

# Constraint Based Strategies

Claude Kirchner<sup>1</sup>, Florent Kirchner<sup>2</sup>, and Hélène Kirchner<sup>1</sup>

<sup>1</sup> INRIA

Centre de Recherche  
INRIA Bordeaux - Sud-Ouest  
351, cours de la Libération,  
33405 Talence Cedex France

<sup>2</sup> INRIA

Centre de Recherche  
INRIA Rennes - Bretagne Atlantique  
Campus universitaire de Beaulieu,  
35042 Rennes Cedex France

**Abstract.** Numerous computational and deductive frameworks use the notion of strategy to guide reduction and search space exploration, making the macro scale control of micro operations an explicit object of interest. In recent works, abstract strategies have been defined in extension but also intensionally. In this paper we complete these views with a new declarative approach based on constraints, which are used to model the different parts of a strategy. This procedure allows us to express elaborate strategies in a declarative and reusable way.

## 1 Introduction

The fundamental complementarity between deduction and computation, as emphasized in particular in deduction modulo [8], gives now rise to a completely new generation of proof assistants where customized deductions are performed modulo appropriate and user definable computations. This collusion of deduction and computation in next-generation proof assistants has inspired our recent work at providing a uniform definition for strategies, starting from a rule-based view point [12].

For term rewriting, reduction strategies study which expressions should be selected for evaluation and which rules should be applied. These choices are usually made to increase the efficiency of evaluation but may affect fundamental properties of computations such as confluence or (non-)termination. Programming languages like TOM [3], ELAN [4], Maude [20] and Stratego [22] allow for the explicit definition of the evaluation strategy.

Deductive environments include interactive proof assistants, automated theorem provers, proof checkers, and logical frameworks. For these systems, strategies (also called tacticals in some contexts) are used for proof search and proof planning, restriction of search spaces, specification of control components, combination of different proof techniques and computation paradigms, or meta-level

programming in reasoning systems. Interest for these fundamental elements of deductive systems has been growing in recent years: see for instance [7,10,6,17] in the field of procedural proof assistants.

One main difficulty in defining strategies, both for deduction or computation, is precisely the specification of the strategy that the user wants the system to apply. Indeed, a strategy describes paths in the search or the reduction space that are complex objects, depending of the initial goal, but also of past and future of the current state exploration. To deal with this difficulty, we have first defined abstract strategies *in extension* [12] so that the strategy object, as well as its basic properties, can be studied thoroughly and precisely. A second step then consisted in providing a language to help describe the strategies of interest in an operational fashion. This approach is developed in [5] under the name of *intensional strategies*.

In what follows we develop another point of view, using the modeling capabilities of constraints to provide a declarative way of defining strategies. One of the main interest of this new approach is that the constraints solving process is taken care of by the environment, releasing the user of making procedural description of search or reduction paths. The constraints then become very powerful tools, well-suited to model the complexity, non-determinism, and infiniteness of strategies. By calling upon constraints either symbolic or numeric, the user can focus on the properties and correctness of the strategies he wants to define. Moreover, this declarative approach being free from circumstantial details such as an execution context, it bears the promise of both portability and reusability.

Based on the previous works of [12,5], the next section synthesizes the frameworks of abstract reduction strategies and of intensional strategies. We then define derivation schemas, a way to algebraize the notion of abstract strategies. Derivation variables are in particular introduced and are one of the main schematisation tool. These schemas can then be instantiated by the solutions of appropriate constraints via the adapted notion of substitution. We illustrate our approach by using progressively complex examples, making use mainly in this paper of equality constraints.

We assume the reader familiar with term rewriting and basic notions of term algebra as typically defined in [2] or [13].

## 2 Abstract reduction strategies

In this section, we recall from previous works, the general notion of abstract reduction systems already presented in [21,12,5], as well as the definitions of abstract strategies given in [12] and intensional strategies given in [5].

### 2.1 Abstract reduction systems

When abstracting the notion of strategies, one important preliminary remark is that we need to start from an appropriate notion of *abstract reduction system* (ARS) based on the notion of oriented labeled graph instead of binary relation.

This is due to the fact that, speaking of derivations, we need to make a difference between “being in relation” and “being connected”. Typically modeling ARS as relations as in [2] allows us to say that, *e.g.*,  $a$  and  $b$  are in relation but not that there may be several different ways to derive  $b$  from  $a$ . Consequently, we need to use a more expressive approach, similar to the one proposed in [21, Chapter 8], based on a notion of oriented graph. Our definition is similar to the one given in [12] with the slight difference that we make more precise the definition of steps and labels. Similarly to the step-based definition of an abstract reduction system of [21], this definition that identifies the reduction steps avoids the so-called *syntactic accidents* [18], related to different but indistinguishable derivations.

**Definition 1 (Abstract reduction system).** *Given a countable set of objects  $\mathcal{O}$  and a countable set of labels  $\mathcal{L}$  mutually disjoint, an abstract reduction system (ARS) is a triple  $(\mathcal{O}, \mathcal{L}, \Gamma)$  such that  $\Gamma$  is a functional relation from  $\mathcal{O} \times \mathcal{L}$  to  $\mathcal{O}$ : formally,  $\Gamma \subseteq \mathcal{O} \times \mathcal{L} \times \mathcal{O}$  and  $(a, \phi, b_1) \in \Gamma$  and  $(a, \phi, b_2) \in \Gamma$  implies  $b_1 = b_2$ .*

*The tuples  $(a, \phi, b) \in \Gamma$  are called steps and are often denoted by  $a \xrightarrow{\phi} b$ . We say that  $a$  is the source of  $a \xrightarrow{\phi} b$ ,  $b$  its target and  $\phi$  its label. Moreover, two steps are composable if the target of the former is the source of the latter.*

The condition that  $\Gamma$  is a functional relation implies that an ARS is a particular case of a labeled transition system. Actually, labels characterize the way an object is transformed: given an object and a transformation, there is at most one object resulting from the transformation applied to this particular object.

The next definitions can be seen as a renaming of usual ones in graph theory. Their interest is to allow us to define uniformly derivations and strategies in different contexts.

**Definition 2 (Derivation).** *Given an abstract reduction system  $\mathcal{A} = (\mathcal{O}, \mathcal{L}, \Gamma)$  we call derivation over  $\mathcal{A}$  any sequence  $\pi$  of steps  $((t_i, \phi_i, t_{i+1}))_{i \in \mathfrak{S}}$  for any right-open interval  $\mathfrak{S} \subseteq \mathbb{N}$  starting from 0. If  $\mathfrak{S}$  contains at least one element, then:*

- *$\text{Dom}(\pi) = t_0$  is called the source (or domain) of  $\pi$ ,*
- *$l(\pi) = (\phi_i)_{i \in \mathfrak{S}}$  is a sequence called label of  $\pi$ ,*
- *For any non empty subinterval  $\mathfrak{S}' \subseteq \mathfrak{S}$ ,  $\pi' = ((t_i, \phi_i, t_{i+1}))_{i \in \mathfrak{S}'}$  is a factor of  $\pi$ . If  $\mathfrak{S}'$  contains 0, then  $\pi'$  is a prefix of  $\pi$ . If  $\mathfrak{S}' \neq \mathfrak{S}$ ,  $\pi'$  is a strict factor (prefix).*

*If  $\mathfrak{S}$  is finite, it has a smallest upper bound denoted by  $n_{\mathfrak{S}}$  or simply  $n$  and then:*

- *$\text{Im}(\pi) = t_n$  is called the target (or image) of  $\pi$ ,*
- *$|\pi| = \text{card}(\mathfrak{S})$  is called the length of  $\pi$ ,*

*In such a case,  $\pi$  is said to be finite and is also denoted by  $\pi = (t_0, l(\pi), t_n)$  or  $t_0 \xrightarrow{l(\pi)} t_n$ . The sequence containing no step is called empty derivation and is denoted by  $\Lambda$  and by convention  $l(\Lambda) = \epsilon$  where  $\epsilon$  is the empty sequence of elements of  $\mathcal{L}$ .*

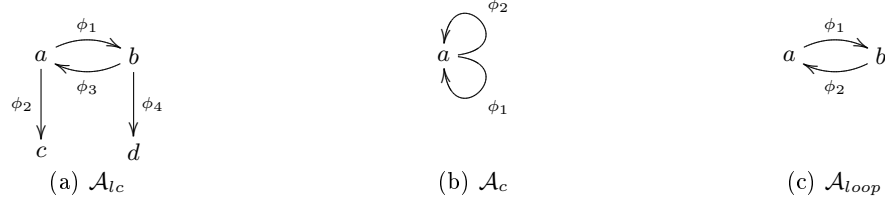
We denote by  $\Gamma_{\mathcal{A}}^{\omega}$  (resp.  $\Gamma_{\mathcal{A}}^*$ , resp.  $\Gamma_{\mathcal{A}}^+$ ) the set of all derivations (resp. finite, resp. non-empty and finite) over  $\mathcal{A}$ .

Note that a step may be considered as a derivation of length 1.

**Definition 3 (Composable derivations).** *Two derivations over a same abstract reduction system  $\mathcal{A} = (\mathcal{O}, \mathcal{L}, \Gamma)$ , say  $\pi_1 = ((t_i, \phi_i, t_{i+1}))_{i \in \mathfrak{S}_1}$  and  $\pi_2 = ((u_i, \phi_i, u_{i+1}))_{i \in \mathfrak{S}_2}$  are composable iff either one of the derivation is empty or  $\mathfrak{S}_1$  is finite and then  $t_{n_1} = u_0$  where  $n_1$  is the smallest upper bound of  $\mathfrak{S}_1$ . In such a case, the composition of  $\pi_1$  and  $\pi_2$  is the unique derivation  $\pi = ((v_i, \phi_i, v_{i+1}))_{i \in \mathfrak{S}}$  denoted by  $\pi = \pi_1 \pi_2$  such that for all  $j < |\pi_1|$ ,  $v_j = t_j$  and for all  $j \geq |\pi_1|$ ,  $v_j = u_{j-|\pi_1|}$ .*

The composition is associative and has a neutral element which is  $\Lambda$ . Adopting the product notation, we denote  $\prod_{i=1}^n \pi_i = \pi_1 \dots \pi_n$ ,  $\pi^n = \prod_{i=1}^n \pi$  and  $\pi^{\omega} = \prod_{i \in \mathbb{N}} \pi$ .

Like for graphs and labeled transition systems, in order to support intuition, we will often use the obvious graphical representation to denote the corresponding ARS.



**Fig. 1.** Graphical representation of abstract reduction systems

*Example 1 (Abstract reduction systems).* The abstract reduction system

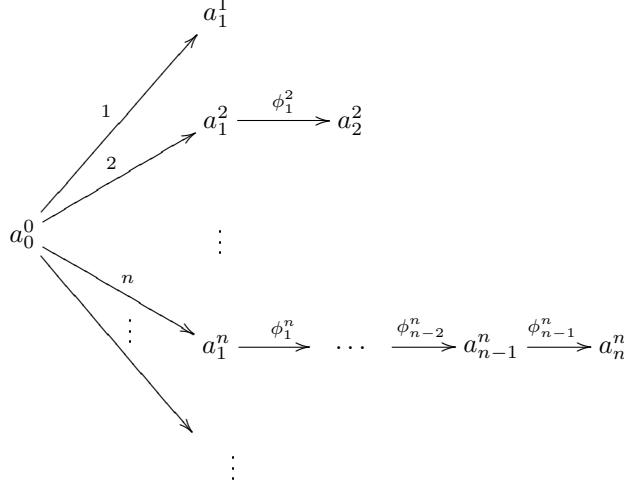
$$\mathcal{A}_{lc} = (\{a, b, c, d\}, \{\phi_1, \phi_2, \phi_3, \phi_4\}, \{(a, \phi_1, b), (a, \phi_2, c), (b, \phi_3, a), (b, \phi_4, d)\})$$

with a finite number of objects but infinite derivations is depicted in Figure 1(a).  $\Gamma_{\mathcal{A}_{lc}}^{\omega}$  contains for instance  $\pi_1, \pi_1 \pi_3, \pi_1 \pi_4, \pi_1 \pi_3 \pi_1, (\pi_1 \pi_3)^n, (\pi_1 \pi_3)^{\omega}, \dots$ , with  $\pi_1 = (a, \phi_1, b)$ ,  $\pi_2 = (a, \phi_2, c)$ ,  $\pi_3 = (b, \phi_3, a)$ ,  $\pi_4 = (b, \phi_4, d)$ .

Another abstract reduction system with an infinite set of objects and a countable infinite set of derivations starting from a same source is  $\mathcal{A}_{ex} =$

$$(\{a_i^j \mid i, j \in \mathbb{N}\}, \mathbb{N} \cup \{\phi_i^j \mid 1 \leq i < j\}, \{(a_0^0, j, a_1^j) \mid 1 \leq j\} \cup \{(a_i^j, \phi_i^j, a_{i+1}^j) \mid 1 \leq i < j\})$$

whose relation can be depicted by:



## 2.2 Abstract strategies

We use a general definition slightly different from the one used in [21, Chapter 9]. This approach has already been proposed in [12] and improved in [5].

**Definition 4 (Abstract Strategy).** *Given an ARS  $\mathcal{A}$ , an abstract strategy  $\zeta$  over  $\mathcal{A}$  is a subset of derivations of  $\Gamma_{\mathcal{A}}^{\omega}$ .*

A strategy can be a finite or an infinite set of derivations, and the derivations themselves can be finite or infinite in length.

An abstract strategy over an abstract reduction system  $\mathcal{A} = (\mathcal{O}, \mathcal{L}, \Gamma)$  induces a (partial) function from  $\mathcal{O}$  to  $2^{\mathcal{O}}$ . This functional point of view has been already proposed in [4]; we just briefly recall it in our formalism.

The *domain* of a strategy  $\zeta$  is the set of objects that are source of a derivation in  $\zeta$ :

$$Dom(\zeta) = \bigcup_{\pi \in \zeta} Dom(\pi)$$

The application of a strategy is defined (only) on the objects of its domain. The application of a strategy  $\zeta$  on  $a \in Dom(\zeta)$  is denoted  $\zeta a$  and is defined as the set of all objects that can be reached from  $a$  using a finite derivation in  $\zeta$ :

$$\zeta a = \{Im(\pi) \mid \pi \in \zeta, \pi \text{ finite and } Dom(\pi) = a\}$$

If  $a \notin Dom(\zeta)$  we say that  $\zeta$  *fails* on  $a$  (either  $\zeta$  contains no derivation or it contains no derivation of source  $a$ ).

If  $a \in Dom(\zeta)$  and  $\zeta a = \emptyset$ , we say that the strategy  $\zeta$  is *indeterminate* on  $a$ . In fact,  $\zeta$  is indeterminate on  $a$  if and only if  $\zeta$  contains no finite derivation starting from  $a$ .

*Example 2 (Strategies).* Let us consider again the abstract reduction system  $\mathcal{A}_{lc}$  presented in Example 1 and define the following strategies:

1. The strategy  $\zeta_u = \Gamma_{\mathcal{A}_{lc}}^\omega$ , also called the *Universal* strategy [12] (w.r.t.  $\mathcal{A}_{lc}$ ), contains all the derivations of  $\mathcal{A}_{lc}$ . We have  $\zeta_u a = \zeta_u b = \{a, b, c, d\}$  and  $\zeta_u$  fails on  $c$  and  $d$ .
2. The strategy  $\zeta_f = \emptyset$ , also called *Fail*, contains no derivation and thus fails on any  $x \in \{a, b, c, d\}$ .
3. For the strategy  $\zeta_c = \left\{ \left( a \xrightarrow{\phi_1 \phi_3} a \right)^n a \xrightarrow{\phi_2} c \mid n \geq 0 \right\}$  no matter which derivation is considered, the object  $a$  eventually reduces to  $c$ :  $\zeta_c a = \{c\}$ .  $\zeta_c$  fails on  $b$ ,  $c$  and  $d$ .
4. The strategy  $\zeta_\omega = \left\{ \left( a \xrightarrow{\phi_1 \phi_3} a \right)^\omega \right\}$  is not terminating on  $a$  and fails on  $b$ ,  $c$  and  $d$ .

The so-called *Universal* and *Fail* strategies introduced in Example 2 can be obviously defined over any abstract reduction system.

### 2.3 Intensional Strategies

In [5], the notion of intensional strategy is introduced. The essence of the idea is that strategies are considered as a way of constraining and guiding the steps of a reduction. So at any step in a derivation, it should be possible to say whether a contemplated next step obeys the strategy  $\zeta$ . In order to take into account the past derivation steps to decide the next possible ones, the history of a derivation has to be memorized and available at each step. Let us first introduce the notion of traced-object where each object memorizes how it has been reached.

**Definition 5 (Traced-object).** *Given a countable set of objects  $\mathcal{O}$  and a countable set of labels  $\mathcal{L}$  mutually disjoint, a traced-object is a pair  $[\alpha]a$  where  $\alpha$  is a sequence of elements of  $\mathcal{O} \times \mathcal{L}$  called trace or history .*

The set of traces may be considered as a monoid  $((\mathcal{O} \times \mathcal{L})^*, \odot)$  generated by  $(\mathcal{O} \times \mathcal{L})$  and whose neutral element is denoted by  $\Lambda$ .

**Definition 6 (Traced object compatible with an ARS).** *A traced-object  $[\alpha]a$  is compatible with  $\mathcal{A} = (\mathcal{O}, \mathcal{L}, \Gamma)$  iff  $\alpha = ((a_i, \phi_i))_{i \in \mathfrak{S}}$  for any right-open interval  $\mathfrak{S} \subseteq \mathbb{N}$  starting from 0 and  $a = a_n$  and for all  $i \in \mathfrak{S}$ ,  $(a_i, \phi_i, a_{i+1}) \in \Gamma$ . In such a case, we denote by  $\llbracket \alpha \rrbracket$  the derivation  $((a_i, \phi_i, a_{i+1}))_{i \in \mathfrak{S}}$  and by  $\mathcal{O}^{[\mathcal{A}]}$  the set of traced objects compatible with  $\mathcal{A}$ . Moreover, we define an equivalence relation  $\sim$  over  $\mathcal{O}^{[\mathcal{A}]}$  as follows:  $[\alpha]a \sim [\alpha']a'$  iff  $a = a'$ . We naturally have  $\mathcal{O}^{[\mathcal{A}]} / \sim = \mathcal{O}$ .*

An intensional strategy, as defined in [5], chooses the next step not only regarding the current object (or state), but also taking the history of objects into account.

**Definition 7 (Intensional strategy (with memory)).** An intensional strategy over an abstract reduction system  $\mathcal{A} = (\mathcal{O}, \mathcal{L}, \Gamma)$  is a partial function  $\lambda$  from  $\mathcal{O}^{[\mathcal{A}]}$  to  $2^\Gamma$  such that for every traced object  $[\alpha] a$ ,  $\lambda([\alpha] a) \subseteq \{\pi \in \Gamma \mid \text{Dom}(\pi) = a\}$ .

An intensional strategy naturally generates an abstract strategy, as follows [5].

**Definition 8 (Extension of an intensional strategy).** Let  $\lambda$  be an intensional strategy over an abstract reduction system  $\mathcal{A} = (\mathcal{O}, \mathcal{L}, \Gamma)$ . The extension of  $\lambda$  is the abstract strategy  $\zeta_\lambda$  consisting of the following set of derivations:

$$\pi = ((a_i, \phi_i, a_{i+1}))_{i \in \mathfrak{S}} \in \zeta_\lambda \quad \text{iff} \quad \forall j \in \mathfrak{S}, \quad (a_j, \phi_j, a_{j+1}) \in \lambda([\alpha] a_j)$$

where  $\alpha = ((a_i, \phi_i))_{i \in \mathfrak{S}}$ .

This extension may obviously contain infinite derivations; in such a case it also contains all the finite derivations that are prefixes of the infinite ones. Indeed, it is easy to see from Definition 8 that the extension of an intensional strategy is closed under taking prefixes.

It has been shown in [5] that the set of finite derivations generated by an intensional strategy  $\lambda$  can be constructed inductively as follows. Given an intensional strategy with memory  $\lambda$  over an abstract reduction system  $\mathcal{A} = (\mathcal{O}, \mathcal{L}, \Gamma)$ , the finite support of its extension is the subset of  $\zeta_\lambda$  made of only finite derivations. This is again an abstract strategy denoted  $\zeta_\lambda^{<\omega}$  inductively defined as follows:

- $\forall [A] a \in \mathcal{O}^{[A]}, \lambda([A] a) \subseteq \zeta_\lambda^{<\omega}$ ,
- $\forall \alpha$  s.t.  $\pi = [[\alpha]] \in \zeta_\lambda^{<\omega}$  and  $\pi' \in \lambda([\alpha] \text{Im}(\pi))$ ,  $\pi\pi' \in \zeta_\lambda^{<\omega}$

Each intensional strategy  $\lambda$  over  $\mathcal{A} = (\mathcal{O}, \mathcal{L}, \Gamma)$  induces an ARS  $\mathcal{A}_\lambda = (\mathcal{O}^{[\mathcal{A}]}, \mathcal{L}, \Gamma_\lambda)$  such that  $([\alpha] a, \phi, [\alpha \odot (a, \phi)] b) \in \Gamma_\lambda$  iff  $(a, \phi, b) \in \lambda([\alpha] a)$ .

This is denoted by:  $[\alpha] a \xrightarrow{\phi}_\lambda [\alpha \odot (a, \phi)] b$ .

A special case are memoryless strategies, where the function  $\lambda$  does not depend on the history of the objects. This is the case of many strategies used in rewriting systems, as shown in the next examples.

*Example 3.* Let us define the following strategies:

- The intensional strategy  $\lambda_u$  defined on all objects in  $\mathcal{O}$  such that for any object  $a \in \mathcal{O}$ ,  $\lambda_u(a) = \{\pi \mid \pi \in \Gamma, \text{Dom}(\pi) = a\}$  obviously generates the Universal strategy  $\zeta_u$  (of Example 2).
- The intensional strategy  $\lambda_f$  defined on no object in  $\mathcal{O}$  generates the Fail strategy  $\zeta_f$  (of Example 2).
- Let us consider an abstract reduction system  $\mathcal{A}$  where objects are terms, reduction of  $\Gamma$  is term rewriting with a rewrite rule in the rewrite system, and labels are positions where the rewrite rules are applied. Let us consider an order  $<$  on the labels which is the prefix order on positions. Then the

intensional strategy that corresponds to innermost rewriting is  $\lambda_{inn}$  such that  $\lambda_{inn}(t) = \{\pi : t \xrightarrow{p} t' \mid p = \max(\{p' \mid t \xrightarrow{p'} t' \in \Gamma\})\}$ . When a lexicographic order is used, the classical *rightmost-innermost* strategy is obtained.

However the following examples of strategies cannot be expressed without the knowledge of the history and illustrate the interest of traced objects.

*Example 4.*

- The intensional strategy that restricts the derivations to be of bounded length  $k$  makes use of the size of the trace  $\alpha$ , denoted  $|\alpha|$ :

$$\lambda_{tk}([\alpha] a) = \{\pi \mid \pi \in \Gamma, \text{Dom}(\pi) = a, |\alpha| < k - 1\}$$

- The strategy that alternates reductions from a set (of steps)  $\Gamma_1$  with reductions from a set  $\Gamma_2$  can be generated by the following intensional strategy:

$$\begin{aligned} \lambda_{\Gamma_1; \Gamma_2}([A] a) &= \{\pi_1 \mid \pi_1 \in \Gamma_1, \text{Dom}(\pi_1) = a\} \\ \lambda_{\Gamma_1; \Gamma_2}([\alpha' \odot (u, \phi')] a) &= \{\pi_1 \mid \pi_1 \in \Gamma_1, \text{Dom}(\pi_1) = a\} \text{ if } u \xrightarrow{\phi'} a \in \Gamma_2 \\ \lambda_{\Gamma_1; \Gamma_2}([\alpha' \odot (u, \phi')] a) &= \{\pi_2 \mid \pi_2 \in \Gamma_2, \text{Dom}(\pi_2) = a\} \text{ if } u \xrightarrow{\phi'} a \in \Gamma_1 \end{aligned}$$

However, as noticed in [5], the fact that intensional strategies generate only prefix closed abstract strategies prevents us from computing abstract strategies that look straightforward like the ones in the next example.

*Example 5.* We consider again the abstract reduction system  $\mathcal{A}_{lc}$  and a strategy reduced to only one derivation  $\zeta = \{a \xrightarrow{\phi_1} b \xrightarrow{\phi_3} a \xrightarrow{\phi_2} c\}$ .  $\zeta$  cannot be computed by an intensional strategy  $\lambda$  built as before since  $\zeta_\lambda^{<\omega}$  would contain too many derivations, namely all prefixes of the derivation:

$$[A] a \xrightarrow{\phi_1}_\lambda [(a, \phi_1)] b \xrightarrow{\phi_3}_\lambda [(a, \phi_1) \odot (b, \phi_3)] a \xrightarrow{\phi_2}_\lambda [(a, \phi_1) \odot (b, \phi_3) \odot (a, \phi_2)] c$$

In a similar way, there is no intensional strategy that can generate a set of derivations of length exactly  $k$ .

The next section introduces another approach to define abstract strategies that in particular avoids this problem.

### 3 Constraints and strategies

Since a strategy is a set of derivations, this set can be described in extension, as done in Example 2. But such definitions are not so convenient especially when the set is infinite. In this section, we make use of constraints as the basic language construction to describe strategies in a compact way. The schematization power of constraints is universally used in informatics and mathematics. More specifically, constraints have been extensively used in constraint logic programming [9],

rewriting and deduction [14], and constraint solving itself [11] to mention just a few..

Since abstract reduction systems may involve infinite sets of objects, of reduction steps and of derivations, we can use constraints at different levels that can be ultimately combined: (i) to describe the objects occurring in a derivation (ii) to describe via the labels the requirements on the steps of reductions (iii) to describe the structure of the derivation itself (iv) to express requirements on the histories.

The framework we develop now allows us to define a strategy  $\zeta$  as all instances  $\sigma(S)$  of a derivation schema  $S$  such that  $\sigma$  is solution of a constraint  $C$  involving derivation variables in  $\mathcal{X}_{\mathcal{D}}$ , typically denoted by  $D$ , object variables in  $\mathcal{X}_{\mathcal{O}}$ , typically denoted  $X$ , and label variables in  $\mathcal{X}_{\mathcal{L}}$ , typically denoted  $L$ . Similar schemes could be done for history but are left out of this paper for simplification purpose. Therefore, in order to represent the objects is a generic way, we make use of (open) first-order terms  $\mathcal{T}(\mathcal{F}, \mathcal{X}_{\mathcal{O}})$  over a signature  $\mathcal{F}$ .

**Definition 9.** *A derivation schema is a derivation on an abstract reduction system  $\mathcal{A} = (\mathcal{T}(\mathcal{F}, \mathcal{X}_{\mathcal{O}}), \mathcal{L} \cup \mathcal{X}_{\mathcal{L}}, \Gamma \cup \mathcal{X}_{\mathcal{D}})$ . It is a sequence of steps, written either  $\pi = ((a_i, \phi_i, a_{i+1}))_{i \in \mathbb{S}}$  or  $\pi = a_j \xrightarrow{\phi_j} a_{j+1} \xrightarrow{\phi_{j+1}} \dots a_i \xrightarrow{\phi_i} a_{i+1}$  when  $\pi$  is finite, and where each  $a_i$  is a term of  $\mathcal{T}(\mathcal{F}, \mathcal{X}_{\mathcal{O}})$ , each  $\phi_i$  may be a label or a variable in  $\mathcal{X}_{\mathcal{L}}$ , each step  $(a, \phi, b)$  may be a step of  $\Gamma$  or a variable in  $\mathcal{X}_{\mathcal{D}}$ , each sequence  $\pi$  may be a sequence of steps or a variable in  $\mathcal{X}_{\mathcal{D}}$ .*

Notice that in this definition, we may also define term based labels and even derivations. Again, we choose not to get too general in this paper to keep the main ideas more apparent.

Notice also that, up to the labels, derivation schema are terms build over the binary symbol  $\rightarrow$  with terms of  $\mathcal{T}(\mathcal{F}, \mathcal{X}_{\mathcal{O}})$  as leaves. As we will see, this view is quite fruitful and require to assume from now on that the labeled arrow symbols are “associative” with neutral element  $A$ , in the following sense:

$$a_1 \xrightarrow{\phi_1} (a_2 \xrightarrow{\phi_2} a_3) = (a_1 \xrightarrow{\phi_1} a_2) \xrightarrow{\phi_2} a_3 \quad (1)$$

$$A \xrightarrow{\phi} a = a \xrightarrow{\phi} A = a \quad (2)$$

We have now to make clear how derivation schema can be instantiated to express derivations.

**Definition 10.** *Given a derivation schema  $S$  on an abstract reduction system  $\mathcal{A} = (\mathcal{T}(\mathcal{F}, \mathcal{X}_{\mathcal{O}}), \mathcal{L} \cup \mathcal{X}_{\mathcal{L}}, \Gamma \cup \mathcal{X}_{\mathcal{D}})$ , a substitution  $\sigma$  is composed of substitutions  $\sigma_{\mathcal{O}}$  of  $\mathcal{T}(\mathcal{F}, \mathcal{X}_{\mathcal{O}})$ ,  $\sigma_{\mathcal{L}}$  from  $\mathcal{X}_{\mathcal{L}}$  to  $\mathcal{L} \cup \mathcal{X}_{\mathcal{L}}$  and  $\sigma_{\Gamma}$  from  $\mathcal{X}_{\mathcal{D}}$  to  $\Gamma \cup \mathcal{X}_{\mathcal{D}}$ . The image of  $S$  by  $\sigma$  is given by the instance of each of its step:*

if  $a, b \in \mathcal{T}(\mathcal{F}, \mathcal{X}_{\mathcal{O}})$ ,  $\phi \in \mathcal{L} \cup \mathcal{X}_{\mathcal{L}}$ ,  $D, D' \in \Gamma \cup \mathcal{X}_{\mathcal{D}}$ ,  $\pi = ((a_i, \phi_i, a_{i+1}))_{i \in \mathbb{S}}$ ,

$$\begin{aligned}\sigma(a, \phi, b) &= (\sigma_{\mathcal{O}}(a) \xrightarrow{\sigma_{\mathcal{L}}(\phi)} \sigma_{\mathcal{O}}(b)) \\ \sigma(D, \phi, b) &= (\sigma_{\Gamma}(D) \xrightarrow{\sigma_{\mathcal{L}}(\phi)} \sigma_{\mathcal{O}}(b)) \\ \sigma(a, \phi, D) &= (\sigma_{\mathcal{O}}(a) \xrightarrow{\sigma_{\mathcal{L}}(\phi)} \sigma_{\Gamma}(D)) \\ \sigma(D, \phi, D') &= (\sigma_{\Gamma}(D) \xrightarrow{\sigma_{\mathcal{L}}(\phi)} \sigma_{\Gamma}(D')) \\ \sigma(\pi) &= (\sigma(a_i, \phi_i, a_{i+1}))_{i \in \mathbb{S}}\end{aligned}$$

To describe sets of derivations, it is convenient to use notations like  $\{x \rightarrow D \mid D \in \zeta\}$  where by convention  $x \rightarrow D$  is exactly  $x$  when  $\zeta = \emptyset$ .

**Definition 11.** A derivation constraint over  $(\mathcal{T}(\mathcal{F}, \mathcal{X}_{\mathcal{O}}), \mathcal{L} \cup \mathcal{X}_{\mathcal{L}}, \Gamma \cup \mathcal{X}_{\Gamma})$  is a formula  $C$  involving free variables variables of  $\mathcal{X}_{\mathcal{O}}$ ,  $\mathcal{X}_{\mathcal{L}}$  and  $\mathcal{X}_{\Gamma}$ . We denote  $Sol(C)$  the set of solutions  $\sigma$  of  $C$ , that are ground instances of variables of  $C$ .

The previous definition is on purpose quite general and leaves open the language on which the formulae are built. This can be for instance equality constraints, membership constraints or ordering constraints but also anti-pattern constraints like in [15] as well as first-order constraints or higher-order constraints involving functional variables, all these possibly occurring modulo some congruence typically defined by an equational theory like associativity or commutativity.

In order to make some syntactic evidence that a given formula is considered as a constraint, the predicate symbols appearing in such a constraint are distinguished with a question mark like in  $X + Y =? Z + X$ .

**Definition 12.** The abstract strategy schematized by  $(S \mid C)$ , where  $S$  is a derivation schema and  $C$  a derivation constraint over an abstract reduction system  $\mathcal{A} = (\mathcal{O} \cup \mathcal{X}_{\mathcal{O}}, \mathcal{L} \cup \mathcal{X}_{\mathcal{L}}, \Gamma \cup \mathcal{X}_{\Gamma})$ , is the subset  $\zeta$  of derivations of  $\Gamma_{(\mathcal{O}, \mathcal{L}, \Gamma)}^{\omega}$  defined as

$$\zeta = \{\sigma(S) \mid \sigma \in Sol(C)\}$$

Note that for any (ground) abstract reduction system  $\mathcal{A} = (\mathcal{O}, \mathcal{L}, \Gamma)$ , the universal strategy, which corresponds to the set of all derivations can be (trivially) described as  $(D \mid D \in? \Gamma_{\mathcal{A}}^{\omega})$ .

In order to relate these definitions with the previous concepts, we may point out that if we consider the abstract strategy  $\zeta_{\lambda}$  generated by the intensional strategy  $\lambda$ ,  $\zeta_{\lambda}$  can be described as

$$(D \mid D \in? \Gamma_{\mathcal{A}}^{\omega} \wedge [(X, L, Y) \in? D \Rightarrow (X, L, Y) \in? \lambda(X)])$$

To illustrate the expressivity of strategies with constraints, let us begin with an example of an abstract reduction system on terms, and remind the classical fact that if the symbol  $\cdot$  is associative, then the equation  $x \cdot a =?_A a \cdot x$  has an infinite set of solutions  $\{x \mapsto a^n \mid n \in \mathbb{N}\}$  (where as usual we use the notation  $a^n = a \cdot a^{n-1}$ ,  $a$  is a constant and  $x$  is a variable).

*Example 6 (Simple constraints on terms).* The infinite set of derivations of length one that transform  $a$  into  $f(a^n)$  is simply described by:

$$(a \rightarrow f(X) \mid X \cdot a =_A^? a \cdot X)$$

But it is of course also interesting to use constraints to describe the structure of derivations. In this context a simple and useful remark is that, as mentioned above, the arrow symbols in derivation can be considered as associative, so that the derivations  $a \rightarrow (b \rightarrow c)$  and  $(a \rightarrow b) \rightarrow c$  are equivalent.

*Example 7 (Simple constraints on derivations).* Let us consider the ARS  $\mathcal{A}_{lc}$  of Example 1. Then, the same equation as above allows us to describe  $\zeta_{loop_1}$ , the infinite set of derivations that have a finite cycle on  $a \rightarrow b$ :

$$(D \mid D \xrightarrow{\phi_3} (a \xrightarrow{\phi_1} b) =_A^? (a \xrightarrow{\phi_1} b) \xrightarrow{\phi_3} D)$$

that is (omitting obvious labels for simplicity)

$$\{a \rightarrow b, (a \rightarrow b) \rightarrow (a \rightarrow b), (a \rightarrow b)^3, \dots\} = \{(a \rightarrow b)^n \mid n \geq 0\}.$$

Since we have no restriction on linearity, we may do it twice as in

$$(D \xrightarrow{L} D \mid D \xrightarrow{L} (a \xrightarrow{\phi_1} b) =_A^? (a \xrightarrow{\phi_1} b) \xrightarrow{\phi_3} D)$$

(notice here the use of the label variable).

Then strategies like  $\zeta_c = \left\{ \left( a \xrightarrow{\phi_1 \phi_3} a \right)^n \xrightarrow{\phi_2} c \mid n \geq 0 \right\}$  given in Example 2, can be simply expressed in the following way:

$$(D \xrightarrow{\phi_3} a \xrightarrow{\phi_2} c \mid D \xrightarrow{\phi_3} (a \xrightarrow{\phi_1} b) =_A^? (a \xrightarrow{\phi_1} b) \xrightarrow{\phi_3} D).$$

*Example 8.* In order to insert an element  $a$  at all possible positions into a given derivation  $\pi$ , we can use again an equational constraint modulo associativity and write :

$$(D \rightarrow a \rightarrow D' \mid D \rightarrow D' =_A^? \pi).$$

*Example 9.* To illustrate more complex constraints, let us express the set of derivations  $\mathcal{A}_{ex}$  of Example 1 as follows:

$$(a_0^0 \xrightarrow{L} X \xrightarrow{L'} D \mid X \in^? \mathcal{O} \wedge L \in^? \mathbb{N} \wedge L' \in^? \mathcal{L} \wedge |X \rightarrow D| =^? L \wedge D \in^? \Gamma_{\mathcal{A}_{ex}}^*)$$

where  $|X \rightarrow D|$  denotes the length of this derivation.

Constraints can also be used to control exponents in regular expressions or be used to impose precise behaviors, e.g., for all variables to provide different values. We may use this agility by introducing anti-patterns [16].

Let us terminate this list of increasingly elaborated examples with a constraint based description of innermost rewriting.

*Example 10 (Innermost).* In this example, we assume the reader familiar with first-order term rewriting. The arrow symbol here means rewriting with a set of rewrite rules  $\mathcal{R}$  build over a set of term  $T(F, X)$ . Therefore, given a term, a derivation step is characterized by an occurrence and a rewrite rule. How can we describe, using appropriate constraints, that only innermost redexes are reduced? One of the difficulty is that the property to be innermost is not intrinsic to a derivation but to the application of a derivation to a term.

Let  $x$  be a variable representing the term to be reduced. The set of innermost derivations can be defined in the following way:

$$\begin{aligned}
IM(x) = & (D \mid (\exists y \in T(F, X), D =^? x \rightarrow D' && \text{the derivation is non} \\
& \wedge && \text{empty} \\
& D' \in^? IM(y) && \text{the remainder should} \\
& \wedge && \text{be innermost too} \\
& (\exists g \rightarrow d \in \mathcal{R}, \exists \omega, \exists \sigma, x|_{\omega} =^? \sigma(g)) && \text{it matches} \\
& \wedge && \\
& \neg(\exists w' < w, \exists l \rightarrow r \in \mathcal{R}, \exists \alpha, x|_{\omega'} =^? \alpha(l)) && \text{with no match above} \\
& \wedge && \\
& y =^? x[\sigma(d)]_{\omega}) && \text{gives the value to } y.
\end{aligned}$$

Remark that when a term is normalized, there is no IM derivation. As for the variety of the language, notice the set constraint on the second part of the conjunction. Finally, we could also use matching constraints to get rid of the explicit use of  $\sigma$  and  $\alpha$ .

Relating this definition to the intensional strategy  $\lambda_{inn}$  defined in Example 3, we could also write:

$$\begin{aligned}
IM(x) = & (D \mid (\exists y, D =^? x \xrightarrow{L} D' && \text{the derivation is non empty} \\
& \wedge && \\
& D' \in^? IM(y) && \text{the remainder should be innermost too} \\
& \wedge && \\
& (x \xrightarrow{L} y) \in^? \lambda_{inn}(x)) && \text{with } \lambda_{inn} \text{ defined as in Example 3.}
\end{aligned}$$

## 4 Conclusion

In this paper we came back on the definition of abstract strategies. After recalling the extensional and intensional ways to define them, we introduced derivation schemas and constraint based description of strategies. We applied this to both simple and more complex examples to illustrate the agility of the constraint based approach.

This procedure is quite powerful and allows us to describe, in a declarative fashion, strategies than can arguably be seen as elaborate. Our goal here is to free the user from the burden of engaging in the procedural definition of search spaces or reduction trees, and the complexity associated with the specification of history- and future-dependent derivations.

We are planning to apply this approach to theorem proving strategies, such as focusing, that currently rely mainly on syntactic apparatus [1,19] to guide proof search exploration. Application to constraint based specification of reduction strategies also appears quite challenging and remains to be explored.

Let us mention that the ability to explicitly use negations or complement problems could be of great help in defining strategies. For instance, the possibility of excluding conflicting assignments from proof exploration is a feature that is widely used in modern SAT solvers. In this context, we anticipate anti-patterns [16] to provide a useful constrained strategy tool.

Let us finally remark that strategies are currently defined using ML-like functional languages in theorem provers like Coq, HOL or Isabelle. This work is a first step to allow for a constraint based family of strategy languages in a constraint programming style.

## References

1. Jean-Marc Andreoli. Logic Programming with Focusing Proofs in Linear Logic. *Journal of Logic and Computation*, 2(3):297–347, 1992.
2. Franz Baader and Tobias Nipkow. *Term Rewriting and all That*. Cambridge University Press, 1998.
3. Emilie Balland, Paul Brauner, Radu Kopetz, Pierre-Etienne Moreau, and Antoine Reilles. Tom: Piggybacking Rewriting on Java. In *Proceedings of the 18th Conference on Rewriting Techniques and Applications*, volume 4533 of *Lecture Notes in Computer Science*, pages 36–47. Springer Verlag, 2007.
4. Peter Borovanský, Claude Kirchner, Helene Kirchner, and Christophe Ringeissen. Rewriting with strategies in ELAN: a functional semantics. *International Journal of Foundations of Computer Science*, 12(1):69–98, February 2001.
5. Tony Bourdier, Horatiu Cirstea, Daniel J. Dougherty, and Kirchner Hélène. Extensional and Intensional Strategies. In *Proceedings of the 9th International Workshop on Reduction Strategies in Rewriting and Programming, 28 June 2009, Brasilia (WRS'09)*, Electronic Proceedings In Theoretical Computer Science (to appear), 2009.
6. Claudio Sacerdoti Coen, Enrico Tassi, and Stefano Zacchiroli. Tincals: Step by Step Tacticals. *Electronic Notes in Theoretical Computer Science*, 174(2):125–142, 2007.
7. David Delahaye. A Tactic Language for the System Coq. In Michel Parigot and Andrei Voronkov, editors, *Proc. 7th Int. Conf. on Logic for Programming and Automated Reasoning*, volume 1955 of *Lecture Notes in computer Science*, pages 85–95. Springer-Verlag, November 2000.
8. Gilles Dowek, Thérèse Hardin, and Claude Kirchner. Theorem Proving Modulo. *Journal of Automated Reasoning*, 31(1):33–72, Nov 2003.
9. J. Jaffar and Jean-Louis Lassez. Constraint Logic Programming. In *Proceedings of the 14th Annual ACM Symposium on Principles Of Programming Languages, Munich (Germany)*, pages 111–119, 1987.
10. Gueorgui Jojgov. Holes with Binding Power. In *Proc. 2002 Int. Workshop on Proofs and Programs*. Springer-Verlag, 2003.
11. Jean-Pierre Jouannaud, Claude Kirchner, and Hélène Kirchner. Incremental Construction of Unification Algorithms in Equational Theories. In Josep Díaz, editor,

- Proceedings International Colloquium on Automata, Languages and Programming, Barcelona (Spain)*, volume 154 of *Lecture Notes in Computer Science*, pages 361–373. Springer-Verlag, 1983.
12. Claude Kirchner, Florent Kirchner, and H el ene Kirchner. Strategic Computations and Deductions. In Christoph Benzm uller, Chad E. Brown, J org Siekmann, and Richard Statman, editors, *Reasoning in Simple Type Theory. Festschrift in Honour of Peter B. Andrews on His 70th Birthday*, volume 17 of *Studies in Logic and the Foundations of Mathematics*, pages 339–364. College Publications, 2008.
  13. Claude Kirchner and H el ene Kirchner. Rewriting, Solving, Proving. A preliminary version of a book available at <http://www.loria.fr/~ckirchne/=rsp/rsp.pdf>, 1999.
  14. Claude Kirchner, H el ene Kirchner, and M. Rusinowitch. Deduction with symbolic constraints. *Revue d'Intelligence Artificielle*, 4(3):9–52, 1990. Special issue on Automatic Deduction.
  15. Claude Kirchner, Radu Kopetz, and Pierre-Etienne Moreau. Anti-Pattern Matching. In *16th European Symposium on Programming (ESOP'07)*, Braga, Portugal, 2007.
  16. Claude Kirchner, Radu Kopetz, and Pierre-Etienne Moreau. Anti-Pattern Matching Modulo. In Carlos Mart ın-Vide, Friedrich Otto, and Henning Fernau, editors, *Language and Automata Theory and Applications, Second International Conference, LATA 2008, Tarragona, Spain, March 13-19, 2008. Revised Papers*, volume 5196 of *Lecture Notes in Computer Science*, pages 275–286, Tarragona, Spain, 2008. Springer Verlag.
  17. Florent Kirchner and C esar Mu noz. The Proof Monad. Submitted to the Journal of Logic and Algebraic Programming, 2008.
  18. Jean-Jacques L evy. *R eductions correctes et optimales dans le lambda-calcul*. PhD thesis, Universit e de Paris VII, 1978.
  19. Chuck Liang and Dale Miller. A Unified Sequent Calculus for Focused Proofs. In *LICS: 24th Symp. on Logic in Computer Science*, pages 355–364, 2009.
  20. Narciso Mart ı-Oliet, Jos e Meseguer, and Alberto Verdejo. Towards a Strategy Language for Maude. In Narciso Mart ı-Oliet, editor, *Proceedings Fifth International Workshop on Rewriting Logic and its Applications, WRLA 2004, Barcelona, Spain, March 27 – April 4, 2004*, volume 117 of *Electronic Notes in Theoretical Computer Science*, pages 417–441. Elsevier, 2005.
  21. Terese. *Term Rewriting Systems*. Cambridge University Press, 2003. M. Bezem, J. W. Klop and R. de Vrijer, eds.
  22. Eelco Visser. Stratego: A Language for Program Transformation based on Rewriting Strategies. System Description of Stratego 0.5. In A. Middeldorp, editor, *Rewriting Techniques and Applications (RTA'01)*, volume 2051 of *Lecture Notes in Computer Science*, pages 357–361. Springer Verlag, May 2001.