On Confluence of Non-terminating CHR Programs

Frank Raiser[1] and Paolo Tacchella[2]

[1] Faculty of Engineering and Computer Sciences, University of Ulm, Germany
`Frank.Raiser@uni-ulm.de`
[2] Dipartimento di Scienze dell'Informazione, Università di Bologna, Italy
`Paolo.Tacchella@cs.unibo.it`

Abstract. Confluence is an important property for any kind of rewrite system including CHR, which is a general-purpose declarative committed-choice language consisting of multi-headed guarded rules. CHR can yield a confluence problem, because of non-determinism in the choice of rules using the abstract semantics. Confluence in CHR is an ongoing research topic, because it provides numerous benefits for implementations. However, for non-terminating CHR programs confluence is generally undecidable. In this paper we apply the so-called Strong Church-Rosser property to CHR. This allows determination of confluence for a subset of non-terminating CHR programs.

1 Introduction

Confluence is an important property for any kind of rewrite system. Many confluence-related results have been developed for term rewriting systems (TRS) [1–3] and it also plays an important role for Constraint Handling Rules (CHR) [4, 5].

CHR is a concurrent committed-choice constraint logic programming language consisting of guarded rules, which transform multi-sets of atomic formulas (constraints) into simpler ones until exhaustion [6].

Given a CHR state $\mathcal{S}$ and multiple applicable rules, the standard semantics non-deterministically applies a rule. In general this means that, at the end of the computation, different result states $\mathcal{S}_1$ and $\mathcal{S}_2$ are obtained depending on the choice of the applied rules. Confluence is an important property, because it is desirable for the program to have a unique result for a given input. In CHR confluence is even more important, as it also implies consistency of the logical reading of the program [4].

Confluence for CHR is a well-known problem and it is discussed in [4, 5]. The most interesting case is when there are two rules r_1 and r_2 which need at least one common constraint to be applied and the application of one of these rules deletes the considered constraint. It is proved that confluence of minimal states, which fulfill these characteristics, grants confluence of the whole program if the computation is terminating. If the CHR program satisfies the confluence property, all states can be transformed by rule applications to yield states which are variants of each other whereas two states are variants if they are equal modulo a variable rename.

To the best of our knowledge no results about confluence for non-terminating CHR programs are obtained yet. In spite of this, confluence for non-terminating programs is a desirable property from a practical point of view. Many distributed concurrent algorithms are non-terminating including operating systems, control programs, and agents. As a motivating example for these kinds of programs the next section presents a CHR program for the well-known confluent and non-terminating dining philosophers synchronization problem. However, confluence cannot be proved for this example using existing results from [4].

In CHR a confluent program allows easy parallelization, which often makes confluence analysis a necessity. In [7] a parallel version of the classical union-find algorithm was developed for CHR based on results of confluence analyses. Similarly a confluent CHR program for the preflow-push algorithm allowed its efficient parallelization [8]. Furthermore, a CHR program can be considered to be an online algorithm, thus making confluence analysis an important topic for another large range of applications.

There are numerous similarities between term rewriting systems and CHR: both of them are rewrite systems working on terms and constraints respectively, rewriting them based on a set of rules. Confluence properties are interesting for both due to the non-deterministic choice of rule applications. Nevertheless, there are also significant differences to be found between these two systems: in a TRS the terms, rules are applied to, are always ground, while CHR allows non-ground initial goals as well. The rewriting of terms in a TRS is always locally restricted to the replacement of a subterm, whereas in CHR rules can be applied to constraints independent of their position in the store. Finally, the existence of propagation rules, which add new constraints as a logical consequence of existing constraints, has no equivalent in a term rewriting system.

Because of the similarities between TRSs and CHR, ideas used for TRSs can sometimes be adapted to CHR. In particular, this work shifts the Strong Church Rosser (SCR) property [1] from TRSs to CHR. This property is a stricter version of local confluence, which is used for confluence analysis in terminating CHR programs.

The following section introduces necessary preliminaries about CHR, Sect. 3 contains formal definitions and results about confluence for non-terminating CHR programs, based on the SCR property. Finally, Sect. 4 concludes with related and future works.

2 Preliminaries

This section presents the syntax and operational semantics of Constraint Handling Rules [6, 9, 10]. Constraints are first-order predicates [11] which we separate into *built-in constraints* and *user-defined constraints*. Built-in constraints are solved by an underlying constraint solver, that is driven by a constraint theory CT, while user-defined constraints are managed by a CHR program. In the following, let a lowercase letter represent a single constraint and an uppercase letter represent a conjunction of constraints.

2.1 CHR Syntax

A CHR program P is a finite set of CHR rules. There are three different kinds of CHR rules: simplification, propagation, and simpagation ones.
A **simplification** rule has the form:

$$r@H \Leftrightarrow D \mid B$$

A **propagation** rule has the form:

$$r@H \Rightarrow D \mid B$$

A **simpagation** rule has the form:

$$r@H_1 \setminus H_2 \Leftrightarrow D \mid B,$$

where r is a unique identifier of the rule, H, H_1 and H_2 are non-empty multisets of user-defined constraints called *heads*, D is a possibly empty multiset of built-in constraints called *guard* and B is a possibly empty multiset of built-in and user-defined constraints called *body*. A *CHR goal* is a multiset of both user-defined and built-in constraints.

Note that the behaviour of a propagation rule can be simulated by a simplification rule: let $H \Rightarrow D \mid B$ be a propagation rule. Then the simplification rule associated to it is $H \Leftrightarrow D \mid H \wedge B$. Similarly a simpagation rule $H_1 \setminus H_2 \Leftrightarrow D \mid B$ is associated with $H_1 \wedge H_2 \Leftrightarrow D \mid H_1 \wedge B$. In Sect.3 and subsequently, we only consider simplification rules because of these observations.

After the CHR syntax exposition we can introduce the following motivating example about dining philosophers.

Example 1. The following exemplary CHR program is a version of the dining philosophers problem. It models three philosophers and their corresponding chopsticks. Every rule $p_i (i = 1, 2, 3)$ expects the constraints for a thinking philosopher and his two chopsticks to be present. The application of such a rule results in that philosopher beginning to eat using the two chopsticks. The application of the rules $p'_i (i = 1, 2, 3)$ then results in the philosopher to stop eating and revert to thinking. Additionally the two chopsticks become available again. The usual dead-lock problem connected to the chopstick usage doesn't occur because of the atomicity of the application of a CHR rule. For the purpose of simplicity we do not consider the problem of starvation. The start state of this program $Q = \{p_1, p_2, p_3, p'_1, p'_2, p'_3\}$ is

$$\langle ph_1(\mathit{think}) \wedge ph_2(\mathit{think}) \wedge ph_3(\mathit{think}) \wedge c_1 \wedge c_2 \wedge c_3, \top\rangle.$$

$$
\begin{array}{lll}
p_1 & @ & ph_1(think) \wedge c_1 \wedge c_3 \Leftrightarrow ph_1(eat). \\
p'_1 & @ & ph_1(eat) \Leftrightarrow ph_1(think) \wedge c_1 \wedge c_3. \\
p_2 & @ & ph_2(think) \wedge c_1 \wedge c_2 \Leftrightarrow ph_2(eat). \\
p'_2 & @ & ph_2(eat) \Leftrightarrow ph_2(think) \wedge c_1 \wedge c_2. \\
p_3 & @ & ph_3(think) \wedge c_2 \wedge c_3 \Leftrightarrow ph_3(eat). \\
p'_3 & @ & ph_3(eat) \Leftrightarrow ph_3(think) \wedge c_2 \wedge c_3.
\end{array}
$$

2.2 CHR Operational Semantics

The operational semantics is based on an underlying constraint theory CT for the built-in constraints which has to contain at least the syntactic equality (denoted by $\doteq$), `true` (denoted by $\top$), and `false` (representing inconsistent built-in constraints). A *state* $\mathcal{S}$ is a pair $\langle G, C \rangle$ where G is a multi-set conjunction of CHR and built-in constraints and it is called *goal store* and C is a set conjunction of built-in constraints and it is called *built-in constraint store.* An *initial state* is a pair $\langle G, \top \rangle$ [10]. We further define the function *vars* to map from a conjunction of constraints to the set of variables used in the conjunction.

Let us now consider the transitions introduced in Table 1, which represent the CHR operational semantics. The $\mathcal{N}$ function represents the normalization function introduced by [5] omitting token elimination as this work only considers simplification rules.

Solve moves a built-in constraint from the goal store to the built-in constraint store;

Simplify uses the rule $r@H' \Leftrightarrow D \mid B$ provided that a matching substitution θ exists, such that $H = (H')\theta$ and D is entailed by the built-in constraint store of the computation; H is replaced by B and $(H \doteq H') \wedge D$ is added to the built-in constraint store;

Propagate uses the rule $r@H' \Rightarrow D \mid B$ provided that a matching substitution θ exists, such that $H = (H')\theta$ and D is entailed by the built-in constraint store of the computation; B is added to the goal store and $(H \doteq H') \wedge D$ is added to the built-in constraint store;

Simpagate uses the rule $r@H'_1 \backslash H'_2 \Leftrightarrow D \mid B$ provided that a matching substitution θ exists, such that $(H_1, H_2) = (H'_1, H'_2)\theta$ and D is entailed by the built-in constraint store of the computation; H_2 is replaced by B and $((H_1, H_2) \doteq (H'_1, H'_2)) \wedge D$ is added to the built-in constraint store.

From the transition system introduced in Table 1 follows that only two conditions are needed to fire a rule r on a conjunction of constraints G: the head of the chosen rule must match with some constraints in G and the guard of r must be entailed by the built-in constraint store. It is clear that for a fixed goal there could be more than one rule that can be fired and it is also clear that a rule could be applied to different constraints in the goal. This yields two types of non-determinism: non-deterministic rule selection and non-deterministic constraint selection. For the notion of confluence we are especially interested in the non-determinism involved in rule selections.

Solve	$\dfrac{CT \models c \wedge C \leftrightarrow C' \text{ and } c \text{ is a built-in constraint}}{\langle \{c\} \wedge G, C\rangle \mapsto \mathcal{N}(\langle G, C'\rangle)}$
Simplify	$\dfrac{r@H' \Leftrightarrow D \mid B \in P \quad x = vars(H') \quad CT \models C \rightarrow \exists_x((H \doteq H') \wedge D)}{\langle G \wedge H, C\rangle \mapsto \mathcal{N}(\langle B \wedge G, (H \doteq H') \wedge C \wedge D\rangle)}$
Propagate	$\dfrac{r@H' \Rightarrow D \mid B \in P \quad x = vars(H') \quad CT \models C \rightarrow \exists_x((H \doteq H') \wedge D)}{\langle G \wedge H, C\rangle \mapsto \mathcal{N}(\langle B \wedge G \wedge H, (H \doteq H') \wedge C \wedge D\rangle)}$
Simpagate	$\dfrac{r@H'_1 \setminus H'_2 \Leftrightarrow D \mid B \in P \quad x = vars(H'_1 \wedge H'_2) \quad CT \models C \rightarrow \exists_x(((H_1, H_2) \doteq (H'_1, H'_2)) \wedge D)}{\langle G \wedge H_1 \wedge H_2, C\rangle \mapsto \mathcal{N}(\langle B \wedge G \wedge H_1, ((H_1, H_2) \doteq (H'_1, H'_2)) \wedge C \wedge D\rangle)}$

Table 1. The transition system for the original CHR semantics

Two conjunctions of constraints $H_1 = H'_1 \wedge C_1$ and $H_2 = H'_2 \wedge C_2$, with H'_i a conjunction of CHR constraints, C_i a conjunction of built-in constraints in H_i and $i = \{1, 2\}$, are said to be *CHR variants*, and we write $H_1 \simeq H_2$, if $H'_1 \doteq H'_2\theta$ and $CT \models C_1 \leftrightarrow C_2\theta$, where θ is a renaming of local variables in H_2, i.e. variables which do not occur in the initial goal. Note that the definition of CHR variants is symmetric, such that there is also a θ' with $H_1\theta' \doteq H_2$. In the following, we consider only CHR variants and simply call them variants.

The transition system in Table 1 allows us to use the $\mapsto$ relation between states with $\mathcal{S} \mapsto \mathcal{S}'$ meaning any of the possible transitions. Furthermore, we make use of the following extensions: $\mathcal{S} \mapsto^k \mathcal{S}_k (k > 0)$ is used as a shortcut for exactly k transitions between states $\mathcal{S}$ and $\mathcal{S}_k$. More precisely $\mapsto^k = \mapsto \cup \mapsto^{k-1}$ where $\mathcal{S} \mapsto^0 \mathcal{S}'$ means $\mathcal{S} \simeq \mathcal{S}'$. Finally, we define $\mapsto^\varepsilon = \mapsto^0 \cup \mapsto$ and $\mapsto^* = \bigcup_{i=0}^{\infty} \mapsto^i$. Note that $\mapsto^*$ always denotes a finite number of transitions.

3 Confluence Properties

In this section we investigate confluence of CHR programs and we present our results on confluence of non-terminating CHR programs. First of all we formally introduce the notion of confluence:

Definition 1 (Confluence). *A CHR program is* confluent*, if for all states $\mathcal{S}$ with $\mathcal{S} \mapsto^* \mathcal{S}_1$ and $\mathcal{S} \mapsto^* \mathcal{S}_2$ there exist states $\mathcal{S}'_1, \mathcal{S}'_2$ such that $\mathcal{S}_1 \mapsto^* \mathcal{S}'_1 \simeq \mathcal{S}'_2 \;{}^*\!\!\leftarrow\!\!\!\mid \mathcal{S}_2$. Then the states $\mathcal{S}_1$, $\mathcal{S}_2$ are called* joinable.

Fig. 2 (a) shows a graphical representation of the previous definition, where dashed arrows represent the existence of a CHR computation with ≥ 0 steps.

Example 2 (confluent philosophers). The previous program for dining philosophers is intuitively confluent: no matter which rules are applied the overall situation of three philosophers and three chopsticks remains and rules $p'_i (i = 1, 2, 3)$ can always be applied to get back to a state with only thinking philosophers and all chopsticks being available. However, it is hard to proof confluence of this program using the above definition, and the other existing methods presented below are not applicable due to its non-termination. With the help of our results though, we prove the program's confluence in Example 7.

The following exemplary CHR program based on an example in [1] is used to point out characteristics of non-terminating CHR programs with respect to previous confluence results for terminating ones.

Example 3. Let $P = \{r_1, r_2, r_3, r_4, r_5, r_6\}$ be a CHR program composed of the following rules

$$\begin{aligned}
r_1 &\ @\ a(X) \Leftrightarrow b(X).\\
r_2 &\ @\ a(X) \Leftrightarrow e(X).\\
r_3 &\ @\ b(X) \Leftrightarrow a(X).\\
r_4 &\ @\ b(X) \Leftrightarrow f(X).\\
r_5 &\ @\ e(X) \Leftrightarrow e(X) \wedge g(X).\\
r_6 &\ @\ f(X) \Leftrightarrow f(X) \wedge g(X).
\end{aligned}$$

whose behaviour is illustrated in Fig. 1. The circles contain the constraints that are rewritten. Note that rules r_5 and r_6 could also be written as propagation rules $r_5\ @\ e(X) \Rightarrow g(X).$ and $r_6\ @f(X)\ \Rightarrow g(X)$ as explained before.

It is clear from rules r_1 and r_3 of Example 3, that the exemplary CHR program P is non-terminating. Due to the circular nature of these rules we refer to this example as the *circular* example, to avoid confusion with the previous dining philosophers example. Furthermore, the program is not confluent, because $\langle a(X), \top\rangle \mapsto^* \langle e(X), \top\rangle$ and $\langle a(X), \top\rangle \mapsto^* \langle f(X), \top\rangle$ with no possibility to join these two states, as only rules r_5 and r_6 are applicable afterwards.

When analysing confluence of a CHR program it is unfeasible to directly prove Definition 1, as an infinite number of possible states S_1, S_2 has to be checked for an infinite number of initial states S. It has been shown, however, with the help of Newman's Lemma [12, 13], that the analysis of a finite number of minimal states, where more than one rule is applicable, suffices to determine confluence of terminating CHR programs. To this end, the concept of critical pairs was introduced in [4] according to the following definition:

Definition 2 (Critical pair). *Let*

$$r_1 @ H_1 \Leftrightarrow G_1 \mid B_1.$$

$$r_2 @ H_2 \Leftrightarrow G_2 \mid B_2.$$

be (not necessarily different) simplification rules, whose variables have been renamed apart. Let $H'_i \wedge A_i$ be separations of the heads H_i, such that $\forall i \in \{1, 2\}$: $(H'_i \wedge A_i) \doteq H_i$, with A_i being non-empty conjunctions and $CT \models \exists((A_1 \doteq A_2) \wedge G_1 \wedge G_2)$, then a critical ancestor state *of r_1 and r_2 is*

$$\langle H'_1 \wedge A_1 \wedge H'_2, (A_1 \doteq A_2) \wedge G_1 \wedge G_2\rangle$$

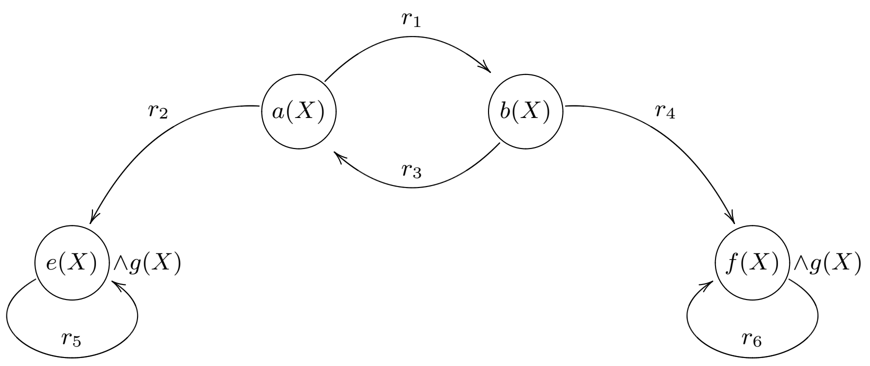

Fig. 1. Graphical representation of Example 3

Let S be a critical ancestor state of r_1 *and* r_2*. If* $\mathcal{S} \overset{r_1}{\mapsto} \mathcal{S}_1$ *and* $\mathcal{S} \overset{r_2}{\mapsto} \mathcal{S}_2$*, then the tuple* $(\mathcal{S}_1, \mathcal{S}_2)$ *is a* critical pair *of* r_1 *and* r_2*. A critical pair* $(\mathcal{S}_1, \mathcal{S}_2)$ *is* joinable *if* $\mathcal{S}_1$ *and* $\mathcal{S}_2$ *are joinable.*

In the following, we present the critical pair analysis of Examples 1 and 3.

Example 4 (Critical pairs). The above Example 3 has exactly two critical pairs, due to rules r_1, r_2 and r_3, r_4. The ancestor states are respectively $\langle a(X), \top\rangle$ and $\langle b(X), \top\rangle$ and the corresponding critical pairs are $(\langle b(X), \top\rangle, \langle e(X), \top\rangle)$ and $(\langle a(X), \top\rangle, \langle f(X), \top\rangle)$.

For the dining philosophers example the only constraints shared by different rule heads are the chopstick constraints, yielding exactly three critical ancestor states: $\langle ph_1(think) \wedge c_1 \wedge c_3 \wedge ph_3(think) \wedge c_2, \top\rangle$, $\langle ph_1(think) \wedge c_1 \wedge c_3 \wedge ph_2(think) \wedge c_2, \top\rangle$, and $\langle ph_2(think) \wedge c_2 \wedge c_3 \wedge ph_3(think) \wedge c_1, \top\rangle$. Additionally, rules overlapping with themselves produce another six critical ancestor states for every rule $p_i (i = 1, 2, 3)$. However, these states contain multiple $ph_i(think)$ or c_i constraints. In practical applications the initial input goals consist only of the three philosophers and their corresponding chopsticks though, making these states unreachable.

For a feasible confluence check, a stricter kind of confluence with respect to Definition 1 is introduced, which can be applied to critical ancestor states and critical pairs:

Definition 3 (Local Confluence). *A CHR program is* locally confluent*, if for all states* $\mathcal{S}$ *with* $\mathcal{S} \mapsto \mathcal{S}_1$ *and* $\mathcal{S} \mapsto \mathcal{S}_2$ *there exist states* $\mathcal{S}_1', \mathcal{S}_2'$ *such that* $\mathcal{S}_1 \mapsto^* \mathcal{S}_1' \simeq \mathcal{S}_2' \; {}^*\!\!\leftarrow\!\! \mathcal{S}_2$.

Based on the critical pair and local confluence definitions the following important result for confluence of terminating CHR programs has been established in [13]:

Theorem 1 (Critical pair based confluence). *A terminating CHR program is confluent if and only if all critical pairs are joinable.*

The following example shows the importance of the additional requirement of termination for local confluence to yield confluence of the whole program. Even though all critical pairs are joinable, the non-terminating program is in fact not confluent.

Example 5 (Local confluence insufficient). The circular CHR program is locally confluent, i.e. all critical pairs are joinable. Let us consider the following critical pair $(\langle b(X), \top\rangle, \langle e(X), \top\rangle)$. The two states are joinable as can be seen from $\langle b(X), \top\rangle \overset{r_3}{\mapsto} \langle a(X), \top\rangle \overset{r_2}{\mapsto} \langle e(X), \top\rangle$. Analogously the second critical pair consisting of the states $\langle a(X), \top\rangle$ and $\langle f(X), \top\rangle$ can be joined by these computation steps: $\langle a(X), \top\rangle \overset{r_1}{\mapsto} \langle b(X), \top\rangle \overset{r_4}{\mapsto} \langle f(X), \top\rangle$. Nevertheless the program is not confluent as stated before.

Using the criteria of Theorem 1 the confluence property of a terminating CHR program can automatically be decided [13]. To the best of our knowledge there are no investigations into determining confluence of non-terminating CHR programs. Similarly to terminating CHR programs, however, it is possible to transfer results from term rewriting systems, where for example the Strong Church-Rosser property is used as one possible criteria [1]. The following definition applies this property to CHR programs:

Definition 4 (Strong Church-Rosser property). *A CHR program has the* Strong Church-Rosser *(SCR) property, if for all states $\mathcal{S}, \mathcal{S}_1, \mathcal{S}_2$ with $\mathcal{S} \mapsto \mathcal{S}_1$ and $\mathcal{S} \mapsto \mathcal{S}_2$ there exist states $\mathcal{S}_1', \mathcal{S}_2', \mathcal{S}_1''$, and $\mathcal{S}_2''$, such that $\mathcal{S}_1 \mapsto^* \mathcal{S}_1' \simeq \mathcal{S}_2' \ {}^{\varepsilon}\!\!\leftarrow\!\!\!\!\mid \mathcal{S}_2$, and $\mathcal{S}_1 \mapsto^{\varepsilon} \mathcal{S}_1'' \simeq \mathcal{S}_2'' \ {}^{*}\!\!\leftarrow\!\!\!\!\mid \mathcal{S}_2$. (Fig. 2 (b)) A CHR program with the SCR property is also called* strongly confluent.

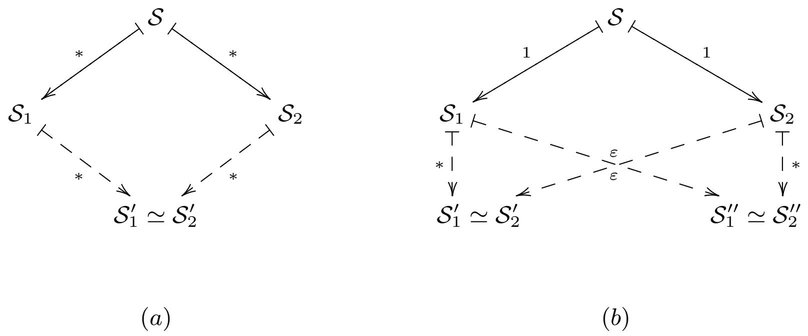

Fig. 2. Confluence and Strong Church-Rosser property

The following definition introduces the concept of a strongly closed program as the criteria which has to be verified for analysing a non-terminating CHR program for confluence. Note that the construction of the critical ancestor state is a syntactical notion, but the remaining part of the analysis involves argumentation about the operational behavior.

Definition 5 (Strong closedness). *A CHR program is called* strongly closed *if for every critical pair* $(\mathcal{S}_1, \mathcal{S}_2)$ *of the critical ancestor state* $\mathcal{S}$ *there exist states* $\mathcal{S}_1', \mathcal{S}_2', \mathcal{S}_1''$ *and* $\mathcal{S}_2''$ *such that* $\mathcal{S}_1 \mapsto^* \mathcal{S}_1' \simeq \mathcal{S}_2' \ {}^{\varepsilon}\!\!\leftarrow\!\!\shortmid \mathcal{S}_2$*, and* $\mathcal{S}_1 \mapsto^{\varepsilon} \mathcal{S}_1'' \simeq \mathcal{S}_2'' \ {}^{*}\!\!\leftarrow\!\!\shortmid \mathcal{S}_2$*.*

Example 6 (No SCR program). We have seen in Example 5 that the CHR program of Example 3, is locally confluent. Furthermore this program is not strongly closed and thus not strongly confluent: Consider again the critical pair $(\mathcal{S}_1, \mathcal{S}_2) = (\langle b(X), \top\rangle, \langle e(X), \top\rangle)$ resulting from the ancestor state $\langle a(X), \top\rangle$. Due to Definition 4 and Definition 5 we have to show that the SCR property for the critical ancestor state $\langle a(X), \top\rangle$ does not hold. Let us consider the case $\mathcal{S}_1 \mapsto^{\varepsilon} \mathcal{S}_1', \mathcal{S}_2 \mapsto^* \mathcal{S}_2'$. Note that $\mathcal{S}_2$ allows only rule r_5 to be applied, thus the only states reachable from $\mathcal{S}_2$ consist of an $e(X)$ constraint and any number of $g(X)$ constraints. If $\mathcal{S}_1 \mapsto^0 \mathcal{S}_1'$, i.e. $\mathcal{S}_1 \simeq \mathcal{S}_1'$, then we know that $\mathcal{S}_2 \not\mapsto^* \mathcal{S}_2' \simeq \mathcal{S}_1'$, because the necessary constraint $b(X)$ can never be generated from $\mathcal{S}_2$. If $\mathcal{S}_1 \mapsto \mathcal{S}_1'$, there are two cases: $\mathcal{S}_1' = \langle a(X), \top\rangle$ and $\mathcal{S}_1' = \langle f(X), \top\rangle$. However, in both cases it holds again that $\mathcal{S}_2 \not\mapsto^* \mathcal{S}_2' \simeq \mathcal{S}_1'$. Therefore we have shown, that the exemplary CHR program is not strongly closed.

The following theorem proves the equivalence between the strongly closed property and the strongly confluent one of a program.

Theorem 2 (Strong confluence). *A (possibly non-terminating) CHR program is strongly confluent if and only if it is strongly closed.*

Proof. The two implications have to be proved:

"⇒": *This implication follows directly from Definition 4 and Definition 5.*

"⇐": *This implication follows from the monotonicity property of CHR analogous to the proof of our Theorem 1 that is given in [13]. For the proof a state* $\mathcal{S}$ *with* $\mathcal{S} \mapsto \mathcal{S}_1$ *and* $\mathcal{S} \mapsto \mathcal{S}_2$ *is considered. There are three different combinations of computation steps possible: In the case of a* **Solve+Solve** *or* **Solve+Simplify** *combination it can be shown that the two computations can be performed in either order, such that* $\mathcal{S} \mapsto \mathcal{S}_i \mapsto \mathcal{S}' (i = 1, 2)$*. For a* **Simplify+Simplify** *combination it is possible that the two rules are applied to distinct CHR constraints. In such a case the rules can be applied consecutively again to yield strong confluence. In the other case the proof idea is to determine the overlap of head constraints of both of the involved rules and construct a critical ancestor state from it. By assumption the critical pair resulting from this state is strongly joinable and it is then shown, that all computation steps applicable to the critical pair are also applicable to* $(\mathcal{S}_1, \mathcal{S}_2)$ *and result in a joined state.* □

Analogous to the step from local confluence and termination to confluence of the whole program, Theorem 3 uses strong confluence instead. For its proof the following lemma is required:

Lemma 1. *Let $\mathcal{S}$ be a state with $\mathcal{S} \mapsto^k \mathcal{S}'_k$ and $\mathcal{S} \mapsto \mathcal{S}_1$ for a strongly confluent program, there exist $\mathcal{S}'_C$ and $\mathcal{S}_C$ with $\mathcal{S}'_k \mapsto^\varepsilon \mathcal{S}'_C \simeq \mathcal{S}_C \;{}^*\!\!\leftarrow\!\!\dashv \mathcal{S}_1$. Fig. 3(a) shows the situation given in this lemma.*

Proof. By induction over k:

Base step *for $k = 1$ the claim follows directly from the SCR property as shown in Fig. 3(b).*

Inductive step *we suppose that the claim holds for $k-1$ (inductive hypothesis) and we prove that it holds also for k.*

- **Fig. 3(c)** *considers the case $\mathcal{S}'_{k-1} \mapsto^0 \mathcal{S}'_C$, which means that $\mathcal{S}'_{k-1} \simeq \mathcal{S}'_C$ and therefore the required property holds, because then $\mathcal{S}'_{k-1} \simeq \mathcal{S}'_C \simeq \mathcal{S}_C \mapsto \mathcal{S}'_k$.*
- **Fig. 3(d)** *considers the case $\mathcal{S}'_{k-1} \mapsto \mathcal{S}'_C$. In this case the required property holds by SCR.* □

Theorem 3 (SCR yields confluence). *A (possibly non-terminating) strongly confluent CHR program is confluent.*

Proof. We consider a strongly confluent CHR program. In order to prove confluence let $\mathcal{S}$ be a state with $\mathcal{S} \mapsto^k \mathcal{S}'_k$ and $\mathcal{S} \mapsto^n \mathcal{S}_n$. We have to show that the states $\mathcal{S}'_k$ and $\mathcal{S}_n$ can be joined as given in Fig. 4(a). Proof follows by induction over n and by considering Lemma 1:

Base step *we suppose that $n = 1$, then the joinability is given by applying the above Lemma 1 as shown in Fig. 4(b).*

Inductive step *we suppose that $\mathcal{S}'_k \mapsto^* \mathcal{S}'_{C'}$ and $\mathcal{S}_{n-1} \mapsto^* \mathcal{S}_{C'}$ and by inductive hypothesis we suppose also that $\mathcal{S}'_{C'} \simeq \mathcal{S}_{C'}$, which means that $\mathcal{S}'_k$ and $\mathcal{S}_{n-1}$ are joinable. In the following, we distinguish two possible cases for the transition $\mathcal{S}_{n-1} \mapsto^* \mathcal{S}_{C'}$:*

- $\mathcal{S}_{n-1} \mapsto^0 \mathcal{S}_{C'}$**:** *in this case $\mathcal{S}_{n-1} \simeq \mathcal{S}_{C'}$ which means that the same rule can be applied to both $\mathcal{S}_{n-1}$ and $\mathcal{S}_{C'}$.*
- $\mathcal{S}_{n-1} \mapsto^l \mathcal{S}_{C'} (l > 0)$**:** *the proof is given by a direct application of Lemma 1 to the states $\mathcal{S}_n$ and $\mathcal{S}'_{C'}$ or $\mathcal{S}_{C'}$ as depicted in Fig. 4(c).* □

Corollary 1 (closedness yields confluence). *A (possibly non-terminating) strongly closed CHR program is confluent.*

Proof. The proof of this corollary follows directly from Theorem 2 and Theorem 3. □

The following example is an application of the previously obtained results.

Example 7 (strongly confluent dining). In the case of the dining philosophers example we can now investigate the critical pairs for their strong closedness.

Beginning with the first critical ancestor state $\mathcal{S} = \langle ph_1(think) \wedge c_1 \wedge c_3 \wedge ph_3(think) \wedge c_2, \top\rangle$ we get the critical pair $(\mathcal{S}_1, \mathcal{S}_2) = (\langle ph_1(eat) \wedge ph_3(think) \wedge$

$$\begin{tikzcd}
& \mathcal{S} \arrow[dl, mapsto, "k"'] \arrow[dr, mapsto, "1"] & \\
\mathcal{S}'_k \arrow[dr, dashed, mapsto, "\varepsilon"'] & & \mathcal{S}_1 \arrow[dl, dashed, mapsto, "*"] \\
& \mathcal{S}'_C \simeq \mathcal{S}_C &
\end{tikzcd}$$

(a)

$$\begin{tikzcd}
& \mathcal{S} \arrow[dl, mapsto, "1"'] \arrow[dr, mapsto, "1"] & \\
\mathcal{S}'_1 \arrow[dr, dashed, mapsto, "\varepsilon"'] & SCR & \mathcal{S}_1 \arrow[dl, dashed, mapsto, "*"] \\
& \mathcal{S}'_C \simeq \mathcal{S}_C &
\end{tikzcd}$$

(b)

$$\begin{tikzcd}
& \mathcal{S} \arrow[ddl, mapsto, "k-1"'] \arrow[dr, mapsto, "1"] & \\
& & \mathcal{S}_1 \arrow[ddl, dashed, mapsto, "*"] \\
\mathcal{S}'_{k-1} \arrow[d, mapsto, "1"'] \arrow[dr, dashed, mapsto, "0"] & & \\
\mathcal{S}'_k & \mathcal{S}'_C \simeq \mathcal{S}_C \arrow[l, mapsto, "1"] &
\end{tikzcd}$$

(c)

$$\begin{tikzcd}
& \mathcal{S} \arrow[ddl, mapsto, "k-1"'] \arrow[dr, mapsto, "1"] & \\
& & \mathcal{S}_1 \arrow[ddl, dashed, mapsto, "*"] \\
\mathcal{S}'_{k-1} \arrow[dr, dashed, mapsto, "1"] \arrow[dd, mapsto, "1"'] & & \\
& \mathcal{S}'_{C'} \simeq \mathcal{S}_{C'} \arrow[dd, dashed, mapsto, "*"] & \\
\mathcal{S}'_k \arrow[dr, dashed, mapsto, "\varepsilon"'] & SCR & \\
& \mathcal{S}'_C \simeq \mathcal{S}_C &
\end{tikzcd}$$

(d)

Fig. 3. Proof diagrams of Lemma 1

$$\begin{tikzcd}
& \mathcal{S} \arrow[dl, mapsto, "k"'] \arrow[dr, mapsto, "n"] & \\
\mathcal{S}'_k \arrow[dr, dashed, mapsto, "*"'] & & \mathcal{S}_n \arrow[dl, dashed, mapsto, "*"] \\
& \mathcal{S}'_C \simeq \mathcal{S}_C &
\end{tikzcd}$$

(a)

$$\begin{tikzcd}
& \mathcal{S} \arrow[dl, mapsto, "k"'] \arrow[r, mapsto, "1"] & \mathcal{S}_1 \arrow[dl, dashed, mapsto, "*"] \\
\mathcal{S}'_k \arrow[dr, dashed, mapsto, "\varepsilon"'] & \text{Lemma 1} & \\
& \mathcal{S}'_C \simeq \mathcal{S}_C &
\end{tikzcd}$$

(b)

$$\begin{tikzcd}
& \mathcal{S} \arrow[dl, mapsto, "k"'] \arrow[dr, mapsto, "n-1"] & & \\
\mathcal{S}'_k \arrow[dr, dashed, mapsto, "*"'] & \text{Ind. Hyp.} & \mathcal{S}_{n-1} \arrow[dl, dashed, mapsto, "l"] \arrow[dr, mapsto, "1"] & \\
& \mathcal{S}'_{C'} \simeq \mathcal{S}_{C'} \arrow[dr, dashed, mapsto, "\varepsilon"'] & \text{Lemma 1} & \mathcal{S}_n \arrow[dl, dashed, mapsto, "*"] \\
& & \mathcal{S}'_C \simeq \mathcal{S}_C &
\end{tikzcd}$$

(c)

Fig. 4. Proof diagrams of Theorem 3

$c_2, \top\rangle, \langle ph_1(think) \wedge c_1 \wedge ph_3(eat), \top\rangle)$ by applying rules p_1 and p_3 respectively. Strong confluence follows directly from $\mathcal{S}_1 \overset{p_1'}{\mapsto} \mathcal{S}$ and $\mathcal{S}_2 \overset{p_3'}{\mapsto} \mathcal{S}$. Note that this already proves both symmetric cases needed for strong confluence of the critical pair. The other two similar critical ancestor states can be shown to be strongly confluent analogously. Also for the unreachable states, which include multiple $ph_i(i = 1, 2, 3)$ or $c_i(i = 1, 2, 3)$ constraints, strong confluence is given by applying the appropriate r_i' rules, and thus it follows from Corollary 1 that the dining philosophers program is confluent.

4 Conclusion

In this work we revisited existing results about confluence for terminating CHR programs and investigated confluence for non-terminating programs. Applying the Strong Church-Rosser property to Constraint Handling Rules we are able to give a criteria for confluence of non-terminating CHR programs.

As confluence in general is undecidable it is important to note that only the subset of strongly closed CHR programs can be shown to be confluent using our theorem. However, due to possible non-termination an automated test for strong closedness is not possible, such that the SCR property has to be investigated manually as shown in the previous example.

Research on confluence in CHR is an ongoing endeavor. A current result by Duck, et.al. [14] on observable confluence suggests that a stricter definition of critical pairs could be found, as the considered set of critical pairs is often based on unreachable ancestor states. As an example of this result consider the critical ancestor states of the dining philosophers Example 4 again.

Confluence is also considered by the CCP community. For example [15] defines simple denotational semantics for subsets of CCP programs adding restrictions on the notion of choice or requiring confluence. Furthermore, confluence is actively being investigated by the term rewriting community [2, 3].

Due to the many available results for term rewriting systems it may prove valuable to investigate their application to CHR, as we did here for the strong Church-Rosser property. In order to ease this process a formal embedding of term rewriting systems in CHR could be of help. Furthermore the results given in this paper could be extended to general CHR programs, including propagation and simpagation rules. It may also be worthwhile to explore syntactical criteria for strong closedness, as this would allow for an efficient automated confluence test detecting confluence of a subset of the strongly closed non-terminating CHR programs.

Acknowledgments

The authors wish to express their gratitude to Thom Frühwirth and Maurizio Gabbrielli for the initial suggestion, helpful discussions, and supervision. We are grateful for comments from the anonymous referees, and we want to thank Hariolf Betz for his tireless endeavors to improve the quality of the paper's parlance.

References

1. Huet, G.: Confluent reductions: Abstract properties and applications to term rewriting systems: Abstract properties and applications to term rewriting systems. Journal of the ACM **27**(4) (1980) 797–821
2. Gramlich, B., Lucas, S.: Generalizing Newman's Lemma for left-linear rewrite systems. In Pfenning, F., ed.: Proc. 17th Int. Conf. on Rewriting Techniques and Applications (RTA'06), Seattle, Washington, USA, August 12-14, 2006. Volume 4098 of Lecture Notes in Computer Science., Springer-Verlag (2006) 66–80 ISBN: 3-540-36834-5; DOI: 10.1007/11805618.
3. Gramlich, B.: Confluence without termination via parallel critical pairs. In: Colloquium on Trees in Algebra and Programming. (1996) 211–225
4. Abdennadher, S., Frühwirth, T., Meuss, H.: Confluence and semantics of constraint simplification rules. Constraints **4**(2) (1999) 133–165
5. Abdennadher, S.: Operational semantics and confluence of constraint propagation rules. In: Principles and Practice of Constraint Programming. (1997) 252–266
6. Frühwirth, T.: Constraint Handling Rules - the story so far. In: Principles and Practice of Declarative Programming 2006 (PPDP'06), Venice - Italy (July 2006) Invited Tutorial.
7. Frühwirth, T.: Parallelizing union-find in constraint handling rules using confluence analysis. In: International Conference on Logic Programming. (2005)
8. Meister, M.: Fine-grained parallel implementation of the preflow-push algorithm in CHR. In: WLP. (2006) 172–181
9. Frühwirth, T.: Theory and practice of Constraint Handling Rules. Journal of Logic Programming, Special Issue on Constraint Logic Programming **37**(1-3) (October 1998) 95–138
10. Frühwirth, T., Abdennadher, S.: Essentials of Constraint Programming. Springer-Verlag (2003)
11. Lloyd, J.W.: Foundations of Logic Programming. Springer - Verlag (1987) Second, Extended Edition.
12. Newman, M.H.A.: On theories with a combinatorial definition of "equivalence". Annals of Mathematics **43**(2) (1942) 223–243
13. Meuss, H.: Konfluenz von Constraint Handling Rules-Programmen. Diplomarbeit/diploma thesis, Institute of Computer Science, LMU, Munich (1996)
14. Duck, G.J., Stuckey, P.J., Sulzmann, M.: Observable confluence for constraint handling rules. In: International Conference on Logic Programming (to appear). (2007)
15. Falaschi, M., Gabbrielli, M., Mariott, K., Palamidessi, C.: Confluence in concurrent constraint programming. Theoretical Computer Science **183**(2) (1997) 281–315