TLP **12** (4–5): 737–753, 2012. © Cambridge University Press 2012
doi:10.1017/S1471068412000270

*Diagrammatic confluence for Constraint Handling Rules**

RÉMY HAEMMERLÉ
Technical University of Madrid

Abstract

Confluence is a fundamental property of Constraint Handling Rules (CHR) since, as in other rewriting formalisms, it guarantees that the computations are not dependent on rule application order, and also because it implies the logical consistency of the program declarative view. In this paper we are concerned with proving the confluence of non-terminating CHR programs. For this purpose, we derive from van Oostrom's decreasing diagrams method a novel criterion on CHR critical pairs that generalizes all preexisting criteria.We subsequently improve on a result on the modularity of CHR confluence, which permits modular combinations of possibly non-terminating confluent programs, without loss of confluence.

KEYWORDS: CHR, confluence, decreasing diagrams, modularity of confluence

1 Introduction

Constraint Handling Rules (CHR) is a committed-choice constraint logic programming language, introduced by Frühwirth (1998) for the easy development of constraint solvers. It has matured into a general-purpose concurrent programming language. Operationally, a CHR program consists of a set of guarded rules that rewrite multisets of constrained atoms. Declaratively, a CHR program can be viewed as a set of logical implications executed on a deduction principle.

Confluence is a basic property of rewriting systems. It refers to the fact that any two finite computations starting from a common state can be prolonged so as to eventually meet in a common state again. Confluence is an important property for any rule-based language, because it is desirable for computations to not be dependent on a particular rule application order. In the particular case of CHR, this property is even more desirable, as it guarantees the correctness of a program (Abdennadher *et al.* 1999; Haemmerlé *et al.* 2011): any program confluent has a consistent logical

* The research leading to these results has received funding from the Programme for Attracting Talent (PICD) / young PHD of the MONTEGANCEDO Campus of International Excellence, the Madrid Regional Government under the CM project P2009/TIC/1465 (PROMETIDOS), and the Spanish Ministry of Science under the MEC project TIN-2008-05624 *DOVES*.

reading. Confluence of a CHR program is also a fundamental prerequisite for logical completeness results (Abdennadher *et al.* 1999; Haemmerlé 2011a), makes possible program parallelization (Frühwirth 2005; Meister 2006), and may simplify program equivalence analyses (Abdennadher and Frühwirth 1999; Haemmerlé 2011b).

Following the pioneering research of Abdennadher *et al.* (1996), most existing work dealing with the confluence of CHR limits itself to terminating programs (see for instance the works by Abdennadher (1997) and Duck *et al.* (2007)). Nonetheless, proving confluence without global termination assumptions is still a worthwhile objective.

From a theoretical point of view, this is an interesting topic, because, as illustrated by the following example typical CHR programs fail to terminate on the level of abstract semantics, even if they do terminate on more concrete levels. Indeed, number of analytical results for the language rest on the notion of confluence, but only when programs are considered with respect to abstract semantics. For instance, in the current state of knowledge, even a result as important as the guarantee of correction by confluence only holds when programs are considered with respect to the most general operation semantics for CHR, namely the *very abstract semantics*.

Example 1 (*Partial order constraint*)
Let $\mathscr{P}_1$ be the classic CHR introductory example, namely the constraint solver for partial order. This consists of the following four rules, which define the meaning of the *user-defined* symbol $\leqslant$ using the built-in equality constraint $=$:

$$\begin{array}{lll} \textit{duplicate} & @ & x \leqslant y \setminus x \leqslant y \iff \top \\ \textit{reflexivity} & @ & x \leqslant x \iff \top \\ \textit{antisymmetry} & @ & x \leqslant y, y \leqslant x \iff x = y \\ \textit{transitivity} & @ & x \leqslant y, y \leqslant z \implies x \leqslant z \end{array}$$

The *duplicate* rule implements so-called *duplicate removal*. In other words, it states that if two copies of the same user-defined atom are present, then one of them can be removed. The *reflexivity* and *transitivity* rules respectively state that any atom of the form $x \leqslant x$ can be removed, and that two atoms $x \leqslant y$ and $x \leqslant y$ can be substituted with the built-in constraint $x = y$. Finally, the *transitivity* rule is a *propagation rule*. It states that if $x \leqslant y$ and $y \leqslant z$ are present, then the atom $x \leqslant z$ may be added.

It is well know that this program, like any other program using propagation rules, faces the so-called *trivial non-termination problem* when considered with respect to the very abstract semantics. Indeed, for these semantics, a propagation rule applies to any state it produces, leading to trivial loops. In order to solve this problem, Abdennadher (1997) proposed a token-based semantics in which propagation rules may be applied only once to the same combination of atoms. Nonetheless, such a proposal does not solve all the problems of termination. Indeed the *transitivity* rule may loop on queries containing a cycle in a chain of inequalities when considered against Abdennadher's semantics. Consider, for instance, the query $x \leqslant y, y \leqslant x$.

In fact, in order for $\mathscr{P}_1$ to be terminating, the rules of *reflexivity*, *antisymmetry*, and *transitivity* must have priority over the *transitivity* rule. This behaviour can be achieved by considering concrete semantics, such as the refined semantics of Duck

https://doi.org/10.1017/S1471068412000270 Published online by Cambridge University Press

et al. (2005). These semantics reduce the non-determinism of the CHR execution model by applying the rules in textual order.

In exchange for gaining termination, the most concrete semantics lose a number of analytical results. For instance, as explained by Frühwirth (2009), although any CHR program can be run in parallel in abstract semantics, one can obtain incorrect results for programs written with the refined semantics in mind. Indeed, if the result of a program relies on a particular rule application order, parallel execution will garble this order, leading to unexpected results. Interestingly, confluence on an abstract (but possibly non-terminating) level may come to the rescue of the most concrete semantics: If a program is confluent on a semantic level where the rule application order is not specified, then the result will not be dependent on the particular application order. Similar considerations have been discussed for equivalences of CHR programs (Haemmerlé 2011b).

From a more practical point of view, proving confluence without the assumption of termination is important, because it may be desirable to prove the confluence of a program for which termination cannot be inferred. Indeed, there exist very simple programs, such as the Collatz function, for which termination is only a conjecture (Guy 2004). Furthermore, since CHR is now a general-purpose language, analytical tools for the language must handle programs that do not terminate on any semantic level—for instance, interpreters for a Turing-complete language (Sneyers *et al.* 2009), or typical concurrent programs (see the numerous examples of concurrent systems given by Milner (1999)). We have also recently demonstrated that non-terminating execution models for CHR yield elegant frameworks for programming with coinductive reasoning (Haemmerlé 2011a). As a motivating example for the class of intrinsically non-terminating programs, we will use the following solution for the seminal dining philosophers problem.

Example 2 (*Dining philosophers*)
Consider the following CHR program $\mathscr{P}_2$ that implements a solution to the dining philosophers problem extended to count the number of times a philosopher eats:

$$\begin{array}{lcll} \mathit{eat} & @ & \mathbf{t}(x,y,i), \mathbf{f}(x), \mathbf{f}(y) & \Longleftrightarrow \quad \mathbf{e}(x,y,i+1) \\ \mathit{thk} & @ & \mathbf{e}(x,y,i) & \Longleftrightarrow \quad \mathbf{f}(x), \mathbf{f}(y), \mathbf{t}(x,y,i) \end{array}$$

The atom $\mathbf{f}(x)$ represents the fork x, the atom $\mathbf{e}(x,y,i)$ (resp. $\mathbf{t}(x,y,i)$) represents an eating (thinking) philosopher seated between forks x and y, who has already eaten i times. On the one hand, the rule *eat*, states that if a thinking philosopher is seated between two forks lying on the table, then he may start eating once he has picked up both forks. On the other hand, the rule *thk* states that a philosopher may stop eating if he puts down the forks he has been using. The initial state corresponding to n dining philosophers seated around a table can be encoded by the set of atoms $\mathbf{f}(1), \mathbf{t}(1,2,0), \mathbf{f}(2), \mathbf{t}(2,3,0), \cdots \mathbf{f}(n), \mathbf{t}(n,1,0)$.

Despite the fact that this program is intrinsically non-terminating, we may be interested in its confluence, for example, so that we may make use of one of the previously mentioned applications (e.g. confluence simplifies observational equivalence (Haemmerlé 2011b)). Confluence of $\mathscr{P}_2$ may also simplify the proofs of

https://doi.org/10.1017/S1471068412000270 Published online by Cambridge University Press

fundamental properties of concurrent systems, such as, for instance, the absence of deadlock: Starting from the initial state, one can easily construct a derivation where the i^{th} philosopher ($i \in 1,\ldots,n$) has eaten an arbitrary number of times. Hence if $\mathscr{P}_2$ is confluent, we can then infer that it is possible to extend any finite derivation such that the i^{th} philosopher eats strictly more, i.e. no derivation leads to a deadlock.

To the best of our knowledge, the only existing principle for proving confluence of non-terminating programs is the so-called *strong confluence* criterion (Haemmerlé and Fages 2007; Raiser and Tacchella 2007). However this criterion appears to be too weak to apply to common CHR programs, such as Examples 1 and 2. In this paper, we are concerned with extending CHR confluence theory to be able to capture a large class of possibly non-terminating programs. For this purpose we derive from the so-called *decreasing diagrams* technique a novel criterion that generalizes all existing confluence criteria for CHR. The decreasing diagrams technique is a method developed by van Oostrom (1994) which subsumes all sufficient conditions for confluence. Applying this method requires that all local rewrite peaks (i.e. points where the rewriting relation diverges because of non-determinism) can be completed into so-called decreasing diagrams.

The present paper presents two main contributions. In Section 4 , we present a particular instantiation of the decreasing diagrams technique to CHR, and show that in the context of this particular instantiation, the verification of decreasingness can be restricted to the standard notion of critical pairs. Then in Section 5, we extend the so-called *modularity of confluence* (Frühwirth 2009) so as to be able to combine programs which have independently been proven confluent, without losing confluence.

2 Preliminaries on abstract confluence

In this section, we gather some required notations, definitions, and results on the confluence of abstract rewriting systems. Terese's compendium (2003) can be referred to for a more detailed presentation.

A *rewrite relation* (or *rewrite* for short) is a binary relation on a set of objects E. For any rewrite $\rightarrow$, the symbol $\leftarrow$ will denote its converse, $\rightarrow^{=}$ its reflexive closure, $\rightarrow^{+}$ its transitive closure, and $\twoheadrightarrow$ its transitive-reflexive closure. We will use $\rightarrow_\alpha \cdot \rightarrow_\beta$ to denote the left-composition of all rewrites $\rightarrow_\alpha$ and $\rightarrow_\beta$. A *family* of rewrites is a set $(\rightarrow_\alpha)_{\alpha\in I}$ of rewrites indexed by a set I of labels. For such a family and any set K, $\rightarrow_K$ will denote the union $\bigcup_{\alpha\in(K\cap I)}(\rightarrow_\alpha)$.

A *reduction* is a finite sequence of rewriting steps of the form $(e_0 \xrightarrow{\mathscr{P}}_{\alpha_1} e_1 \xrightarrow{\mathscr{P}}_{\alpha_2} \cdots \xrightarrow{\mathscr{P}}_{\alpha_n} e_n)$. Such a reduction would be abreviated as $e_0 \xrightarrow{\mathscr{P}}_{\bar\alpha} e_n$ with $\bar\alpha = \alpha_1, \alpha_2, \ldots, \alpha_n$ when the intermediary states $e_1,\ldots,e_{n-1}$ are not relevant. A *peak* is a pair of reductions $e_l \xleftarrow{\mathscr{P}}_{\bar\alpha} e \xrightarrow{\mathscr{P}}_{\bar\beta} e_r$ from a common element e. A *local peak* is a peak formed by two one-step reductions. A *valley* is a pair of reductions $e_l \xrightarrow{\mathscr{P}}_{\bar\alpha} e' \xleftarrow{\mathscr{P}}_{\bar\beta} e_r$ ending in a common element e'. A peak $e_l \xleftarrow{\mathscr{P}}_{\bar\alpha} e \xrightarrow{\mathscr{P}}_{\bar\alpha'} e_r$ is *joinable* by $\xrightarrow{\mathscr{P}}_\beta \cdot \xleftarrow{\mathscr{P}}_{\beta'}$ if it is true that $e_l \xrightarrow{\mathscr{P}}_\beta \cdot \xleftarrow{\mathscr{P}}_{\beta'} e_r$.

https://doi.org/10.1017/S1471068412000270 Published online by Cambridge University Press

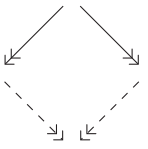

Fig. 1. Confluence.

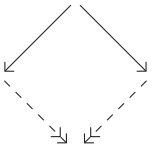

Fig. 2. Local Confluence.

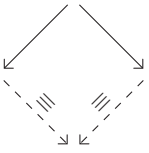

Fig. 3. Strong confluence.

A rewrite $\rightarrow$ is *terminating* if there is no infinite sequence of the form $e_0 \rightarrow e_1 \rightarrow e_2 \ldots$ Furthermore, we will say that $\rightarrow$ is *confluent* if $(\twoheadleftarrow \cdot \twoheadrightarrow) \subseteq (\twoheadrightarrow \cdot \twoheadleftarrow)$ holds, *locally confluent* if $(\leftarrow \cdot \rightarrow) \subseteq (\twoheadrightarrow \cdot \twoheadleftarrow)$ holds, and *strongly confluent*[1] if $(\leftarrow \cdot \rightarrow) \subseteq (\rightarrow^{=} \cdot \leftarrow^{=})$ holds. Figures 1–3 graphically represent these definitions. Following standard diagrammatic notation, solid edges stand for universally quantified rewrites, while dashed edges represent existentially quantified rewrites.

By the seminal lemma of Newman (1942), we know that a terminating and locally confluent rewrite is confluent. Another famous result due to Huet (1980) ensures that strong confluence implies confluence.

We now present a slight variation due to Hirokawa and Middeldorp (2010) of the so-called decreasing diagrams technique, which is more suitable for our purposes. The interest of the decreasing diagrams method (van Oostrom 1994) is that it reduces problems of general confluence to problems of local confluence. In exchange, the method requires the *confluence diagrams* (i.e. the way peaks close) to be decreasing with respect to a labeling provided with a wellfounded preorder. The method is complete in the sense that any countable confluence rewrite can be equipped with such a labeling. But because confluence is an undecidable property, finding such labeling may be difficult.

In the rest of this paper, we will say that a preorder $\succcurlyeq$ is *wellfounded*, if the strict preorder $\succ$ associated with $\succcurlyeq$ (i.e. $\alpha \succ \beta$ iff $\alpha \succcurlyeq \beta$ but not $\beta \succcurlyeq \alpha$) is a terminating relation. Let $(\rightarrow_\alpha)_{\alpha \in I}$ be a family of rewrites and $\succcurlyeq$ be a wellfounded preorder on I. A local peak $e_l \leftarrow_\alpha e \rightarrow_\beta e_r$ $(\alpha, \beta \in I)$ is *decreasing* with respect to $\succcurlyeq$ if the following holds:

$$e_l \twoheadrightarrow_{\curlyvee\{\alpha\}} \cdot \rightarrow^{=}_{\curlyvee\{\beta\}} \cdot \twoheadrightarrow_{\curlyvee\{\alpha,\beta\}} e' \twoheadleftarrow_{\curlyvee\{\alpha,\beta\}} \cdot \leftarrow^{=}_{\curlyvee\{\alpha\}} \cdot \twoheadleftarrow_{\curlyvee\{\beta\}} e_r \qquad (\star)$$

where for any set K of labels, $\curlyvee K$ stands for $\{\gamma \in I \mid \exists \delta \in K.\delta \succcurlyeq \gamma\}$ and $\curlywedge K$ for $\{\gamma \in I \mid \exists \delta \in K.\delta \succ \gamma\}$. A family $(\rightarrow_\alpha)_{\alpha \in I}$ of rewrites is *(locally) decreasing* if all local peaks of the form $u \leftarrow_\alpha \cdot \rightarrow_\beta v$ $(\alpha, \beta \in I)$ are decreasing with respect to a common wellfounded preorder on I. A rewrite is *(locally) decreasing* if it is the union of some decreasing families of rewrites. Property $(\star)$ is graphically represented in Figure 4.

Theorem 3 (*Decreasing Diagram (van Oostrom 1994)*)
A countable rewrite is confluent if and only if it is locally decreasing.

[1] For the sake of simplicity, we use a definition weaker than the one of Huet (1980). It is worth noting, that the counterexamples given in introduction stay relevant for the general definition.

https://doi.org/10.1017/S1471068412000270 Published online by Cambridge University Press

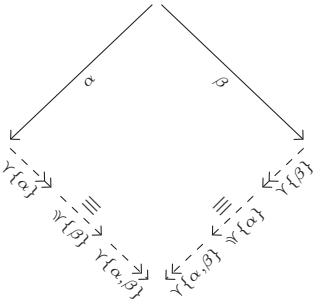

Fig. 4. Local decreasingness.

We recall now some other state-of-the-art results which will be used later.

Lemma 4 (*Terese 2003*)

(i) For all rewrites $\to_1, \to_2$ if $(\leftarrow_1 \cdot \twoheadrightarrow_2) \subseteq (\twoheadrightarrow_2 \cdot \twoheadleftarrow_1)$, then $(\twoheadleftarrow_1 \cdot \twoheadrightarrow_2) \subseteq (\twoheadrightarrow_2 \cdot \twoheadleftarrow_1)$.

(ii) For all rewrites $\to_1,\to_2$ s.t. $\to_1 \subseteq \to_2 \subseteq \twoheadrightarrow_1$, $\to_2$ is confluent iff $\to_1$ is confluent.

3 Preliminaries on constraint handling rules

In this section, we recall the syntax and the semantics of CHR. Frühwirth's book (2009) can be referred to for a more general overview of the language.

3.1 Syntax

The formalization of CHR assumes a language of *(built-in) constraints* containing equality over some theory $\mathscr{C}$, and defines *(user-defined) atoms* using a different set of predicate symbols. In the following, $\mathscr{R}$ will denote an arbitrary set of identifiers. By a slight abuse of notation, we allow confusion of conjunctions and multiset unions, omit braces around multisets, and use the comma for multiset union. We use $\mathrm{fv}(\phi)$ to denote the set of free variables of a formula ϕ. The notation $\exists_{\text{-}\psi}\phi$ denotes the existential closure of ϕ with the exception of free variables of ψ.

A *(CHR) program* is a finite set of eponymous rules of the form:

$$(r \ @\ \mathbb{K} \backslash \mathbb{H} \iff \mathbb{G} \mid \mathbb{B}; \mathbb{C})$$

where $\mathbb{K}$ (the *kept head*), $\mathbb{H}$ (the *removed head*), and $\mathbb{B}$ (the *user body*) are multisets of atoms, $\mathbb{G}$ (the *guard*) and $\mathbb{C}$ (the *built-in body*) are conjunctions of constraints and, $r \in \mathscr{R}$ (the *rule name*) is an identifier assumed unique in the program. Rules in which both heads are empty are prohibited. An empty guard $\top$ (resp. an empty kept head) can be omitted with the symbol $\mid$ (resp. with the symbol $\backslash$). Rules are divided into two classes: *simplification rules*[2] if the removed head is non-empty and

[2] Unlike standard presentations, our definition does not distinguish between simplification rules form the so-called simpagation rules.

https://doi.org/10.1017/S1471068412000270 Published online by Cambridge University Press

propagation rules otherwise. Propagation rules can be written using the alternative syntax:

$$(r \ @ \ \mathbb{K} \Longrightarrow \mathbb{G} \mid \mathbb{B}; \mathbb{C})$$

3.2 Operational semantics

In this section, we recall the equivalence-based operational semantics ω_e of Raiser *et al.* (2009). It is equivalent to the *very abstract* semantics ω_{va} of Frühwirth (1998), which is the most general operational semantics of CHR. We prefer the former because it includes an rigorous notion of equivalence, which is an essential component of confluence analysis.

A *(CHR) state* is a tuple $\langle \mathbb{C}; \mathbb{E}; \bar{x} \rangle$, where $\mathbb{C}$ (the *user store*) is a multiset of atoms, $\mathbb{E}$ (the *built-in store*) is a conjunction of constraints, and $\bar{x}$ (the *global variables*) is a finite set of variables. Unsurprisingly, the *local variables* of a state are those variables of the state which are not global. When no confusion can occur, we will syntactically merge user and built-in stores. We may futhermore omit the global variables component when states have no local variables. In the following, we use Σ to denote the set of states. Following Raiser *et al.*, we will always implicitly consider states modulo a structural equivalence. Formally, this *state equivalence* is the least equivalence relation $\equiv$ over states satisfying the following rules:

- $\langle \mathbb{E}; \mathbb{C}; \bar{x} \rangle \equiv \langle \mathbb{E}; \mathbb{D}; \bar{x} \rangle$ if $\mathscr{C} \vDash \exists_{\text{-}(\mathbb{E},\bar{x})} \mathbb{C} \leftrightarrow \exists_{\text{-}(\mathbb{E},\bar{x})} \mathbb{D}$
- $\langle \mathbb{E}; \bot; \bar{x} \rangle \equiv \langle \mathbb{F}; \bot; \bar{y} \rangle$
- $\langle \mathbb{A}, c; \mathbb{C}, c{=}d; \bar{x} \rangle \equiv \langle \mathbb{A}, d; \mathbb{C}, c{=}d; \bar{x} \rangle$
- $\langle \mathbb{A}; \mathbb{C}; \bar{x} \rangle \equiv \langle \mathbb{A}; \mathbb{C}; \{y\} \cup \bar{x} \rangle$ if $y \notin \mathrm{fv}(\mathbb{A}, \mathbb{C})$.

Once states are considered modulo equivalence, the operation semantics of CHR can be expressed by a single rule. Formally the operational semantcs of a program $\mathscr{P}$ is given by the least relation $\xrightarrow{\mathscr{P}}$ on states satisfying the rule:

$$\frac{\big(r \ @ \ \mathbb{K} \backslash \mathbb{H} \Longleftrightarrow \mathbb{G} | \mathbb{B}; \mathbb{C}\big) \in \mathscr{P}\rho \quad \mathrm{lv}(r) \cap \mathrm{fv}(\mathbb{E}, \mathbb{D}, \bar{x}) = \emptyset}{\langle \mathbb{K}, \mathbb{H}, \mathbb{E}; \mathbb{G}, \mathbb{D}; \bar{x} \rangle \xrightarrow{\mathscr{P}} \langle \mathbb{K}, \mathbb{B}, \mathbb{E}; \mathbb{G}, \mathbb{C}, \mathbb{D}; \bar{x} \rangle}$$

where ρ is a renaming. A program $\mathscr{P}$ is *confluent* (resp. *terminating*) if $\xrightarrow{\mathscr{P}}$ is confluent (resp. terminating).

Before going further, we recall an important property of CHR semantics. This property, monotonicity, means that if a transition is possible in a state, then the same transition is possible in any larger state. To help reduce the level of verbosity we introduce the notion of the *quantified conjunction* of states (Haemmerlé and Fages 2007). This operator allows the composition of states with disjoint local variables while quantifying some of their global variables (i.e. changing global variables into local ones). Formally, the quantified conjunction is a binary operator on states parametrized by a set of variables $\bar{z}$ satisfying:

$$\langle \mathbb{E}; \mathbb{C}; \bar{x} \rangle \oplus_{\bar{z}} \langle \mathbb{F}; \mathbb{D}; \bar{y} \rangle = \langle \mathbb{E}, \mathbb{F}; \mathbb{C}, \mathbb{D}; (\bar{x}\bar{y}) \setminus \bar{z} \rangle \text{ if } (\mathrm{fv}(\mathbb{E}, \mathbb{C}) \cap \mathrm{fv}(\mathbb{F}, \mathbb{D})) \subseteq (\bar{x} \cap \bar{y})$$

Note the side condition is not restrictive, as local variables can always be renamed using the implicit state equivalence.

https://doi.org/10.1017/S1471068412000270 Published online by Cambridge University Press

Proposition 5 (*Monotonicity of CHR*)
Let $\mathscr{P}$ be a CHR program, S, S_1, S_2 be CHR states, and $\bar{x}$ be a set of variables.

$$\text{If } S_1 \xrightarrow{\mathscr{P}} S_2, \text{ then } S_1 \oplus_{\bar{x}} S \xrightarrow{\mathscr{P}} S_2 \oplus_{\bar{x}} S$$

3.3 Declarative semantics

Owing to its origins in the tradition of CLP, the CHR language features declarative semantics through direct interprestation in first-order logic. Formally, the *logical reading* of a rule of the form:

$$\mathbb{K} \backslash \mathbb{H} \iff \mathbb{G} \mid \mathbb{B}; \mathbb{C}$$

is the guarded equivalence:

$$\forall\big((\mathbb{K} \wedge \mathbb{G}) \rightarrow \big(\mathbb{H} \leftrightarrow \exists_{\text{-}(\mathbb{K},\mathbb{H})}(\mathbb{G} \wedge \mathbb{C} \wedge \mathbb{B})\big)\big)$$

The *logical reading* of a program $\mathscr{P}$ within a theory $\mathscr{C}$ is the conjunction of the logical readings of its rules with the constraint theory $\mathscr{C}$. It is denoted by $\mathscr{C}\mathscr{P}$.

Operational semantics is sound and complete with respect to this declarative semantics (Frühwirth 1998; Abdennadher *et al.* 1999). Furthermore, any program confluent with respect to ω_e has a consistent logical reading (Abdennadher *et al.* 1999; Haemmerlé *et al.* 2011).

4 Diagrammatic confluence for Constraint Handling Rules

In this section, we are concerned with proving confluence of a large class of CHR programs. Indeed, as explained in the introduction, existing criteria are not sufficiently powerful to infer confluence of common non-terminating programs. (See Examples 13 and 14 for concrete examples). To avoid this limitation, we will derive from the decreasing diagrams technique a novel csriterion on CHR critical pairs that generalizes both local and strong confluence criteria. An analogue criterion has been developed for linear Term Rewriting Systems (TRS) (Jouannaud and van Oostrom 2009).

4.1 Labels for Constraint Handling Rules

In order to apply the decreasing diagram technique to CHR, we will need first to label CHR transitions. In this work, we will use two labelings proposed by van Oostrom (2008) for TRS. The first one is the so-called *rule-labeling*. It consists of labeling each transition $a \xrightarrow{\mathscr{P}} b$ with the name of the applied rule. This labeling is ideal for capturing strong confluence-like properties for linear TRS. Within the proof of our main result, we will also use the so-called *self-labeling* which consists of labeling each transition $a \xrightarrow{\mathscr{P}} b$ with its source a. This second labeling captures the confluence of terminating rewrites.

In practice, we will assume that the set $\mathscr{R}$ of rule identifiers is defined as a disjoint union $\mathscr{R}_i \uplus \mathscr{R}_c$. For a given program $\mathscr{P}$, we denote by $\mathscr{P}^i$ (resp. $\mathscr{P}^c$) the set of rules

https://doi.org/10.1017/S1471068412000270 Published online by Cambridge University Press

form $\mathscr{P}$ built with $\mathscr{R}_i$ (resp. $\mathscr{R}_c$). We call $\mathscr{P}^i$ the *inductive* part of $\mathscr{P}$, because we will subsequently assume that $\mathscr{P}^i$ is terminating, while $\mathscr{P}^c$ will be called *coinductive*, as it will be typically non-terminating.

Definition 6 (*Rule-labeling*)
The *rule-labeling* of a CHR program $\mathscr{P}$ is the family of rewrites $(\xrightarrow{\mathscr{P}}_r)_{r\in\mathscr{R}}$ indexed by rule identifiers, where $\xrightarrow{\mathscr{P}}_r = \xrightarrow{\{r\}}$. A preorder $\succcurlyeq$ on rule identifiers is *admissible*, if any inductive rule identifier is strictly smaller than any coinductive one (i.e. for any $r_i \in \mathscr{R}_i$ and any $r_c \in \mathscr{R}_c$, $r_c \succ r_i$ holds).

4.2 Critical peaks

In TRS, the basic techniques used to prove confluence consist of showing various confluence criteria on a finite set of special cases, called *critical pairs*. Critical pairs are generated by a superposition algorithm, in which one attempts to capture the most general way the left-hand sides of the two rules of the system may overlap. The notion of critical pairs has been successfully adapted to CHR by Abdennadher *et al.* (1996). Here, we introduce a slight extension of the notion that takes into account the rule-labeling we have just defined.

Definition 7 (*Critical peak*)
Let us assume that r_1 and r_2 are CHR rules renamed apart:

$$\big(r_1 \mathbin{@} \mathbb{K}_1 \backslash \mathbb{H}_1 \iff \mathbb{G}_1 \mid \mathbb{B}_1 ; \mathbb{C}_1\big) \in \mathscr{P}_1 \qquad \big(r_2 \mathbin{@} \mathbb{K}_2 \backslash \mathbb{H}_2 \iff \mathbb{G}_2 \mid \mathbb{B}_2 ; \mathbb{C}_2\big) \in \mathscr{P}_2$$

A *critical ancestor (state)* S_c for the rules r_1 and r_2 is a state of the form:

$$S_c = \langle \mathbb{H}_1^{\Delta}, \mathbb{H}_1^{\cap}, \mathbb{H}_2^{\Delta} ; \mathbb{D} ; \bar{x} \rangle$$

satisfying the following properties:

- $(\mathbb{K}_1, \mathbb{H}_1) \doteq (\mathbb{H}_1^{\Delta}, \mathbb{H}_1^{\cap})$, $(\mathbb{K}_2, \mathbb{H}_2) \doteq (\mathbb{H}_2^{\Delta}, \mathbb{H}_2^{\cap})$, $\mathbb{H}_1^{\cap} \neq \emptyset$, and $\mathbb{H}_2^{\cap} \neq \emptyset$;
- $\bar{x}_1 = \mathrm{fv}(\mathbb{K}_1, \mathbb{H}_1)$, $\bar{x}_2 = \mathrm{fv}(\mathbb{K}_2, \mathbb{H}_2)$ and $\bar{x} = \bar{x}_1 \cup \bar{x}_2$;
- $\mathbb{D} = (\mathbb{H}_1^{\cap} \doteq \mathbb{H}_2^{\cap}, \mathbb{G}_1, \mathbb{G}_2)$ and $\exists \mathbb{D}$ is $\mathscr{C}$-satisfiable;
- $\mathbb{H}_1^{\cap} \not\subseteq \mathbb{K}_1$ or $\mathbb{H}_2^{\cap} \not\subseteq \mathbb{K}_2$.

Then the following tuple is called a *critical peak* between r_1 and r_2 at S_c:

$$\langle \mathbb{K}_1, \mathbb{B}_1, \mathbb{H}_2^{\Delta} ; \mathbb{D}, \mathbb{C}_1 ; \bar{x} \rangle \xleftarrow{\mathscr{P}}_{r_1} S_c \xrightarrow{\mathscr{P}}_{r_2} \langle \mathbb{K}_2, \mathbb{B}_2, \mathbb{H}_1^{\Delta} ; \mathbb{D}, \mathbb{C}_2 ; \bar{x} \rangle$$

A critical peak between a program $\mathscr{P}$ and a program $\mathscr{Q}$ is a critical peak between a rule of $\mathscr{P}$ and a rule of $\mathscr{Q}$. A critical peak of a program $\mathscr{P}$ is a critical peak between $\mathscr{P}$ and itself. A critical peak is *inductive* if it involves only inductive rules (i.e. a critical peak of $\mathscr{P}^i$), or *coinductive* if it involves at least one coinductive rule (i.e. a critical peak between $\mathscr{P}^c$ and $\mathscr{P}$).

Example 8
Consider the solver partial order $\mathscr{P}_1$, given in Example 1. The following ciritial peak stems from overlapping the heads of the rules *antisymmetry* and *transitivity*:

$$\langle x = y \rangle \xleftarrow{\mathscr{P}_1}_{anti.} \langle x \leqslant y, y \leqslant x \rangle \xrightarrow{\mathscr{P}_1}_{trans.} \langle x \leqslant y, y \leqslant x, x \leqslant x \rangle$$

https://doi.org/10.1017/S1471068412000270 Published online by Cambridge University Press

4.3 Rule-decreasingness

We now come to our main result, showing that the study of decreasingness with respect to the rule-labeling can be restricted to critical peaks without loss of generality.

Definition 9 (*Critical rule-decreasingness*)
A program $\mathscr{P}$ is *(critically) rule-decreasing* w.r.t. an admissible preorder $\succcurlyeq$ if:

- the inductive part of $\mathscr{P}$ is terminating,
- all inductive critical peaks of $\mathscr{P}$ are joinable by $\xrightarrow{\mathscr{P}^i}{}^{*} \cdot {}^{*}\xleftarrow{\mathscr{P}^i}$, and
- all coinducitve critical peaks of $\mathscr{P}$ are decreasing w.r.t. $\succcurlyeq$.

A program is *rule-decreasing* if it is rule-decreasing with respect to some admissible preorder. A rule-decreasing program is *strongly rule-decreasing* if it is purely coinductive (i.e. without inductive rules).

Theorem 10
Rule-decreasing programs are confluent.

Proof
Let us assume that $\mathscr{P}$ is a rule-decreasing program w.r.t. a given preorder $\succcurlyeq_{\mathscr{R}}$. Now let $(\xrightarrow{\mathscr{P}}_\alpha)_{\alpha\in(\Sigma\cup\mathscr{R}_c)}$, the family of rewrites indexed by rule or state, be defined as

$$\xrightarrow{\mathscr{P}}_\alpha = \begin{cases} \xrightarrow{\mathscr{P}^i} \cap \big(\{\alpha\} \times \Sigma\big) & \text{if } \alpha \in \Sigma \qquad \text{(self-labeling on inductive part)} \\ \xrightarrow{\{\alpha\}} & \text{if } \alpha \in \mathscr{R}_c \qquad \text{(rule-labeling on coinductive part)} \end{cases}$$

Let $\succcurlyeq$ be the union of $\succcurlyeq_{\mathscr{R}}$, $\xrightarrow{\mathscr{P}^i}{}^{+}$, and $\{(r,\alpha) \mid r \in \mathscr{R} \;\&\; \alpha \in I\}$. By assuming without loss of generality that $\mathscr{R}$ is finite (i.e. $\succcurlyeq_{\mathscr{R}}$ is trivially wellfounded), we obtain that $\succcurlyeq$ is wellfounded. With the help of Theorem 3, it suffices to prove that each peak $S_\alpha \xleftarrow{\mathscr{P}}_\alpha S \xrightarrow{\mathscr{P}}_\beta S_\beta$ $(\alpha,\beta \in (\mathscr{R}_c \cup \Sigma))$ is decreasing w.r.t. $\succcurlyeq$. We distinguish two cases:

1 The rules r_α and r_β used to respectively produce S_α and S_β apply to different parts of S. By monotonicity of CHR transitions, we infer $S_\alpha \xrightarrow{\{r_\beta\}} S' \xleftarrow{\{r_\alpha\}} S_\beta$. We have to show this *valley* respects property $(\star)$ within the definition of the decreasing diagrams. We proceed by cases on the types of the rules r_α and r_β:

   1.1 r_α is inductive. We have $\alpha = S$, $\alpha \xrightarrow{\mathscr{P}^i}{}^{+} S_\alpha$, and $S_\beta \xrightarrow{\mathscr{P}}_{S_\beta} S'$.

      1.1.1 r_β is inductive. We have $\beta = S$, $\beta \xrightarrow{\mathscr{P}^i} S_\beta$, and $S_\alpha \xrightarrow{\mathscr{P}}_{S_\alpha} S'$. Since $\mathscr{P}^i$ is terminating, we infer $\alpha \succ S_\alpha$ and $\beta \succ S_\beta$. We conclude $S_\alpha \xrightarrow{\mathscr{P}}_{S_\alpha} S' \xleftarrow{\mathscr{P}}_{S_\beta} S_\beta$, i.e. the peak is decreasing w.r.t. $\succcurlyeq$.

      1.1.2 r_β is coinductive. We have $\beta \in \mathscr{R}_c$, $S_\alpha \xrightarrow{\mathscr{P}}_\beta S'$, and $\beta \succ S_\alpha$. We conclude $S_\alpha \xrightarrow{\mathscr{P}}_{S_\alpha} S' \xleftarrow{\mathscr{P}}_\beta S_\beta$, i.e. the peak is decreasing w.r.t. $\succcurlyeq$.

   1.2 r_α is coinductive. We have $\alpha \in \mathscr{R}_c$ and $S_\beta \xrightarrow{\mathscr{P}}_\alpha S'$.

      1.2.1 r_β is inductive. The case is symmetric with case 1.1.2.

      1.2.2 r_β is coinductive. We have $\beta \in \mathscr{R}_c$ and $S_\alpha \xrightarrow{\mathscr{P}}_\beta S'$. We conclude $S_\alpha \xrightarrow{\mathscr{P}}_\alpha S' \xleftarrow{\mathscr{P}}_\beta S_\beta$, i.e. the peak is decreasing w.r.t. $\succcurlyeq$.

https://doi.org/10.1017/S1471068412000270 Published online by Cambridge University Press

2 The applications of the rules r_α and r_β used to respectively produce S_α and S_β overlap. There should exist a critical peak $R_\alpha \overset{\mathscr{P}}{\leftarrow}_{r_\alpha} S_c \overset{\mathscr{P}}{\rightarrow}_{r_\beta} R_\beta$, a state R, and a set of variables $\bar{y}$, such that $S \equiv S_c \oplus_{\bar{x}} R$, $S_\alpha \equiv R_\alpha \oplus_{\bar{x}} R$, and $R_\beta \equiv R_\beta \oplus_{\bar{x}} R$. We proceed by cases on the types of rules r_α and r_β:

2.1 Both rules are inductive: We have $\beta = \alpha = S$, and by hypothesis we have

$$R_\alpha \equiv R_\alpha^0 \overset{\mathscr{P}^i}{\rightarrow} R_\alpha^1 \overset{\mathscr{P}^i}{\rightarrow} \cdots S_\alpha^m \equiv S' \equiv R_\beta^n \cdots \overset{\mathscr{P}^i}{\leftarrow} R_\beta^1 \overset{\mathscr{P}^i}{\leftarrow} R_\beta^0 \equiv R_\beta$$

By monotony of CHR we infer:

$$S_\alpha \equiv S_\alpha^0 \overset{\mathscr{P}^i}{\rightarrow} S_\alpha^1 \overset{\mathscr{P}^i}{\rightarrow} \cdots S_\alpha^m \equiv S \equiv S_\beta^n \cdots \overset{\mathscr{P}^i}{\leftarrow} S_\beta^1 \overset{\mathscr{P}^i}{\leftarrow} S_\beta^0 \equiv S_\beta$$

where $S_\alpha^i = R_\alpha^i \oplus_{\bar{x}} R$ (for $i \in 0,\ldots m$), $S_\beta^i = R_\beta^i \oplus_{\bar{x}} R$ (for $i \in 0,\ldots n$), and $S = S' \oplus R$. By construction of $(\overset{\mathscr{P}}{\rightarrow}_\alpha)_{\alpha \in \Sigma \times \mathscr{R}_c}$ we get:

$$S_\alpha \overset{\mathscr{P}}{\rightarrow}_{S_\alpha^0} S_\alpha^1 \overset{\mathscr{P}}{\rightarrow} S_\alpha^1 \cdots S_\alpha^m \equiv S \equiv S_\beta^n \cdots \overset{\mathscr{P}}{\leftarrow}_{S_\beta^1} S_\beta^1 \overset{\mathscr{P}}{\leftarrow}_{S_\beta^0} S_\beta$$

To conclude about the discussion of the decreasingness of the peak, it is just necessary to notice that for any $i \in 0,\ldots m$ and any $j \in 0,\ldots n$, both $S \overset{\mathscr{P}}{\rightarrow}^+ S_\alpha^i$ and $S \overset{\mathscr{P}}{\rightarrow}^+ S_\beta^j$ hold, i.e. $S_\alpha^i, S_\beta^j \in \curlyvee\{\alpha, \beta\}$.

2.2 One of the rules is coinductive. By hypothesis we have

$$R_\alpha \twoheadrightarrow_{\curlyvee\{r_1\}} \cdot \rightarrow^{\equiv}_{\curlyvee\{r_2\}} \cdot \twoheadrightarrow_{\curlyvee\{r_1,r_2\}} \cdot \twoheadleftarrow_{\curlyvee\{r_1,r_2\}} \cdot \leftarrow^{\equiv}_{\curlyvee\{r_1\}} \cdot \twoheadleftarrow_{\curlyvee\{r_2\}} R_\beta$$

or equivalently by monotony of CHR:

$$S_\alpha \twoheadrightarrow_{\curlyvee\{r_1\}} \cdot \rightarrow^{\equiv}_{\curlyvee\{r_2\}} \cdot \twoheadrightarrow_{\curlyvee\{r_1,r_2\}} \cdot \twoheadleftarrow_{\curlyvee\{r_1,r_2\}} \cdot \leftarrow^{\equiv}_{\curlyvee\{r_1\}} \cdot \twoheadleftarrow_{\curlyvee\{r_2\}} S_\beta \qquad \square$$

Theorem 10 strictly subsumes all the criteria for proving confluence of CHR programs we are aware of, namely the local confluence (Abdennadher *et al.* 1999) and the strong confluence (Haemmerlé and Fages 2007) criteria.

Corollary 11 (*Local confluence*)
A terminating program $\mathscr{P}$ is confluent if its critical peaks are joinable by $\overset{\mathscr{P}}{\twoheadrightarrow} \cdot \overset{\mathscr{P}}{\twoheadleftarrow}$.

Corollary 12 (*Strong confluence*)
A program $\mathscr{P}$ is confluent if its critical peaks are joinable by $\overset{\mathscr{P}}{\rightarrow}^{\equiv} \cdot \overset{\mathscr{P}}{\leftarrow}^{\equiv}$.

The following examples show that the rule-decreasingness criterion is more powerful than both local and strong confluence criteria.

Example 13
Consider the solver $\mathscr{P}_{1.}$ for partial order given in Example 1. Since $\mathscr{P}_{1.}$ is trivially non-terminating one cannot apply local confluence criterion. Strong confluence does not apply either, because of some non-strongly joinable critical peaks. For instance, considere the peak given at Example 8:

$$\langle x = y \rangle \overset{\mathscr{P}_{1.}}{\leftarrow}_{anti.} \langle x \leqslant y, y \leqslant x \rangle \overset{\mathscr{P}_{1.}}{\rightarrow}_{trans.} \langle x \leqslant y, y \leqslant x, x \leqslant x \rangle$$

It can be seen that $\langle x = y \rangle$ may not be reduced, and that the right-hand side cannot be rewritten into the left-hand side in less than two steps (e.g. by using *reflexivity* and *antisymmetry* rules).

https://doi.org/10.1017/S1471068412000270 Published online by Cambridge University Press

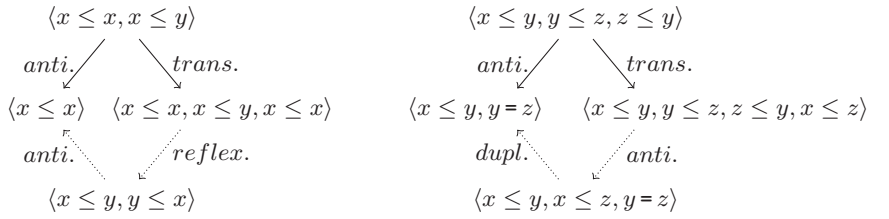

Fig. 5. Some rule-decreasing critical peaks for $\mathscr{P}_{1.}$

Nonetheless, confluence of $\mathscr{P}_{1.}$ can be deduced using the full generality of Theorem 10. For this purpose, assume that all rules except *transitivity* are inductive and take any admissible preorder. Clearly the inductive part of $\mathscr{P}_{1.}$ is terminating. Indeed the application of any one of the three first rules strictly reduces the number of atoms in a state. Then by a systematic analysis of all critical peaks of $\mathscr{P}_{1.}$, we prove that each peak can be closed while respecting the hypothesis of rule-decreasingness. In fact all critical peaks can be closed without using *transitivity*. Some rule-decreasing diagrams involving the *transitivity* rule are given as examples in Figure 5.

Example 14

Consider the program $\mathscr{P}_{2.}$ implementing the dining philosophers problem, as given in Example 2. The confluence of $\mathscr{P}_{2.}$ cannot be inferred by either local or strong confluence. On the one hand, $\mathscr{P}_{2.}$ is obviously non-terminating, and hence prevents the application of the local confluence criterion. On the other hand, $\mathscr{P}_{2.}$ has critical peaks which are not in $(\xrightarrow{\mathscr{P}_{2.}}\!\!\!\!\!\rightarrow \cdot \xleftarrow{\mathscr{P}_{2.}}\!\!\!\!\!\leftarrow)$. Consider as an example the peak given in Figure 6. It is critical for the rule *eating* with itself, but it is not joinable by $(\xrightarrow{\mathscr{P}_{2.}}\!\!\!\!\!\rightarrow^{\equiv} \cdot \xleftarrow{\mathscr{P}_{2.}}\!\!\!\!\!\leftarrow^{\equiv})$. However, the figure shows that it is joinable by

$$\xrightarrow{\mathscr{P}_{2.}}\!\!\!\!\!\rightarrow^{\equiv}_{thk} \cdot \xrightarrow{\mathscr{P}_{2.}}\!\!\!\!\!\rightarrow^{\equiv}_{eat} \cdot \xrightarrow{\mathscr{P}_{2.}}\!\!\!\!\!\rightarrow^{\equiv}_{thk} \cdot \xleftarrow{\mathscr{P}_{2.}}\!\!\!\!\!\leftarrow^{\equiv}_{thk} \cdot \xleftarrow{\mathscr{P}_{2.}}\!\!\!\!\!\leftarrow^{\equiv}_{eat} \cdot \xleftarrow{\mathscr{P}_{2.}}\!\!\!\!\!\leftarrow^{\equiv}_{thk}$$

i.e. the peak is decreasing. In fact, all the critical peaks of $\mathscr{P}_{2.}$ involve only the rule *eat* and may be closed in a similar manner. Thus, by assuming that the *eat* rule is coinductive and strictly greater than *thk*, we can infer, using Theorem 10, that $\mathscr{P}_{2.}$ is confluent.

4.4 On program partitioning

The rule-decreasingness criterion is based on the division of the program into a terminating part and a possibly non-terminating one. Since a program can be partitioned in multiple ways, it may be the case that the rule-decreasingness of a program depends on the splitting used (see Example 16). From a purely theoretical point of view, this is not a particular drawback, since the property we aim at proving (i.e. the confluence of program) is undecidable. From a more pragmatical point of view, it appears that the classic examples of CHR programs can be proved to be rule-decreasing without any assumption of termination. In particular, we were

https://doi.org/10.1017/S1471068412000270 Published online by Cambridge University Press

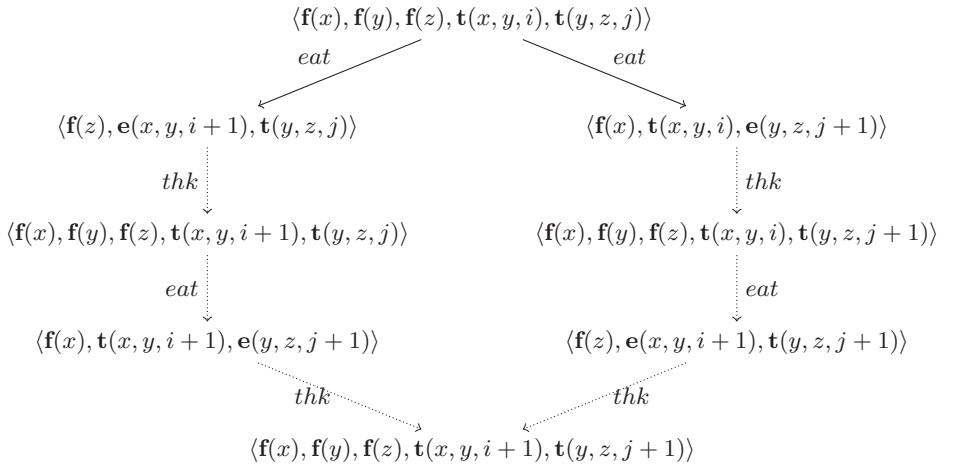

Fig. 6. A rule-decreasing critical peak of $\mathscr{P}_2$.

unable to find a counterexample of a confluent but non-strongly rule-decreasing program in Frühwirth's book (2009).

Example 15
Consider the CHR solver for partial order given in Example 1. Assuming that any rule is coinductive, $\mathscr{P}$ can be shown strongly rule-decreasing with respect the order $\succcurlyeq$ satisfying:

$$\textit{transitivity} \succ \textit{duplicate} \succ \textit{antisymmetry} \succ \textit{reflexivity}$$

As illustrated by Figure 5, critical peaks involving *transitivity* rules may be closed using only rules that are strictly smaller. Similarly, one can verify that any critical peak between a given rule α and a smaller (or equal) one can be closed using only rules strictly smaller than α (i.e. all the peaks are trivialy decreasing).

The choice of a good partition may simplify proofs of rule-decreasingness: by maximizing the inductive part of a program, the number of peaks which must be proved decreasing (i.e. the coinductive critical peaks) is reduced. Indeed, while the joinability of a peak with respect to the inductive part of program – which must be terminating – is a decidable problem and can be efficiently automatized,[3] the rule-decreasingness of a peak with respect to a possibly non-terminating program is likely to be undecidable.[4] Consequently, a good partition will limit the use of heuristics or human interactions necessary to infer a rule-decreasing diagram for each coinductive critical peak.

Since termination is also an undecidable property, we cannot expect to fully automatize the search for the optimal partition, and we must content ourselves

[3] See the works about CHR local confluence (Abdennadher *et al.* 1999; Abdennadher 1997).
[4] Decreasingness of a peak for a given order seems a more difficult problem than joinability without termination assumption—which is itself undecidable.

https://doi.org/10.1017/S1471068412000270 Published online by Cambridge University Press

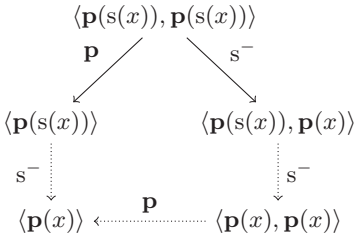

Fig. 7. Critical peak of $\mathscr{P}_{16}^{-}$.

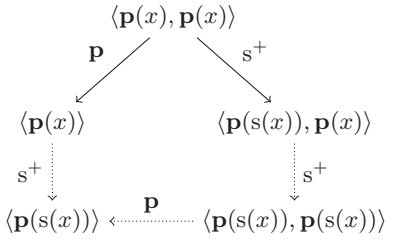

Fig. 8. Critical peak of $\mathscr{P}_{16}^{-}$.

with heuristic procedures. Despite the fact that the formal development of such procedures is beyond the scope of this paper, our practical experience suggests that a trivial partitioning may be interesting. This partition consists of considering as inductive only those rules that strictly reduce the number of atoms in a state. Even if this choice is not necessarily optimal and may even produce bad partitions, it does seem to produce relevant partitions for typical CHR solvers, as illustrated by Example 13.

We now give two counterexamples. The first shows that rule-decreasingness can be dependent on particular splittings, while the second presents a confluent program which is not rule-decreasing.

Example 16
Consider the following CHR rules:

$$\mathit{duplicate} \ @ \ \mathbf{p}(x) \backslash \mathbf{p}(x) \Longleftrightarrow \top \qquad \mathrm{s}^{-} \ @ \ \mathbf{p}(\mathrm{s}(x)) \Longleftrightarrow \mathbf{p}(x) \qquad \mathrm{s}^{+} \ @ \ \mathbf{p}(x) \Longleftrightarrow \mathbf{p}(\mathrm{s}(x))$$

We denote by $\mathscr{P}_{16}^{-}$ the program built from the *duplicate* and s^{-} rules, and by $\mathscr{P}_{16}^{+}$ the program built from the *duplicate* and s^{+} rules.

$\mathscr{P}_{16}^{-}$ is clearly terminating: the *duplicate* rule strictly reduces the number of atoms in a state, while s^{-} leaves the number of atoms unchanged, but strictly reduces the size of the argument of one of them. We can also verify that $\mathscr{P}_{16}^{-}$ has a single critical peak. Figure 7 shows the only way this peak may be closed. Thus, by assuming that all rules are inductive, we can infer that the program is rule-decreasing. However if s^{-} is assumed to be coinductive, we can verify that the sole critical peak of $\mathscr{P}_{16}^{-}$ is decreasing with respect to no admissible order.

As in the case of $\mathscr{P}_{16}^{-}$, $\mathscr{P}_{16}^{+}$ yields only one critical peak which is decreasing with respect to no admissible order (see Figure 8). However, this time s^{+} is not terminating, and so cannot been assumed inductive. Consequently $\mathscr{P}_{16}^{+}$ cannot be inferred to be confluent using Theorem 10.

5 Modularity of CHR confluence

In this section, we are concerned with proving the confluence of union of confluent programs in a modular way (in particular of those programs proved confluent using the rule-decreasing criterion). In practice, we improve on a result of Frühwirth (2009)

https://doi.org/10.1017/S1471068412000270 Published online by Cambridge University Press

which states that a terminating union of confluent programs which do not overlap (i.e. which do not have a critical peak) is confluent. In particular, we allow some overlapping and we drop the termination hypotheses.

Theorem 17 (*Modularity of confluence*)
Let $\mathscr{P}$ and $\mathscr{Q}$ be two confluent CHR programs. If any critical peak between $\mathscr{P}$ and $\mathscr{Q}$ is joinable by $\xrightarrow{\mathscr{Q}}{\twoheadrightarrow} \cdot \xleftarrow{\mathscr{P}}{}^{\equiv}$, then $\mathscr{P}\mathscr{Q}$ is confluent.

Before formally proving the theorem, it is worth noting that, despite the fact that modularity of confluence and the rule-decreasing theorem have similar flavors, both results have different scopes. Indeed, on the one hand modularity of confluence does not assume anything about the way in which $\mathscr{P}$ and $\mathscr{Q}$ are confluent. For instance, if $\mathscr{P}$ and $\mathscr{Q}$ are two rule-decreasing programs, Theorem 17 does not require the union of the inductive parts of $\mathscr{P}$ and $\mathscr{Q}$ to be terminating, while Theorem 10 does. This is important since, termination is not a modular property: even if two terminating programs do not share any user-defined atoms, one cannot be sure that their union is terminating. (See Section 5.4 of Frühwirth's book (2009) for more details.) On the other hand, the rule-decreasing criterion allows the critical peaks to be closed in a more complex way than Theorem 17 permits.

The proof of the theorem rests on the following lemma, which states that under the hypotheses of Theorem 17, $\xrightarrow{\mathscr{P}}$ "strongly commutes" with $\xrightarrow{\mathscr{Q}}$.

Lemma 18
If critical peaks between $\mathscr{P}$ and $\mathscr{Q}$ are in $\xrightarrow{\mathscr{Q}}{\twoheadrightarrow} \cdot \xleftarrow{\mathscr{P}}{}^{\equiv}$, then $(\xleftarrow{\mathscr{P}} \cdot \xrightarrow{\mathscr{Q}}{\twoheadrightarrow}) \subseteq (\xrightarrow{\mathscr{Q}}{\twoheadrightarrow} \cdot \xleftarrow{\mathscr{P}}{}^{\equiv})$.

Proof
We prove by induction on the length of the derivation $S_c \xrightarrow{\mathscr{Q}}{\twoheadrightarrow} S'$ that for any peak $S \xleftarrow{\mathscr{P}} S_c \xrightarrow{\mathscr{Q}}{\twoheadrightarrow} S'$, the property $S \xrightarrow{\mathscr{Q}}{\twoheadrightarrow} \cdot \xleftarrow{\mathscr{P}}{}^{\equiv} S'$ holds. The base case $S_c \equiv S'$ is immediate. For the inductive case $S \xleftarrow{\mathscr{P}} S_c \xrightarrow{\mathscr{Q}}{\twoheadrightarrow} S'' \xrightarrow{\mathscr{Q}} S'$, we know by the induction hypothesis that there exists a state R, such that $S \xrightarrow{\mathscr{Q}}{\twoheadrightarrow} R \xleftarrow{\mathscr{P}}{}^{\equiv} S''$. From here, it is sufficient to prove that $R \xrightarrow{\mathscr{Q}}{\twoheadrightarrow} \cdot \xleftarrow{\mathscr{P}}{}^{\equiv} S'$ and to use the definition of relation composition in order to conclude. We assume that $S'' \xrightarrow{\mathscr{Q}} R$, otherwise $R \xrightarrow{\mathscr{Q}}{\twoheadrightarrow} \cdot \xleftarrow{\mathscr{P}}{}^{\equiv} S'$ holds trivially. We distinguish two cases: either the rules involved in the local peak $R \xleftarrow{\mathscr{P}} S'' \xrightarrow{\mathscr{Q}} S'$ apply to different parts of S'', or else their applications overlap. In the first case, we use CHR monotonicity to infer $R \xrightarrow{\mathscr{Q}} \cdot \xleftarrow{\mathscr{P}} S''$. In the second case, there must exist a critical peak $R'' \xleftarrow{\mathscr{P}} \cdot \xrightarrow{\mathscr{Q}} S'''$, a state R', and a set of variables $\bar{x}$, such that $R'' \oplus_{\bar{x}} R' \equiv R$, $S''' \oplus_{\bar{x}} R' \equiv S'$. Then by the hypotheses and CHR monotonicity, we obtain the results that $R \xrightarrow{\mathscr{Q}}{\twoheadrightarrow} \cdot \xleftarrow{\mathscr{P}}{}^{\equiv} S'$. □

Proof of Theorem 17
Let $\rightarrow_1 = \xrightarrow{\mathscr{P}}{\twoheadrightarrow}$, $\rightarrow_2 = \xrightarrow{\mathscr{Q}}{\twoheadrightarrow}$. On one hand, by the confluence of $\mathscr{P}$ and $\mathscr{Q}$, we have $(\leftarrow_1 \cdot \rightarrow_1) \subseteq (\rightarrow_1 \cdot \leftarrow_1)$ and $(\leftarrow_2 \cdot \rightarrow_2) \subseteq (\rightarrow_2 \cdot \leftarrow_2)$. (Note that ${\twoheadrightarrow_1} = \rightarrow_1$ and ${\twoheadrightarrow_2} = \rightarrow_2$.) On the other hand, by combining Lemma 18 and case *(i)* of Lemma 4, we infer $(\leftarrow_1 \cdot \rightarrow_2) \subseteq (\rightarrow_2 \cdot \leftarrow_1)$. By a trivial application of Theorem 3, we find that $\rightarrow_{\{1,2\}}$ is confluent. We conclude by noting $\xrightarrow{\mathscr{P}\mathscr{Q}} \subseteq \rightarrow_{\{1,2\}} \subseteq \xrightarrow{\mathscr{P}\mathscr{Q}}{\twoheadrightarrow}$, and apply case *(ii)* of Lemma 4. (It is worth noting that $\rightarrow_{\{1,2\}}$ equals neither $\xrightarrow{\mathscr{P}\mathscr{Q}}$ nor $\xrightarrow{\mathscr{P}\mathscr{Q}}{\twoheadrightarrow}$.) □

https://doi.org/10.1017/S1471068412000270 Published online by Cambridge University Press

6 Conclusion

By employing the decreasing diagrams technique in CHR, we have established a new criterion for CHR confluence that generalizes local and strong confluence criteria. The crux of this novel criterion rests on the distinction between the terminating part (the so-called inductive part) and non-terminating part (the so-called coinductive part) of a program, together with the labeling of transitions by rules. Importantly, we demonstrate that in the particular case of the proposed application of the decreasing diagrams, the check on decreasingness can be restricted to the sole critical pairs, hence making it possible to automatize the process. We also improve on a result about the so-called modularity of confluence, which allows a modular combination of rule-decreasing programs, without loss of confluence.

It is worth saying that all the diagrammatic proofs sketched in the paper have been systematically verified by a prototype of a diagrammatic confluence checker. In practice, this checker automatically generates all the critical pairs of a program provided with an admissible order, then using user-defined *tactics* (finit sets of reductions) tries to join these while respecting rule-decreasingness.

Current work involves investigating the development of heuristics to automatically infer rule-decreasingness without human interaction. We also plan to develop a new completion procedure based on the criterion presented here. Because duplicate removal is an important programming idiom of CHR, the development of new confluence-proof techniques capable of dealing with confluent but non-rule-decreasing programs, like those given in Example 16, is also worth investigating.

References

Abdennadher, S. 1997. Operational semantics and confluence of constraint propagation rules. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP)*. LNCS, vol. 1330. Springer, Berlin, Germany, 252–266.

Abdennadher, S. and Frühwirth, T. 1999. Operational equivalence of CHR programs and constraints. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP)*. LNCS, vol. 1713. Springer, Berlin, Germany, 43–57.

Abdennadher, S., Frühwirth, T. and Meuss, H. 1996. On confluence of Constraint Handling Rules. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP)*. LNCS, vol. 1118. Springer, Berlin, Germany, 1–15.

Abdennadher, S., Frühwirth, T. W. and Meuss, H. 1999. Confluence and semantics of constraint simplification rules. *Constraints 4*, 2, 133–165.

Duck, G. J., Stuckey, P. J., García de la Banda, M. and Holzbaur, C. 2005. The refined operational semantics of Constraint Handling Rules. In *Proceedings of the International Conference on Logic Programming (ICLP)*. LNCS, vol. 3668. Springer, Berlin, Germany, 90–104.

Duck, G. J., Stuckey, P. J. and Sulzmann, M. 2007. Observable confluence for Constraint Handling Rules. In *Proceedings of the International Conference on Logic Programming (ICLP)*. LNCS, vol. 4670. Springer, Berlin, Germany, 224–239.

Frühwirth, T. 1998. Theory and practice of Constraint Handling Rules. *Journal of Logic Programming, Special Issue on Constraint Logic Programming 37*, 1–3, 95–138.

Frühwirth, T. 2005. Parallelizing union-find in Constraint Handling Rules using confluence. In *Proceedings of the International Conference on Logic Programming (ICLP)*. LNCS, vol. 3668. Springer, Berlin, Germany, 113–127.

Frühwirth, T. 2009. *Constraint Handling Rules*. Cambrige University Press, Cambrige, UK.

Guy, R. 2004. *Unsolved Problems in Number Theory*. Problem Books in Mathematics. Springer, Berlin, Germany.

Haemmerlé, R. 2011a. (Co)-Inductive semantics for Constraint Handling Rules. *Theory and Practice of Logic Programming, 27th Int'l. Conference on Logic Programming (ICLP'11) Special Issue 11*, 4–5, 593–609.

Haemmerlé, R. 2011b. Observational equivalences for linear logic concurrent constraint languages. *Theory and Practice of Logic Programming, 27th Int'l. Conference on Logic Programming (ICLP'11) Special Issue 11*, 4–5, 469–485.

Haemmerlé, R. and Fages, F. 2007. Abstract critical pairs and confluence of arbitrary binary relations. In *Proceedings of the International Conference on Rewriting Techniques and Applications (RTA)*. LNCS, vol. 4533. Springer, Berlin, Germany, 214–228.

Haemmerlé, R., López, P. and Hermenegildo, M. 2011. CLP projection for Constraint Handling Rules. In *Proceedings of the International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (PPDP)*. ACM Press, New York, NY, USA, 137–148.

Hirokawa, N. and Middeldorp, A. 2010. Decreasing diagrams and relative termination. In *IJCAR*. LNCS, vol. 6173. Springer, Berlin, Germany, 487–501.

Huet, G. 1980. Confluent reductions: Abstract properties and applications to term rewriting systems: Abstract properties and applications to term rewriting systems. *Journal of the ACM 27*, 4, 797–821.

Jouannaud, J.-P. and van Oostrom, V. 2009. Diagrammatic confluence and completion. In *Proceedings of 36th Internatilonal Collogquium on Automata, Languages and Programming: ICALP 2009*. LNCS, vol. 5556. Springer, Berlin, Germany, 212–222.

Meister, M. 2006. Fine-grained parallel implementation of the preflow-push algorithm in CHR. In *Workshop on Logic Programming (WLP)*. INFSYS Research report 1843-06-02. T.U.Wien, Vienna, Austria, 172–181.

Milner, R. 1999. *Communicating and Mobile Systems - the Pi-Calculus*. Cambrige University Press, Cambrige, UK.

Newman, M. H. A. 1942. On theories with a combinatorial definition of "equivalence". *Annals of Mathematics 43*, 2.

Raiser, F., Betz, H. and Frühwirth, T. 2009. Equivalence of CHR states revisited. In *Proceedings of the International Workshop on Constraint Handling Rules (CHR)*. Report CW 555. Kath. University Leuven, Leuven, Belgium, 34–48.

Raiser, F. and Tacchella, P. 2007. On confluence of non-terminating CHR programs. In *Proceedings of the International Workshop on Constraint Handling Rules (CHR)*. 63–76.

Sneyers, J., Schrijvers, T. and Demoen, B. 2009. The computational power and complexity of Constraint Handling Rules. *ACM Transactions on Programming Languages and Systems 31*, 2.

Terese. 2003. *Term Rewriting Systems*. Cambrige University Press, Cambrige, UK.

van Oostrom, V. 1994. Confluence by decreasing diagrams. *Theoretical Computer Science 126*, 2, 259–280.

van Oostrom, V. 2008. Confluence by decreasing diagrams converted. In *Proceedings of the International Conference on Rewriting Techniques and Applications (RTA)*. LNCS. Springer, Berlin, Germany, 306–320.