||\Gamma||\Delta$ to a state $\emptyset||\Gamma \cup \{C\}||\Delta'$, where C is a denial. We have $M||\Gamma||\Delta \prec \emptyset||\Gamma \cup \{C\}||\Delta'$. Furthermore r' contains no edges due to *Restart* or *Restart*^{t}, because we eliminated all restarting subpaths in favor of *Artificial* edges. Thus by the same argument as the first part of the proof, r' contains a finite number of edges. We can now conclude that r is finite because r' was built dropping finite fragments from r.

It's easy to see that any maximal restart-safe path ends with a state that is semi-terminal. If it were such a path ending in a non semi-terminal not, we could apply a basic transition rule to it, contradicting its maximality.

(ii) Let $M||\Gamma||\Delta$ be a semi-terminal state reachable from initial state. Since *Decide* is not applicable, M assigns all literals or M is inconsistent. M is consistent because if it were inconsistent, *Backtrack* or *Fail* would be applicable. So M assigns all literals.

M^+ is an answer set of $\Pi[\mathcal{C}]$. Proof by contradiction. Assume that M^+ is not an answer set of $\Pi[\mathcal{C}]$. It follows that M is not an answer set of $\mathcal{P}$. By 2.3.3, M is not an answer set of $\mathcal{P}[\Gamma \cup \Delta]$ and M^+ is not an answer set of $\Pi[\mathcal{C}] \cup \Gamma \cup \Delta$. Since M is a complete and consistent set of literals over $At(\Pi)$ it follows that there is no complete and consistent set M' of literals over $At(\Pi)$ such that $M \subset M'$ and M'^+ is an answer set of $\Pi[\mathcal{C}] \cup \Gamma \cup \Delta$. We trivially conclude that $\mathcal{P}[\Gamma \cup \Delta]$ asp-entails any literal l. It follows that *ASP-Propagate* is applicable to state $M||\Gamma||\Delta$, that contradict our assumption that this state is semi-terminal.

(iii) Left-to-right: Since $Failstate$ is reachable from the initial state by a restart-safe path, there is an inconsistent state $M||\Gamma||\Delta$ without decision literals such that there exists a path a path from the initial state to $M||\Gamma||\Delta$. By 2.3.7, any answer set of $\mathcal{P}$ satisfies M. Since M is inconsistent we conclude that $\mathcal{P}$ has no answer sets.

Right-to-left: From (i) any maximal restart-safe path from initial state ends with some semi-terminal state S. By (ii) S cannot be different from $Failstate$ because

$\mathcal{P}$ has no answer sets.

$\square$

Theorem 2.3.9. *For any CA program $\mathcal{P}$:*

(i) every restart-safe path in EZSM$_\mathcal{P}$ is finite, and any maximal restart-safe path ends with a state that is semi-terminal;

(ii) for any semi-terminal state $M||\Gamma||\Delta$ of EZSM$_\mathcal{P}$ reachable from initial state, M is an answer set of $\mathcal{P}$;

(iii) state $Failstate$ is reachable from initial state in EZSM$_\mathcal{P}$ by a restart-safe path if and only if $\mathcal{P}$ has no answer set.

Proof. Let $\mathcal{P} = \langle \Pi, \mathcal{C}, \gamma, D \rangle$ be a CA program.

(i) The finiteness follows from (i) of 2.3.8 since EZSM$_\mathcal{P}$ is a subgraph of EZ$_\mathcal{P}$.

By the same argument as in (i) of 2.3.8, any maximal restart-safe path ends with a state that is semi-terminal.

(ii) Let $M||\Gamma||\Delta$ be a semi-terminal state reachable from initial state. As in proof of part (ii) of 2.3.8 we conclude that M assigns all literals and is consistent. Also, CSP $K_{\mathcal{P},M}$ has a solution.

We now show that M^+ is an answer set of $\Pi[\mathcal{C}]$. Proof by contradiction. Assume that M^+ is not an answer set of $\Pi[\mathcal{C}]$. It follows that M is not an answer set of $\mathcal{P}$. By 2.3.3, M is not an answer set of $\mathcal{P}[\Gamma \cup \Delta]$ and M^+ is not an answer set of $\Pi[\mathcal{C}] \cup \Gamma \cup \Delta$. By 1.1.8 it follows that either M is not a model of $\Pi[\mathcal{C}] \cup \Gamma \cup \Delta$ or M contains a non-empty subset unfounded on M w.r.t. $\Pi[\mathcal{C}] \cup \Gamma \cup \Delta$. In case the former holds we derive that the rule *Unit Propagate* is applicable in the state $M||\Gamma||\Delta$, because there is $C \vee l \in (\Pi[\mathcal{C}] \cup \Gamma \cup \Delta)^{cl}$ such that $M \models \overline{C \vee l}$ and in particular $M \models \overline{C}$. In case the latter holds we derive that the rule *Unfounded* is applicable in the state $M||\Gamma||\Delta$, since given the unfounded set U we can choose any $l \in U$ and it satisfies the conditions of the rule. In either cases this contradicts our assumption that $M||\Gamma||\Delta$ is semi-terminal.

From the conclusions that M^+ is an answers set of $\Pi[\mathcal{C}]$ and $K_{\mathcal{P},M}$ has a solution we derive that M is an answer set of $\mathcal{P}$.

(iii) Left-to-right: It follows from (iii) of 2.3.8, noting that 2.3.7 holds also for the graph EZSM$_\mathcal{P}$ because it is a subgraph of EZ$_\mathcal{P}$.

Right-to-left: From (i) any maximal restart-safe path from initial state ends with some semi-terminal state S. By (ii) S cannot be different from $Failstate$ because $\mathcal{P}$ has no answer sets.

$\square$

2.4 Integration schemas

An algorithm that traverses the graph $EZ_{\mathcal{P}}$ is characterized by a strategy for choosing a path at each state. EZCSP solver implements three different strategies, *black box*, *grey-box* or *clear box* integration, depending on how and when the ASP and the CP solver are invoked and how interact with each other.

We now outline these three strategies, which provide a skeleton of the algorithm implemented in the EZCSP system. These skeletons are meant to highlight key differences between the variants of integration schemas, whilst details are omitted.

2.4.1 Black-box integration

In this configuration schema, the ASP solver is called iteratively and then the solution are checked with the CP solver. In terms of $EZ_{\mathcal{P}}$, we can describe this strategy as follows.

1. *Restart* never applies;
2. *Learn*^{t} applies only if the denial R learnt by the application of this rule is such that $\mathcal{P}$ asp-entails R;
3. *CP-Propagate* never applies only if rules *Decide*, *Backtrack*, *Fail*, *ASP-Propagate* are not applicable;
4. A single application of *Learn* follows immediately after an application of the rule *CP-Propagate*. Furthermore, the denial R learnt by the application of this rule is such that $\mathcal{P}$ cp-entails R;
5. *Restart*^{t} follows immediately after an application of the rule *Learn*. *Restart*^{t} does not apply under any other condition.

2.4.2 Grey-box integration

In this approach, rather than calling the ASP solver from scratch at each iteration, the EZCSP solver implements an incremental interface that saves informations obtained from the ASP solver at each iteration. Formally, in terms of graph $EZ_{\mathcal{P}}$, this configuration is captured by the following strategy.

1. *Restart*^{t} never applies;
2. *Learn*^{t} applies only if the denial R learnt by the application of this rule is such that $\mathcal{P}$ asp-entails R;
3. *CP-Propagate* never applies only if rules *Decide*, *Backtrack*, *Fail*, *ASP-Propagate* are not applicable;
4. A single application of *Learn* follows immediately after an application of the rule *CP-Propagate*. Furthermore, the denial R learnt by the application of this rule is such that $\mathcal{P}$ cp-entails R;

5. *Restart* follows immediately after an application of the rule *Learn*. *Restart* does not apply under any other condition.

This strategy differs from the previous one only in rules 1 and 5. Here rule *Restart* plays the role of rule *Restart^t* in black-box configuration, avoiding to delete the temporal part of the state, i.e. asp-entailed denials, throughout the traversing of the graph.

2.4.3 Clear-box integration

In this approach, satisfiability of the partial constraint satisfaction problem is checked while the assignment is being built by the ASP solver. This configuration is captured by the following strategy in navigating the graph $\text{EZ}_{\mathcal{P}}$.

1. *Restart^t* and *Restart* never apply.

We omit details about this strategy. Note that permitting frequent dialogue between ASP and CP solvers as in clear-box integration is not the best choice in all cases. Indeed it has been shown that there is no single choice that is faster in all problems (see [1]).

Chapter 3

Application Domains

3.1 Blending Problem

A classical problem that can be modeled in the EZCSP language concerns blending or mixing substances, each of them containing different quantities of some components, to obtain a product whose amount of components lies within specified bounds. Each substance has a known cost, our goal is to find the cheapest mixture that satisfies the requests. We refer to the decisional version of this problem: given a maximum cost k, we want to establish if there exists a solution whose cost is less than or equal to k.

For example consider the problem of determining the optimum amounts of three ingredients to include in an animal feed mix. The final product must satisfy several nutrient restrictions. Similar problems arise also in other domains, for example, the oil, paint, and food-processing industries.

Here we present a true-CASP encoding of the problem in the EZCSP language. We denote with *true-CASP* the encodings that exploits both ASP rules and constraint rules, i.e. that will use both the ASP and the CP solver. The input file is composed of the following predicates:

- $subs(X)$., which states that X is a substance;
- $cost(S, P)$., which states that substance S costs P (in a suitable predetermined unit of measurement);
- $comp(X)$., which states that X is a component;
- $mi(C, Q)$., which states that the final product has to contain at least Q units of component C;
- $tab(S, C, Q)$., which states that substance S contains Q units of component C for each predetermined unit of measurement established before;
- $max\_money(M)$., which gives the maximum cost of an acceptable solution.

In addition to this basic problem there can be additional requests. We consider the following:

- $using\_cost(P)$., which states that there is a fixed cost P for each substance used in the final product, which we have to add to the cost per unit;
- $not\_together(X, Y)$., which states that substance X and Y cannot be both used

for the final product;

- *have_to_use*(X, N, D)., where N and D are integers. It states that we have to use minimum N/D units of substance X.

```
cspdomain(q).
cspvar(q(X),0,10) :- subs(X).   % Quantity of each substance
cspvar(p(X),0,1000) :- subs(X). % Expense for each substance
cspvar(c(Y,X),0,200) :- subs(X), comp(Y). % Amount of component Y in
                                            used quantity of substance X

% Decide whether a substance is used or not.
{used(X)} :- sost(X).
required(q(X)=0) :- not used(X), subs(X).
required(q(X)>0) :- used(X), subs(X).

% Expence for each substance = quantity * cost per unit + fixed cost.
required( p(X)=Y*q(X)+Z ) :- subs(X), cost(X,Y), used(X), using_cost(Z).
required( p(X)=0 ) :- subs(X), cost(X,Y), not used(X).

% Amount of each component in each substance.
required(c(Y,X) == q(X) * Z) :- subs(X), comp(Y), tab(X,Y,Z).

% For each component, the amount in the final product has to be
% greater than the minimum requested.
required(sum([c(Y)/2],>=,Z)) :- comp(Y), mi(Y,Z).

% Sum of expenses cannot exceed the budget.
required(sum([p/1],<=,X)) :- max_money(X).

%%% EXTRA REQUIREMENTS

% Substances X and Y cannot be used together.
:- used(X), used(Y), not_together(X,Y).

% Requirements of minimum amount of some substances in the final product.
required(Z*q(X)>=Y) :- have_to_use(X,Y,Z).
```

This problem allows us to show one of the greatest advantages of EZCSP system over ASP. It allows us to postpone the grounding, which is often the bottleneck of ASP algorithm. In particular, grounding is a problem when we have unknown variables which ranges over non-discrete domains, , such as $q(X)$ that ranges over $\mathbb{Q}$ in this problem. In order to encode similar problems in ASP, which deals only with integer variables, we have to set a “precision”, a fraction of the unit of measurement of the problem, and make $q(X)$ range only over integer multiples of this precision. For example, here we present an

ASP encoding of the basic blending problem (without extra requirements).

```
% In the input file, fact prec(N) states that the precision is set
% to 1/N of the unit of measurement.
units(1..X) :- prec(X).

% Maximum money considered: 2*budget.
money(1..X) :- X=Y*2*Z , max_money(Y), prec(Z).

% Choose a quantity for each substance.
1 {q(X,Y) : units(Y)} 1 :- subs(X).

% Compute total money spent and compare with budget.
tm_dot_prec(Z1) :- Z1=#sum{Y*Z : subs(X), q(X,Z), cost(X,Y)}, money(Z1).
:- X>Z*A , max_money(A), tm_dot_prec(X), prec(Z).

% Compute resulting components and compare with requirements.
qoc_dot_prec(Y,Z) :- Z=#sum{ A*B,X : q(X,A), tab(X,Y,B)} , comp(Y).
:- Z< X*Z1 , comp(Y), qoc_dot_prec(Y,Z), mi(Y,Z1), prec(X).
```

However, this approach has a problem: even with low precisions processing the grounded file is infeasible due to its dimensions. With higher precisions the grounding in itself becomes infeasible. For example consider the following simple input composed of 4 substances and 3 components, and a low, non acceptable precision of 1/10 of the unit of measurement.

```
subs(1..4).
cost(1,40). cost(2,30). cost(3,60). cost(4,30).

comp(1..3).
mi(1,100). mi(2,300). mi(3,80).

tab(1,1,20). tab(1,2,510). tab(1,3,50).
tab(2,1,320). tab(2,2,230). tab(2,3,25).
tab(3,1,60). tab(3,2,0). tab(3,3,180).
tab(4,1,160). tab(3,2,435). tab(4,3,90).

max_money(60).
prec(10).
```

On a computer with an Intel Core i5 processor at 2.5GHz, ASP solver CLINGO takes about 16s to find the first solution, while if we change the precision to 1/20 of the unit of measurement, the process takes about 518s. On the contrary, using the true-CASP encoding of the problem, the EZCSP system finds the first solution in less than 1 second, and with the variables ranging over rational numbers instead of over integer multiples of a fixed precision. The EZCSP system allows to exploit advantages of ASP solvers in encoding requirements, while avoiding the problem of grounding by leaving the treatment of rational quantities to the csp solver.

3.2 Scheduling Tasks

Here we present an approach to solve a modified job shop scheduling problem in constraint answer set programming using the EZCSP system. This classic problem concerns a set of jobs an a set of machines, each job comprises some operations, which must each be done on a different machine for different specified processing time, in a given job-dependent order. Each job has a release time and a due time to complete. The release time of a job is the arrival time for that job, while the due time is the time in which the job must be completed. A legal schedule is a schedule of job sequences on each machine such that each job's operation order is preserved, a machine can process at most one operation at one time, and different operations of the same job are not simultaneously processed on different machines.

This problem is suitable for encoding in the EZCSP language because it combines aspects of planning (determine the order of operation given dependencies between some of them) with aspects of scheduling. EZCSP allows to leverage ASP and CSP solvers for solving different parts of the problem at hand. Thus we can use ASP to encode the planning part and CSP to encode the scheduling.

The input file is composed of the following facts.

- $job(j, start, end)$., which states that j is a job whose release time is *start* and due time is *end*;
- $operation(J, O, P, M)$., which states that operation O of job J has a process time of P and requires machine M;
- $depend(J, O1, O2)$., which states that operation $O1$ of job J must occur before operation $O2$ of job J.

Here we present the true-CASP encoding of this problem.

```
cspdomain(fd).

% Set up topological sort.
1{first(J,O,P,M):operation(J,O,P,M)}1 :- job(J,R,D).
1{dep(J,O,O1):operation(J,O1,P1,M1), O!=O1}1 :- job(J,R,D),
    operation(J,O,P,M), not first(J,O,P,M).
tdep(J,O,O1) :- tdep(J,O,O2), tdep(J,O2,O1).    % Transitive dependence.
tdep(J,O,O1) :- dep(J,O,O1).
:- depend(J,O,O1), not tdep(J,O,O1).
:- not tdep(J,O,O1), not tdep(J,O1,O), operation(J,O,P,M),
    operation(J,O1,P1,M1), O!=O1.

% Start time of operation (J,O,P,M).
cspvar(st(J,O,P,M),R,D-P) :- operation(J,O,P,M), job(J,R,D).
```

```
% Each operation starts after the end time of the previous one
% in the topological sort.
required(st(J,O,P,M) >= P1+st(J,O1,P1,M1)) :- operation(J,O,P,M),
    operation(J,O1,P1,M1), dep(J,O,O1).

% Max one job per machine at any given time.
required((st(J,O,P,M) >= st(J1,O1,P1,M) + P1) \/
    (st(J1,O1,P1,M) >= st(J,O,P,M) + P)) :- operation(J,O,P,M),
    operation(J1,O1,P1,M), J!=J1.
```

Bibliography

[1] Marcello Balduccini and Yuliya Lierler. Constraint answer set solver EZCSP and why integration schemas matter. *Theory and Practice of Logic Programming*, 17(4): 462–515, 2017.

[2] Marcello Balduccini, Michael Gelfond, and Monica Nogueira. Answer set based design of knowledge systems. *Annals of Mathematics and Artificial Intelligence*, 47: 183–219, 2006.

[3] Gerhard Brewka, Thomas Eiter, and Miros Truszczyński. Answer Set Programming at a glance. *Communications of the ACM*, 54(12):92–103, 2011.

[4] Omar EL Khatib. Job shop scheduling under Answer Set Programming environment. *International Journal of Engineering and Innovative Technology*, 5:36–41, 2015.

[5] Esra Erdem and Ferhan Türe. Efficient haplotype inference with Answer Set Programming. In *Proceedings of the 23rd National Conference on Artificial Intelligence*, volume 1, pages 436–441. AAAI Press, 2008.

[6] Martin Gebser, Torsten Schaub, and Sven Thiele. *GrinGo: A New Grounder for Answer Set Programming*, pages 266–271. Springer Berlin Heidelberg, 2007.

[7] Martinì Gebser, Max Ostrowski, and Torsten Schaub. *Constraint Answer Set Solving*, pages 235–249. Springer Berlin Heidelberg, 2009.

[8] Yuliya Lierler. Relating Constraint Answer Set Programming languages and algorithms. *Artificial Intelligence*, 207C:1–22, 2014.

[9] Yuliya Lierler and Benjamin Susman. On relation between Constraint Answer Set Programming and Satisfiability Modulo Theories. *CoRR*, abs/1702.07461, 2017.

[10] Veena S. Mellarkod, Michael Gelfond, and Yuanlin Zhang. Integrating answer set programming and constraint logic programming. *Annals of Mathematics and Artificial Intelligence*, 53(1):251–287, 2008.

[11] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving SAT and SAT modulo theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). *J. ACM*, 53(6):937–977, 2006.

[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)*, volume 52 of *OpenAccess Series in Informatics (OASIcs)*, pages 1:1–1:15. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016.