Confluence Modulo Equivalence with Invariants in Constraint Handling Rules

Daniel Gall and Thom Frühwirth

Institute of Software Engineering and Programming Languages, Ulm University,
89069 Ulm, Germany
daniel.gall@uni-ulm.de, thom.fruehwirth@uni-ulm.de

Abstract. Confluence denotes the property of a state transition system that states can be rewritten in more than one way yielding the same result. Although it is a desirable property, confluence is often too strict in practical applications because it also considers states that can never be reached in practice. Additionally, sometimes states that have the same semantics in the practical context are considered as different states due to different syntactic representations. By introducing suitable invariants and equivalence relations on the states, programs may have the property to be confluent modulo the equivalence relation w.r.t. the invariant which often is desirable in practice.
In this paper, a sufficient and necessary criterion for confluence modulo equivalence w.r.t. an invariant for Constraint Handling Rules (CHR) is presented. It is the first approach that covers invariant-based confluence modulo equivalence for the de facto standard semantics of CHR. There is a trade-off between practical applicability and the simplicity of proving a confluence property. Therefore, a better manageable subset of equivalence relations has been identified that allows for the proposed confluence criterion and and simplifies the confluence proofs by using well established CHR analysis methods.

1 Introduction

In program analysis, the *confluence* property of a program plays an important role. It ensures that any computation for a given start state results in the same final state. Hence, if more than one rule is applicable in a state it does not matter which rule is chosen.

Constraint Handling Rules (CHR) is a declarative programming language that has its origins in constraint logic programming [1]. Confluence analysis has been studied for CHR for a long time [2,3,4].

While it is a desirable property, in practical applications confluence is often too strict. For instance, it requires even states that can never be reached in a practical context to satisfy the confluence property. Therefore, *invariant-based confluence* [5,6,7] has been established. It only considers states that satisfy a user-defined invariant, whereas standard confluence analysis considers even states that are invalid and cannot appear at runtime. With invariant-based confluence

analysis it is possible to exclude those states from the confluence analysis as long as the rules of the program maintain the invariant.

Another method of making the confluence property available for more practical programs is to define an equivalence relation on states. A program is *confluent modulo a (user-defined) equivalence relation* if all states in the same equivalence class lead to final states of the same equivalence class [8,9]. In many programs, some states can be considered as equivalent with respect to a user-defined equivalence relation, although their actual representation in the program differs. For example, if sets of numbers are represented as lists, all states with permutations of the same list represent the same set and it might be reasonable to consider them equivalent. Hereby, confluence modulo equivalence can be used to show that for the same start state a program yields the same set as a result, although the actual representation as a list might differ.

There is a trade-off between the applicability in practical contexts and the simplicity of proving a confluence property: There is a decidable, sufficient and necessary criterion for strict confluence of terminating CHR programs [1]. When adding invariants, decidability of the criterion is lost depending on the invariant. For confluence modulo equivalence, the proofs become even harder as all states in the same equivalence class have to be considered.

In this paper, a sufficient and necessary criterion for invariant-based confluence modulo a user-defined equivalence is presented. For this purpose, a class of well-behaving equivalence relations is identified for which the proposed criterion can be applied. The confluence criterion is directly available for the equivalence-based operational semantics of CHR [10,7] that is the de facto standard of CHR semantics. By a running example it is shown that the defined class of equivalence relations is meaningful in a sense that it contains a non-trivial equivalence relation that satisfies its restrictions. Further examples have been tried indicating that the approach is promising to be more widely applicable.

In our approach, we use CHR in the pure form. We then restrict the equivalence relations to a meaningful class and present a formal proof method for invariant-based confluence modulo equivalence.

The contributions of the paper are

- the identification of a class of equivalence relations (called *compatible* equivalence relations) that maintains the monotonicity property of CHR and therefore allows for a confluence analysis based on rule states and overlaps of rules (c.f. Section 3),
- a sufficient and necessary criterion for an invariant-based confluence modulo equivalence for terminating CHR programs with a decidable invariant and a compatible equivalence relation (c.f. Section 4), and
- the application of this approach in a non-trivial running example.

Our approach is the first that covers invariant-based confluence modulo equivalence for the standard semantics of CHR. Other approaches either only consider invariants without user-defined equivalence relations [5,6,7] or use a special-purpose operational semantics of CHR that is claimed to extend the standard semantics [9,8]. The latter approach introduces a meta-level to prove

confluence modulo equivalence. In contrast to the meta-level proof method, the confluence criterion in this paper uses well-established standard notions of CHR states and analysis methods.

The paper is structured as follows: In Section 2 the preliminaries necessary for understanding the paper are given. For this purpose, definitions of confluence modulo equivalence, Constraint Handling Rules and some program analysis methods for CHR are recapitulated. Then, the class of equivalence relations regarded in this paper is defined in Section 3. The proof method for invariant-based confluence modulo equivalence is given in Section 4. The results and their relation to existing work are discussed in Section 5.

2 Preliminaries

We recapitulate the basic notions of confluence modulo equivalence, give a brief introduction to CHR and some program analysis techniques and summarize the established results for (invariant-based) confluence in CHR.

2.1 Confluence Modulo Equivalence

The notion of confluence modulo equivalence is defined for general state transition systems in this section.

Definition 1 (state transition system). *A* state transition system *is a tuple* $(\Sigma, \mapsto)$ *where* Σ *is an arbitrary (possibly infinitely large) set of* states *and* $\mapsto \subseteq \Sigma \times \Sigma$ *is a* transition relation *over the states. By* $\mapsto^*$ *we denote the reflexive transitive closure of* $\mapsto$*.*

Informally, confluence modulo equivalence means that all possible computations in a transition system starting in equivalent states finally lead to equivalent states again. We then call two states from those different computations joinable. This is illustrated in Figure 1.

$$\begin{array}{ccc} \sigma_1 & \overset{*}{\longmapsto} & \sigma_2 & \overset{*}{\longmapsto} & \tau \\ \approx & & & & \approx \\ \sigma_1' & \underset{*}{\longmapsto} & \sigma_2' & \underset{*}{\longmapsto} & \tau' \end{array}$$

Fig. 1. Confluence modulo equivalence

Definition 2 (joinability modulo equivalence). *In a state transition system* $(\Sigma, \mapsto)$ *two states* $\sigma, \sigma' \in \Sigma$ *are* joinable modulo an equivalence relation $\approx$ *if and only if* $\exists \tau, \tau' \in \Sigma \ . \ \sigma \mapsto^* \tau \wedge \sigma' \mapsto^* \tau' \wedge \tau \approx \tau'$*. We then write* $\sigma \downarrow^{\approx} \sigma'$*. If* $\approx$ *is the identity equivalence relation* $=$*, we write* $\sigma \downarrow \sigma'$ *and say that* σ *and* σ' *are* joinable*.*

Definition 3 (confluence modulo equivalence [11]). *A state transition system* $(\Sigma, \mapsto)$ *is confluent modulo an equivalence relation* $\approx$*, if and only if for all* $\sigma_1, \sigma_1', \sigma_2, \sigma_2' : (\sigma_1 \approx \sigma_1') \wedge (\sigma_1 \mapsto^* \sigma_2) \wedge (\sigma_1' \mapsto^* \sigma_2') \rightarrow (\sigma_1' \downarrow^{\approx} \sigma_2')$.

If $\approx$ is the state equivalence relation $=$, confluence modulo $=$ coincides with basic confluence. For terminating transition systems, it suffices to show *local confluence*, as we will see in the following definition and theorem.

Definition 4 (local confluence [11]). *A state transition system* $(\Sigma, \mapsto)$ *has the* α *and* β *property w.r.t. an equivalence relation* $\approx$ *if and only if it satisfies the* α *and* β *conditions, respectively:*

α**:** $\forall \sigma, \tau, \tau' \in \Sigma : \sigma \mapsto \tau \wedge \sigma \mapsto \tau' \rightarrow \tau \downarrow^{\approx} \tau'$.
β**:** $\forall \sigma, \tau, \tau' \in \Sigma : \sigma \mapsto \tau \wedge \sigma \approx \tau' \rightarrow \tau \downarrow^{\approx} \tau'$.

A state transition system is locally confluent modulo an equivalence relation $\approx$ *if and only if it has the* α *and the* β *property.*

Note that in the rewriting literature, the α property is also known as *local confluence modulo equivalence* and the β property as *local coherence modulo equivalence* [12]. In this paper, *local confluence modulo equivalence* requires both the α and the β property.

In the theorem of Huet [11] it is shown that local confluence modulo an equivalence relation $\approx$ implies confluence modulo $\approx$ for terminating transition systems.

Theorem 1 (Huet [11]). *Let* $(\Sigma, \mapsto)$ *be a terminating transition system. For any equivalence* $\approx$*,* $(\Sigma, \mapsto)$ *is confluent modulo* $\approx$ *if and only if* $(\Sigma, \mapsto)$ *is locally confluent modulo* $\approx$*.*

2.2 Constraint Handling Rules

We now define the state transition system of CHR. We begin with CHR states.

Definition 5 (CHR state). *A* CHR state *is a tuple* $\langle \mathbb{G}; \mathbb{C}; \mathbb{V} \rangle$ *where the* goal $\mathbb{G}$ *is a multi-set of constraints, the* built-in constraint store $\mathbb{C}$ *is a conjunction of built-in constraints and* $\mathbb{V}$ *is a set of* global variables*. All variables occurring in a state that are not global are called* local variables *[7, p. 33 et seq., def. 8.1]. If the contents of* $\mathbb{C}$ *and* $\mathbb{V}$ *are empty, irrelevant or clear from the context, we use a short-hand notation where only the constraints in* $\mathbb{G}$ *are enumerated.*

CHR states can be modified by rules that together form a CHR program.

Definition 6 (CHR program). *A* CHR program *is a finite set of so-called simpagation rules of the form* $r : H_{\mathrm{k}} \setminus H_{\mathrm{r}} \Leftrightarrow G \mid B_{\mathrm{c}}, B_{\mathrm{b}}$ *where* r *is an optional rule name, the heads* H_{k} *and* H_{r} *are multi-sets of CHR constraints, the guard* G *is a conjunction of built-in constraints and the body is a multi-set of CHR constraints* B_{c} *and a conjunction of built-in constraints* B_{b}*. If* G *is empty, it is interpreted as the built-in constraint* $\top$*.*

We introduce short forms for the following special cases :

Simplification Rules *If* $H_\mathrm{k} = \emptyset$*, we also write* $H_\mathrm{r} \Leftrightarrow G \mid B_\mathrm{c}, B_\mathrm{b}$.
Propagation Rules *If* $H_\mathrm{r} = \emptyset$*, we also write* $H_\mathrm{k} \Rightarrow G \mid B_\mathrm{c}, B_\mathrm{b}$.

Informally, a rule is applicable, if the heads match constraints from the goal store $\mathbb{G}$ and the guard holds, i.e. is a consequence of the built-in constraints $\mathbb{C}$. In that case, the state is rewritten: The constraints matching the part H_r of the head are removed and the constraints matching H_k are kept. The user-defined body constraints B_c are added to the goal store $\mathbb{G}$, the built-in body constraints B_b and the constraints from the guard G are added to the built-in store $\mathbb{C}$.

Example 1 (Multi-Set Items [9]). Consider the following small CHR program, that collects items represented in individual $item/1$ constraints to a multi-set represented by a constraint of the form $mset(L)$ where L is a list of items. The program has the following rule:

$$mset(L), item(A) \Leftrightarrow mset([A|L]).$$

For the initial constraint store $item(a), item(b), mset([])$ the program can apply the rule on $mset([])$ and $item(b)$ which results in the constraint store $mset([b]), item(a)$. The rule can be applied again to this state, resulting in the constraint store $mset([a, b])$. However, the same program can also yield the constraint store $mset([b, a])$. Hence, the program is not confluent. In the following, we will return to this running example and provide an invariant and equivalence relation together with a proof method to show that the program is actually confluent modulo the equivalence relation w.r.t. the invariant.

In the context of the operational semantics, we assume a constraint theory $\mathcal{CT}$ for the interpretation of the built-in constraints. We define an equivalence relation over CHR states.

Definition 7 (state equivalence [7,10]). *Let* $\rho_i := \langle \mathbb{G}_i; \mathbb{C}_i; \mathbb{V}_i \rangle$ *for* $i = 1, 2$ *be two CHR states with local variables* $\bar{y_1}, \bar{y_2}$ *that have been renamed apart.* $\rho_1 \equiv \rho_2$ *if and only if* $\mathcal{CT} \models \forall(\mathbb{C}_1 \rightarrow \exists \bar{y_2}.((\mathbb{G}_1 = \mathbb{G}_2) \wedge \mathbb{C}_2)) \wedge \forall(\mathbb{C}_2 \rightarrow \exists \bar{y_1}.((\mathbb{G}_1 = \mathbb{G}_2) \wedge \mathbb{C}_1))$ *where* $\forall F$ *is the universal closure of formula* F *and* $=$ *is syntactical equivalence. The equivalence class of a CHR state is defined as* $[\rho] := \{\rho' \mid \rho' \equiv \rho\}$.

Example 2 (state equivalence). By the above definition of state equivalence, the following states are equivalent [7, p. 34]:

- $\langle c(X); \top; \emptyset \rangle \equiv \langle c(Y); \top; \emptyset \rangle$, i.e. local variables can be renamed.
- $\langle c(X); X{=}0; \{X\} \rangle \equiv \langle c(0); X{=}0; \{X\} \rangle$, i.e. variable bindings from the built-in store can be applied to the goal store.
- $\langle \emptyset; X{=}Y \wedge Y{=}0; \emptyset \rangle \equiv \langle \emptyset; X{=}0 \wedge Y{=}0; \emptyset \rangle$, i.e. equivalent built-in stores can be interchanged.
- $\langle c(0); \top; \{X\} \rangle \equiv \langle c(0); \top; \emptyset \rangle$, i.e. unused global variables can be omitted.
- However, $\langle c(X); \top; \{X\} \rangle \not\equiv \langle c(Y); \top; \{Y\} \rangle$, i.e. X and Y are free variables and therefore the logical readings of the states are different. Global variables can be used to bridge information between two states.

The operational semantics is now defined by the following transition scheme over equivalence classes of CHR states.

Definition 8 (operational semantics of CHR [7,10]). *For a rule r, the variables appearing in the rule are called local variables. A variant of a rule is a copy of a rule where a subset of its local variables has been renamed.*

For a CHR program, the state transition system over CHR states and the rule transition relation $\mapsto$ is defined as the following transition scheme:

$$\frac{r : H_\mathrm{k} \setminus H_\mathrm{r} \Leftrightarrow G \mid B_\mathrm{c}, B_\mathrm{b}}{[\langle H_\mathrm{k} \uplus H_\mathrm{r} \uplus \mathbb{G}; G \wedge \mathbb{C}; \mathbb{V}\rangle] \mapsto^r [\langle H_\mathrm{k} \uplus B_\mathrm{c} \uplus \mathbb{G}; G \wedge B_\mathrm{b} \wedge \mathbb{C}; \mathbb{V}\rangle]}$$

Thereby, r is a variant of a rule in the program such that its local variables are disjoint from the variables occurring in the representative of the pre-transition state. We may just write $\mapsto$ instead of $\mapsto^r$ if the rule r is clear from the context.

From now on, we only consider equivalence classes of CHR states, since the state transition system is defined over equivalence classes.

Example 3. In this example, the program from Example 1 is executed using the operational semantics of Definition 8.

$$\begin{aligned} & \langle \mathit{mset}([a]), \mathit{item}(b); \top; \emptyset\rangle \\ = \ & \langle \mathit{mset}(L), \mathit{item}(A); L = [a] \wedge A = b; \emptyset\rangle \\ \mapsto \ & \langle \mathit{mset}([A|L]); L = [a] \wedge A = b; \emptyset\rangle \\ = \ & \langle \mathit{mset}([a,b]); \top; \emptyset\rangle \end{aligned}$$

In the first step, an equivalent state with fresh local variables L and A is constructed. The constraints in the goal store of this state are syntactically equivalent to the head of the variant of the rule with variables L and A. The guard of the rule is $\top$ and therefore, the rule is applicable. After applying the rule, the state can be transformed into a more readable form without local variables.

An important analysis technique is the merging of states.

Definition 9 (merge operator $\diamond$). *Let $\sigma_i = \langle \mathbb{G}_i; \mathbb{B}_i; \mathbb{V}_i\rangle$ for $i = 1, 2$ be two CHR states such that local variables of one state are disjoint from all variables in the other state. Then for a set $\mathbb{V}$ of variables*

$$\sigma_1 \diamond_\mathbb{V} \sigma_2 := \langle \mathbb{G}_1 \uplus \mathbb{G}_2; \mathbb{B}_1 \wedge \mathbb{B}_2; (\mathbb{V}_1 \cup \mathbb{V}_2) \setminus \mathbb{V}\rangle.$$

For equivalence classes of CHR states, the merging is defined as $[\sigma_1] \diamond_\mathbb{V} [\sigma_2] := [\sigma_1 \diamond_\mathbb{V} \sigma_2]$ for two representatives of the equivalence class that have disjoint variables. For $\mathbb{V} = \emptyset$ we write $[\sigma_1] \diamond [\sigma_2]$ [7, p. 50, def. 10.1].

Since local variables have to be disjoint when merging two states, it is not possible to extract information about them directly. For instance, $[\langle c(X); X{=}1; \emptyset\rangle]$ is the version of $[\langle c(X); \top; \emptyset\rangle]$ with the local variable X, where X is bound to the number 1. In the state $[\langle c(X), X{=}1, \emptyset\rangle]$, we would consider $X{=}1$ as

contextual information about the local variable X. It is not possible to extract this information by $[\langle c(X); \top; \emptyset\rangle] \diamond [\langle \emptyset; X{=}1; \emptyset\rangle]$, since $[\langle \emptyset; X{=}1; \emptyset\rangle] = [\langle \emptyset; \top; \emptyset\rangle] = [\sigma_\emptyset]$, i.e. the empty state. Hence, the result of merging the two states is $[\langle c(X); \top; \emptyset\rangle]$ although we would like to see the result $[\langle c(X); X{=}1; \emptyset\rangle]$.

It is necessary to rather make X a global variable first that is reduced by the merge operator $\diamond_{\{X\}}$:

$$[\langle c(X); \top; \{X\}\rangle] \diamond_{\{X\}} [\langle \emptyset; X{=}1; \{X\}\rangle] = [\langle c(X); X{=}1; \emptyset\rangle] = [\langle c(1); \top; \emptyset\rangle].$$

Global variables can thus be used to share information between two states that are merged. [7, p. 50, ex. 10.2] In the confluence criterion in Section 4, we only generate states from the program source code where all variables are global.

In general, $\diamond_{\mathbb{V}}$ is not associative. However, the following lemma shows a restricted form of associativity that is used in the proof of the confluence modulo equivalence criterion in Section 4.

Lemma 1. *Let $\sigma_1, \sigma_2, \sigma_3$ be CHR states such that no local variable of a state occurs in another state. Then $[\sigma_1] \diamond_{\mathbb{V}} ([\sigma_2] \diamond [\sigma_3]) = ([\sigma_1] \diamond [\sigma_2]) \diamond_{\mathbb{V}} [\sigma_3]$ holds for all $\mathbb{V}$ [7, p. 52, lemma 10.7].*

2.3 Confluence of CHR Programs

The idea of the confluence criterion is to exploit the *monotonicity* property of CHR, i.e. that all rules applicable in one state are applicable in any larger state.

Lemma 2 (monotonicity). *If $[\sigma] \mapsto [\tau]$, then $[\sigma] \diamond_{\mathbb{V}} [\sigma'] \mapsto [\tau] \diamond_{\mathbb{V}} [\sigma']$ for all $\mathbb{V}$ and $[\sigma]$ [7, p. 51, lemma 10.4].*

Basic Confluence Test Monotonicity allows us to reason from states about larger states. The idea of the basic confluence test is to construct a finite set of *rule states* that consist of the head and guard constraints of a rule and then overlap them with all other rule states. Intuitively, overlapping two rules means that a state is constructed where parts of the rule heads are equated (if possible) and the rest is just included. In such a state, both rules are applicable.

By applying the overlapping rules to the overlap state, we get a *critical pair*. Thereby, one state is the result after applying the first overlapping rule to the overlap state and the other state is the result after applying the second rule to the overlap state. If all critical pairs are joinable, the program is locally confluent. In the following, we formalize this idea. The definitions are taken from [7]. Similar definitions can be found in [1].

Definition 10 (rule state). *For a rule $r : H_\mathrm{k} \setminus H_\mathrm{r} \Leftrightarrow G \mid B_c, B_b$ let $\mathbb{V}$ be the variables occurring in $H_\mathrm{k}, H_\mathrm{r}$ and G. Then the state $\langle H_\mathrm{k} \uplus H_\mathrm{r}; G; \mathbb{V}\rangle$ is called the* rule state *of r. In the literature, the rule states are sometimes called* minimal states. *[7, p. 78, def. 13.8]*

Definition 11 (overlap). *For any two (not necessarily different) rules of a CHR program of the form* $r_1 : H_\mathrm{k} \setminus H_\mathrm{r} \Leftrightarrow G \mid B_\mathrm{c}, B_\mathrm{b}$,$r_2 : H'_\mathrm{k} \setminus H'_\mathrm{r} \Leftrightarrow G' \mid B'_\mathrm{c}, B'_\mathrm{b}$ *and with variables that are renamed apart, let* $O_\mathrm{k} \subseteq H_\mathrm{k}, O_\mathrm{r} \subseteq H_\mathrm{r}$,$O'_\mathrm{k} \subseteq H'_\mathrm{k}, O'_\mathrm{r} \subseteq H'_\mathrm{r}$ *be subsets of the heads of the rules such that for* $B := ((O_\mathrm{k} \uplus O_\mathrm{r}) = (O'_\mathrm{k} \uplus O'_\mathrm{r})) \wedge G \wedge G'$ *it holds that* $\mathcal{CT} \models \exists.B$ *and* $(O_\mathrm{r} \uplus O'_\mathrm{r}) \neq \emptyset$*, where* $\exists.B$ *is the existential closure over* B*. Then the state* $\sigma := \langle K \uplus K' \uplus R \uplus R' \uplus O_\mathrm{k} \uplus O_\mathrm{r}; B; \mathbb{V} \rangle$ *is called an* overlap *of* r_1 *and* r_2 *where* $\mathbb{V}$ *is the set of all variables occurring in heads and guards of both rules and* $K := H_\mathrm{k} \setminus O_\mathrm{k}$*,* $K' := H'_\mathrm{k} \setminus O'_\mathrm{k}$*,* $R := H_\mathrm{r} \setminus O_\mathrm{r}$*,* $R' := H'_\mathrm{r} \setminus O'_\mathrm{r}$*. The pair of states* (σ_1, σ_2) *with* $\sigma_1 := \langle K \uplus K' \uplus R' \uplus O_\mathrm{k} \uplus B_\mathrm{c}; B \wedge B_\mathrm{b}; \mathbb{V} \rangle$ *and* $\sigma_2 := \langle K \uplus K' \uplus R \uplus O'_\mathrm{k} \uplus B'_\mathrm{c}; B \wedge B'_\mathrm{b}; \mathbb{V} \rangle$ *is called* critical pair *of the overlap* σ*. The critical pair can be obtained by applying the rules to the overlap state. [7, p. 82, def. 14.5]*

Invariant-Based Confluence Test The idea of exploiting monotonicity fails, when invariants on the states are introduced. A property $\mathcal{I}$ is an invariant if and only if for all states $[\sigma]$ where $\mathcal{I}([\sigma])$ holds and for all $[\tau]$ with $[\sigma] \mapsto^* [\tau]$ the invariant $\mathcal{I}([\tau])$ holds as well.

If in the confluence test a constructed overlap does not satisfy the invariant, then this overlap state is not part of the transition system and therefore no information can be gained from analyzing it. It is not possible to just ignore such states as there are invariants that are not satisfied in an overlap state, but might be satisfied in a larger state. There are also invariants that are invalidated in an overlap state and that cannot be satisfied by state extension (c.f. Example 4). For instance, if only a constraint is only allowed to appear at most once in a state, this invariant cannot be satisfied by extending the state invalidating it.

Nevertheless, the idea of using overlap states for confluence analysis can be generalized, such that it can be used for invariant-based confluence. For this purpose, for an invariant $\mathcal{I}$ and an overlap state $[\sigma]$ the set of all extensions of $[\sigma]$ such that $\mathcal{I}$ holds – denoted by $\Sigma^{\mathcal{I}}([\sigma])$ – is considered. As this set usually is infinitely large, we want to extract a set of minimal elements of $\Sigma^{\mathcal{I}}([\sigma])$, called $\mathcal{M}^{\mathcal{I}}([\sigma])$, that have to be considered to show local confluence w.r.t. $\mathcal{I}$. However, for this purpose a partial order on states has to be defined. The set $\mathcal{M}^{\mathcal{I}}([\sigma])$ is finite for many invariants, but there are examples of invariants that lead to infinite sets of minimal elements.

In [5,6,7] the following has been proven: If we can show that for all overlap states $[\sigma]$ of a terminating program the critical pairs derived from all states in $\mathcal{M}^{\mathcal{I}}([\sigma])$ are joinable, the program is confluent w.r.t. to $\mathcal{I}$.

We now give formal definitions of the notions used in the above description. Since it is a commutative monoid, a partial order can be derived from the merge operator [7]:

Lemma 3 (partial order $\lhd$). *For the set of CHR states* Σ*, the relation* $\lhd$*:* $\Sigma \times \Sigma$ *defined as* $[\sigma] \lhd [\sigma']$ *if and only if* $\exists [\hat{\sigma}] \,.\, [\sigma] \diamond [\hat{\sigma}] = [\sigma']$ *where* $\sigma, \sigma' \in \Sigma$ *is a partial order. [7, p. 53, lemma 10.8]*

In [5,6], another partial order has been defined. However, it has been shown that the relation defined there is not a partial order by mistake [7]. Therefore, we use

the partial order that has first been introduced in [7, p. 53, lemma 10.8] to avoid these problems.

For an invariant, we can now define the set of minimal elements that extend a state such that the invariant does hold.

Definition 12 (minimal elements). *For an invariant $\mathcal{I}$, let the set $\Sigma^{\mathcal{I}}([\sigma]) := \{[\sigma'] \mid \mathcal{I}([\sigma \diamond \sigma']) \wedge \sigma'$ has no local variables $\}$. The set $\mathcal{M}^{\mathcal{I}}([\sigma])$ is the set of $\lhd$-minimal elements of $\Sigma^{\mathcal{I}}([\sigma])$ such that $\forall [\sigma'] \in \Sigma^{\mathcal{I}}([\sigma]) \,.\, \exists [\sigma_{\mathrm{m}}] \in \mathcal{M}^{\mathcal{I}}([\sigma]) \,.\, [\sigma_{\mathrm{m}}] \lhd [\sigma']$ [7, p. 80, def. 13.11]. By* well-definedness *of $\mathcal{M}^{\mathcal{I}}([\sigma])$ we denote that it has the latter property.*

The set $\Sigma^{\mathcal{I}}$ may become infinitely large for states with local variables. Hence in program analysis, w.l.o.g. the states are restricted to only global variables. Monotonicity of CHR (Lemma 2) ensures that all results remain applicable if any of these variables are made local [7, p. 80].

Note that for an invariant $\mathcal{I}$ and a state $[\sigma]$ where $\mathcal{I}([\sigma])$ holds, the set of minimal extensions is $\mathcal{M}^{\mathcal{I}}([\sigma]) = \{[\sigma_\emptyset]\}$, where $\sigma_\emptyset := \langle \emptyset; \top; \emptyset \rangle$ is the empty state [7, p.80, lemma 13.13]. The invariant-based confluence test then coincides with the basic confluence criterion. In Section 4 we generalize the idea of the invariant-based confluence test for invariant-based confluence modulo equivalence.

Example 4 (Multi-Set Items (cont.)). For the multi-set program from Example 1, the following problem arises: If there is more than one *mset* constraint, the program can choose non-deterministically where to add an item. Therefore, it cannot be confluent. For instance, the following transitions in shorthand notation is possible: $\mathit{mset}([a]), \mathit{mset}([b]), \mathit{item}([c])$ can either end in the final state $\mathit{mset}([a, c]), \mathit{mset}([b])$ or $\mathit{mset}([a]), \mathit{mset}([b, c])$.

The problem can be solved by introducing the multi-set invariant $\mathcal{S}$: In every CHR state there is at most one $\mathit{mset}(\_)$ constraint. Note that the set of minimal extensions $\mathcal{M}^{\mathcal{S}}([\sigma]) = \emptyset$ for all states $[\sigma]$, as there are no extensions for states that do not satisfy the invariant (i.e. where there is more than one *mset* constraint) such that the invariant is satisfied (i.e. there is at most one *mset* constraint).

3 Compatibility of Equivalence Relations

In this section, we motivate a restriction of equivalence relations that make confluence modulo equivalence analysis manageable. Note that in the context of confluence modulo equivalence, typically user-defined equivalence relations different from state equivalence (c.f. Definition 7) are regarded. State equivalence is referred to by $\equiv$ or by the corresponding equivalence class brackets $[\cdot]$. The symbol $\approx$ denotes some general user-defined equivalence relation that is potentially different from $\equiv$ (but is not required to be).

In the confluence criterion, we want to use the idea of exploiting monotonicity of CHR to reason from small states that come from the rules in the program over all states. However, monotonicity can be broken by user-defined equivalence relations. This means that in general for two states with $[\sigma] \approx [\sigma']$, it is possible

that an extension with $[\tau] \approx [\tau']$ leads to states that are not equivalent, i.e. $[\sigma] \diamond_{\mathbb{V}} [\tau] \not\approx [\sigma'] \diamond_{\mathbb{V}} [\tau']$ as shown in the following example.

Example 5. We construct an equivalence relation that breaks monotonicity. Let $\#c : \Sigma \to \mathbb{N}_0$ be a function that returns the number of constraints c in the goal store of a state. We separate the CHR state space Σ into two disjoint subsets:

$$\Sigma_1 := \{[\sigma] \mid \#c([\sigma]) < 3\}, \qquad \Sigma_2 := \{[\sigma] \mid \#c([\sigma]) \geq 3\}.$$

The partition of the state space clearly defines an equivalence relation $\approx$ with equivalence classes Σ_1 and Σ_2.

Let $[\sigma_1] = [\langle c; \top; \emptyset \rangle]$ and $[\sigma_2] = [\langle c, c; \top; \emptyset \rangle]$. Since $[\sigma_1], [\sigma_2] \in \Sigma_1$, it holds that $[\sigma_1] \approx [\sigma_2]$. Let $[\tau] = [\langle c; \top; \emptyset \rangle]$. If we extend the two states by $[\tau]$, the extended states are not equivalent any more:

$$[\sigma_1] \diamond [\tau] = [\langle c, c; \top; \emptyset \rangle] \in \Sigma_1, \text{ but } [\sigma_2] \diamond [\tau] = [\langle c, c, c; \top; \emptyset \rangle] \in \Sigma_2.$$

Hence, although $[\sigma_1] \approx [\sigma_2]$, $[\sigma_1] \diamond [\tau] \not\approx [\sigma_2] \diamond [\tau]$.

This case does not harm testing for the β property, since the extended states do not have to be tested for joinability modulo equivalence according to the β property. However, we can construct the converse case: Let $[\sigma_3] = [\langle c, c, c; \top; \emptyset \rangle] \in \Sigma_2$. Then $[\sigma_2] \not\approx [\sigma_3]$. However, if the two states are extended by $[\tau]$, we get

$$[\sigma_2] \diamond [\tau] = [\langle c, c, c; \top; \emptyset \rangle] \in \Sigma_2, \text{ and } \qquad [\sigma_3] \diamond [\tau] = [\langle c, c, c, c; \top; \emptyset \rangle] \in \Sigma_2.$$

Hence, $[\sigma_2] \diamond [\tau] \approx [\sigma_3] \diamond [\tau]$, although $[\sigma_2] \not\approx [\sigma_3]$. This is critical to the β property: Now it is not possible any more to use a rule state and its equivalent states to reason about all states as we miss some larger state by this attempt.

To ensure monotonicity in the context of equivalence relations, we need equivalence to be maintained by the merge operator. The equivalence relation is then called a *congruence relation* with respect to the merge operator.

Definition 13 (congruence relation). *An equivalence relation $\approx \subseteq A \times A$ is called a* congruence relation *with respect to an operator $\circ : A \times A \to A$ if for all x, x', y, y': If $x \approx x'$ and $y \approx y'$ then $x \circ y \approx x' \circ y'$.*

Unfortunately, this does not suffice to reason from rule states about any other state. It must be ensured that if two states $[\sigma]$ and $[\sigma']$ are equivalent and $[\sigma]$ can be decomposed into two parts, then $[\sigma']$ must be decomposable into two parts that are equivalent to the decomposition of $[\sigma]$. This ensures that when showing joinability of two small states, the larger states can still be joined, as they are syntactically decomposable into smaller joinable states.

Definition 14 (split property). *An equivalence relation $\approx \subseteq A \times A$ has the* split property *with respect to an operator $\circ : A \times A \to A$ if for all x, x_1, x_2, y: If $x = x_1 \circ x_2$ and $x \approx y$ then $\exists y_1, y_2$ such that $x_1 \approx y_1, x_2 \approx y_2$ and $y = y_1 \circ y_2$.*

The split property assumes a syntactic relation between two states that are equivalent under an equivalence relation. If a state can be split into two parts and is equivalent to another state, this state can be split into equivalent parts.

Example 6. This example defines an equivalence relation $\hat{=}$ that does not satisfy the split property. It is the smallest equivalence relation where the following two conditions hold: If $\sigma \equiv \sigma'$ then also $\sigma \mathrel{\hat{=}} \sigma'$. Additionally, if $\langle \mathbb{G}; \mathbb{B}; \mathbb{V} \rangle \mathrel{\hat{=}} \langle \mathbb{G}'; \mathbb{B}'; \mathbb{V}' \rangle$, then $\langle \{c, c\} \uplus \mathbb{G}; \mathbb{B}; \mathbb{V} \rangle \mathrel{\hat{=}} \langle \{d\} \uplus \mathbb{G}'; \mathbb{B}'; \mathbb{V}' \rangle$. Hence, all pairs of c constraints can be replaced by a d constraint.

The equivalence relation obviously is a congruence relation w.r.t. $\diamond$. However, it does not have the split property: Let $\sigma \equiv c, c$ be a CHR state in shorthand notation. Then $\sigma \equiv c \diamond c$. By definition of $\hat{=}$, we have that $\sigma \mathrel{\hat{=}} d$. However, there are no σ_1, σ_2 such that $\sigma_1 \mathrel{\hat{=}} c$, $\sigma_2 \mathrel{\hat{=}} c$ and $d \equiv \sigma_1 \diamond \sigma_2$.

In the confluence test, for all states σ it has to be shown that if $\sigma \equiv \sigma'$ and $\sigma \mapsto_r \tau$ then $\sigma' \downarrow^{\approx} \tau$ to satisfy the β property. By the application of r to σ, we know that for the rule state σ_r, σ can be split into $[\sigma] = [\sigma_r] \diamond [\delta]$. To reason from joinability of σ_r and all its equivalent states, we also have to be able to split σ' into two parts $[\sigma'_r]$ and $[\delta']$. However, for the congruence relation $\hat{=}$ this is not possible as we have shown before. Hence, the idea of reasoning from rule states about all larger states cannot be applied.

Note that the split property is only required to hold for states where the invariant holds. Hence, by an appropriate invariant, the split property can be recovered to show confluence w.r.t. this invariant.

Definition 15 (compatibility). *An equivalence relation $\approx$ is* $\circ$-compatible *w.r.t. an operator $\circ$ if it is a congruence relation with the split property w.r.t. $\circ$.*

At first glance, compatibility is a strict property that does not seem to be satisfied by many equivalence relations. However, there are interesting $\diamond$-compatible equivalence relations different from the trivial state equivalence:

Example 7 (Multi-Set Items (cont.)). Example 1 is continued by introducing the following equivalence relation $\approx^{\mathrm{S}}$ that is the smallest equivalence relation on CHR states such that $[\langle \{mset(S_1)\} \uplus \mathbb{G}_1; \mathbb{B}_1; \mathbb{V}_1 \rangle] \approx^{\mathrm{S}} [\langle \{mset(S_2)\} \uplus \mathbb{G}_2; \mathbb{B}_2; \mathbb{V}_2 \rangle]$ if and only if S_1 is a permutation of S_2 and $[\langle \mathbb{G}_1; \mathbb{B}_1; \mathbb{V}_1 \rangle] \approx^{\mathrm{S}} [\langle \mathbb{G}_2; \mathbb{B}_2; \mathbb{V}_2 \rangle]$.

For instance, the following states in shorthand notation are equivalent according to $\approx^{\mathrm{S}}$: $mset([a,b]), mset([c,d]), item(e) \approx^{\mathrm{S}} mset([b,a]), mset([d,c]), item(e)$ and $item(a) \approx^{\mathrm{S}} item(a)$. However, $mset([a,b]), item(c) \not\approx^{\mathrm{S}} mset([a,b]), item(d)$ and $mset([a,b]), item(c) \not\approx^{\mathrm{S}} mset([a,b]), item(c), item(c)$ because the second item c does not have a partner in the first state. Similarly, $mset([a,b]), mset([b,a]) \not\approx^{\mathrm{S}} mset([a,b])$ because there is only one $mset$ constraint on the right hand side.

Note that by this definition the following holds for states with unbound variables: $[\langle mset(X); perm(X,Y); \{X,Y\} \rangle] \approx^{\mathrm{S}} [\langle mset(Y); perm(X,Y); \{X,Y\} \rangle]$ where $perm(X,Y)$ is a built-in constraint that is true, if X is a permutation of Y, but $[\langle mset(X); \top; \{X\} \rangle] \not\approx^{\mathrm{S}} [\langle mset(Y); \top; \{Y\} \rangle]$. The two variables X and Y are free variables and therefore it is not clear that they are permutations of each other. By adding that X is a permutation of Y, the two states are equivalent.

This equivalence relation is $\diamond$-compatible. For reasons of space, the proof is provided in Appendix A.1.

4 Confluence Modulo Equivalence w.r.t. an Invariant

First of all, the notion of invariant-based confluence modulo equivalence is defined.

Definition 16 ($\mathcal{I}$-confluence modulo $\approx$). *A state transition system is $\mathcal{I}$-confluent modulo an equivalence relation $\approx$ for an invariant $\mathcal{I}$ if and only if*

$$\forall \sigma_1, \sigma_2, \sigma_1', \sigma_2' \,.\, \mathcal{I}(\sigma_1) \wedge \mathcal{I}(\sigma_1') \wedge \sigma_1 \approx \sigma_1' \wedge \sigma_1 \mapsto^* \sigma_2 \wedge \sigma_1' \mapsto^* \sigma_2' \\ \rightarrow \exists \sigma_3, \sigma_3' \,.\, \sigma_2 \mapsto^* \sigma_3 \wedge \sigma_2' \mapsto^* \sigma_3' \wedge \sigma_3 \approx \sigma_3'.$$

In practice, the following restiction is made on invariants:

Definition 17 ($\approx$ maintains $\mathcal{I}$). *An invariant $\mathcal{I}$ is maintained by an equivalence relation $\approx$, if and only if for all states $[\sigma] \approx [\sigma']$ it holds that $\mathcal{I}([\sigma]) \leftrightarrow \mathcal{I}([\sigma'])$.*

This restriction ensures the practicability of Definition 16, since it may be inelegant and misleading if in a program that is $\mathcal{I}$-confluent modulo $\approx$ there exist two equivalent states where one is part of the program (i.e. the invariant holds) and the other is not. This may have undesired effects in further analysis. Hence, the invariant and equivalence relation should be chosen such that they are compliant anyway, although it is not required by Definition 16.

The following lemma is an important generalization of the joinability corollary in [7, p. 85, cor. 14.9] that is a direct consequence of monotonicity. The idea was that if two states are joinable, they are still joinable if they are extended by the identical state. In the context of confluence modulo equivalence, we have to generalize this approach of exploiting monotonicity such that the state extensions are not required to be syntactically identical, but equivalent for some user-defined compatible equivalence relation.

Lemma 4 (joinability). *Let $\approx$ be a congruence relation with respect to $\diamond$ and $[\sigma_1], [\sigma_2], [\sigma_1'], [\sigma_2']$ be CHR states with $[\sigma_1'] \approx [\sigma_2']$. If $[\sigma_1] \downarrow^{\approx} [\sigma_2]$ then $([\sigma_1] \diamond_{\mathbb{V}} [\sigma_1']) \downarrow^{\approx} ([\sigma_2] \diamond_{\mathbb{V}} [\sigma_2'])$ for all $\mathbb{V}$.*

Proof. Let $[\sigma_1], [\sigma_2], [\sigma_1'], [\sigma_2']$ be CHR states with $[\sigma_1'] \approx [\sigma_2']$ and $[\sigma_1] \downarrow^{\approx} [\sigma_2]$. Hence, there are CHR states $[\tau], [\tau']$ with $[\tau] \approx [\tau']$ and $[\sigma_1] \mapsto^* [\tau]$ and $[\sigma_2] \mapsto^* [\tau']$. Due to monotonicity (c.f. Lemma 2), we have that $([\sigma_1] \diamond_{\mathbb{V}} [\sigma_1']) \mapsto^* ([\tau] \diamond_{\mathbb{V}} [\sigma_1'])$ and $([\sigma_2] \diamond_{\mathbb{V}} [\sigma_2']) \mapsto^* ([\tau'] \diamond_{\mathbb{V}} [\sigma_2'])$. Since $[\sigma_1'] \approx [\sigma_2']$, $[\tau] \approx [\tau']$ and $\approx$ is a congruence relation with respect to $\diamond$, we have that $([\tau] \diamond_{\mathbb{V}} [\sigma_1']) \approx ([\tau'] \diamond_{\mathbb{V}} [\sigma_2'])$.

In the next step, we provide a test for the α property in the context of an invariant. The basic idea is that we gather all overlap states and extend them with a minimal extension such that the invariant does hold. For all those minimal extensions of all overlap states we have to show joinability modulo equivalence. Formally, this leads to the following lemma.

Lemma 5 (α property test). *Let $\mathcal{P}$ be a CHR program, $\mathcal{I}$ an invariant, $\approx$ a congruence relation and let $\mathcal{M}^{\mathcal{I}}([\sigma])$ be well-defined for all overlaps σ of rules in $\mathcal{P}$, then: $\mathcal{P}$ has the α property with respect to $\mathcal{I}$ and $\approx$ if and only if for all overlaps σ with critical pairs (σ_1, σ_2) and all $[\sigma_{\mathrm{m}}] \in \mathcal{M}^{\mathcal{I}}([\sigma])$ holds $([\sigma_1] \diamond [\sigma_{\mathrm{m}}] \downarrow^{\approx} [\sigma_2] \diamond [\sigma_{\mathrm{m}}])$.*

Proof. For reasons of space, the proof is provided in Appendix A.2. It is similar to the proof for the β property.

To prove local confluence modulo equivalence, we also have to prove the β property, i.e. we have to consider that if in a state a CHR transition is possible and the state is equivalent to another state, then the successor state and the equivalent state have to be joinable modulo equivalence. In the following lemma, we adapt the test for the α property to cover the β property.

The main idea is to reason from rule states, i.e. the head and guard constraints of rules, over all states. For this purpose, all rule states have to be extended by a minimal extension such that the invariant holds. Then all states that are equivalent to these extended rule states have to be shown to be joinable to the extended rule state after the rule has been applied. Unfortunately – depending on the invariant – in general there can be infinitely many such equivalent states. However, the idea still simplifies the proof procedure for the β property, as only rule states have to be considered in contrast to all states of the transition system.

This is not possible for general equivalence relations, but only for those that are compatible to the merge operator and that maintain the invariant.

Lemma 6 (β property test). *Let $\mathcal{P}$ be a CHR program, $\mathcal{I}$ an invariant, $\approx$ a $\diamond$-compatible equivalence relation that maintains $\mathcal{I}$ and let $\mathcal{M}^{\mathcal{I}}([\sigma])$ be well-defined for all rule states $[\sigma]$ in $\mathcal{P}$, then: $\mathcal{P}$ has the β property with respect to $\mathcal{I}$ and $\approx$ if and only if for all rule states $[\sigma]$ with successor state $[\sigma_1]$, all $[\sigma_2]$ with $[\sigma] \approx [\sigma_2]$ and all $[\sigma^1_{\mathrm{m}}] \in \mathcal{M}^{\mathcal{I}}([\sigma])$ and all $[\sigma^2_{\mathrm{m}}] \approx [\sigma^1_{\mathrm{m}}]$ where $\mathcal{I}([\sigma_2] \diamond [\sigma^2_{\mathrm{m}}])$ is satisfied, it holds that $([\sigma_1] \diamond [\sigma^1_{\mathrm{m}}]) \downarrow^{\approx} ([\sigma_2] \diamond [\sigma^2_{\mathrm{m}}])$.*

Proof. "$\Rightarrow$": This follows from Definition 4 and Lemma 4.

"$\Leftarrow$": Let $[\sigma], [\sigma_1]$ and $[\sigma_2]$ be CHR states where $\mathcal{I}([\sigma])$ and $\mathcal{I}([\sigma_2])$ hold and $[\sigma] \mapsto_r [\sigma_1]$ for some rule r and $[\sigma] \approx [\sigma_2]$. Since a rule is applicable in $[\sigma]$, there is a rule state $\sigma_r = \langle \_; \_; \mathbb{V} \rangle$ of rule r such that for some $[\delta_1] := [\langle \mathbb{G}; \mathbb{B}; \mathbb{V}' \rangle]$ it holds that $[\sigma] = [\sigma_r] \diamond_{\mathbb{V}} [\delta_1]$. The variables $\mathbb{V}$ from the rule r are not part of $[\sigma]$ and are therefore removed by the merging $\diamond_{\mathbb{V}}$.

By definition of the rule state (c.f. Definition 10) and definition of the state transition system (c.f. Definition 8), we also have that there is a state $[\sigma'_1]$ such that $[\sigma_r] \mapsto_r [\sigma'_1]$. Due to monotonicity (c.f. Lemma 2) it holds that $[\sigma_1] = [\sigma'_1] \diamond_{\mathbb{V}} [\delta_1]$.

Let $[\sigma_2] = [\sigma'_2] \diamond_{\mathbb{V}} [\delta_2]$ be a partition of $[\sigma_2]$ such that $[\sigma'_2] \approx [\sigma_r]$ and $[\delta_2] \approx [\delta_1]$. Such a partition exists since $\approx$ is $\diamond$-compatible and $[\sigma] \approx [\sigma_2]$ by precondition.

As $\mathcal{I}([\sigma])$ holds and w.l.o.g. $[\sigma]$ has no local variables (see the comment after Definition 12): $[\delta_1] \in \Sigma^{\mathcal{I}}([\sigma_r])$ and therefore $\exists [\sigma^1_{\mathrm{m}}] \in \mathcal{M}^{\mathcal{I}}([\sigma_r]).[\sigma^1_{\mathrm{m}}] \lhd [\delta_1]$. This means that there is a minimal element $[\sigma^1_{\mathrm{m}}]$ in the set of extensions of the rule state $[\sigma_r]$ that extend $[\sigma_r]$ such that the invariant holds.

It follows by definition of $\lhd$ that $\exists [\delta'_1].[\delta_1] = [\sigma^1_{\mathrm{m}}] \diamond [\delta'_1]$ and hence $[\sigma] = [\sigma_r] \diamond_{\mathbb{V}} ([\sigma_{\mathrm{m}}] \diamond [\delta'_1])$. By Lemma 1, we get $[\sigma] = ([\sigma_r] \diamond [\sigma_{\mathrm{m}}]) \diamond_{\mathbb{V}} [\delta'_1]$. Analogously, by substitution of $[\delta_1]$ in $[\sigma_1]$ and due to the split property of $\approx$, we find that $[\sigma_i] = ([\sigma'_i] \diamond [\sigma^i_{\mathrm{m}}]) \diamond_{\mathbb{V}} [\delta'_i]$ for $i = 1, 2$ where $[\sigma^1_{\mathrm{m}}] \approx [\sigma^2_{\mathrm{m}}]$.

Since $\mathcal{I}$ is maintained by $\approx$, we have by precondition that $([\sigma'_1] \diamond [\sigma^1_{\mathrm{m}}]) \downarrow^{\approx} ([\sigma'_2] \diamond [\sigma^2_{\mathrm{m}}])$. Since $[\sigma_1] = ([\sigma'_1] \diamond [\sigma^1_{\mathrm{m}}]) \diamond_{\mathbb{V}} [\delta'_1]$ and $[\sigma_2] = ([\sigma'_2] \diamond [\sigma^2_{\mathrm{m}}]) \diamond_{\mathbb{V}} [\delta'_2]$ and $[\delta'_1] \approx [\delta'_2]$, we have by Lemma 4 also that $([\sigma_1] \downarrow^{\approx} [\sigma_2])$.

Theorem 2 (confluence modulo $\approx$ w.r.t. invariant). *Let $\mathcal{I}$ be an invariant and $\mathcal{P}$ an $\mathcal{I}$-terminating CHR program. $\mathcal{P}$ has the α and β property with respect to $\mathcal{I}$ and an equivalence relation $\approx$ if and only if $\mathcal{P}$ is $\mathcal{I}$-confluent modulo $\approx$.*

Proof. Theorem 1 is used on the reduced state transition system that only contains states where the invariant holds.

Note that for testing the α property, the criterion only assumes a congruence relation, whereas for proving the β property the split property must hold as well and the invariant must maintain equivalence.

Example 8 (Item Sets (cont.)). It is shown that the program from Example 1 is $\mathcal{S}$-confluent modulo $\approx^{\mathrm{S}}$.

α **property** The only overlap that satisfies the invariant $\mathcal{S}$ has the shorthand notation $item(A), item(B), mset(L)$ with critical pair $item(B), mset([A|L])$ and $item(A), mset([B|L])$. It can be reduced to $mset([B, A|L]) \approx^{\mathrm{S}} mset([A, B|L])$.

β **property** All equivalences to the rule state have the form $[\langle item(A), mset(L); \top; \{A, L\}\rangle] \approx^{\mathrm{S}} [\langle item(A), mset(L'); \top; \{A, L'\}\rangle]$ where L' is a permutation of L. The preconditions of Lemma 6 are satisfied, since both states satisfy the invariant. Both states reduce to the goal stores $mset([A|L])$ and $mset([A|L'])$. It is clear that those two final states are equivalent and therefore joinable modulo $\approx^{\mathrm{S}}$. □

5 Discussion and Related Work

The α property test is decidable for terminating programs as long as the invariant and the equivalence relation (a congruence relation w.r.t. $\diamond$) are decidable and the set of minimal extensions is finite. In the β property test, the class of states that are equivalent to the rule state may be infinitely large in general.

In the multi-set example (c.f. Example 8), it can be seen that only one other state has to be considered to show joinability of all states equivalent to the rule state, since the CHR semantics allows for logical variables. In general, there might be more complicated equivalence relations that are more difficult to test.

Confluence modulo equivalence with invariants has been studied for a variant of CHR that includes non-logical built-in constraints [9,8]. This approach introduces a meta language for CHR to prove confluence modulo equivalence. It is claimed that the traditional proof methods for confluence in CHR expressed in first-order logic are not sufficient in the context of confluence modulo equivalence, especially with non-logical built-in constraints. The meta-level is claimed to allow proving confluence modulo equivalence for all equivalence relations. It is shown to be useful for programs with non-logical built-in constraints.

In many cases the analysis of programs with purely logical CHR is desired and invariants and equivalence relations behave in a way that allow for a more direct treatment. In our approach, no meta-level is necessary. It is directly available for the de facto standard of CHR semantics. It seems to us that the example in

[9] indicates that for proving confluence modulo equivalence with the meta-level approach, monotonicity and therefore $\diamond$-compatibility are used implicitly.

Invariant-based confluence (or observable confluence) for CHR without user-defined equivalence relations has been studied in [5,6]. In [7], it has been shown that the proposed partial order is not well-defined. Our approach integrates the corrected version of invariant-based confluence as found in [7]. Additionally, it extends the idea by confluence modulo user-defined equivalence relations.

6 Conclusion and Future Work

A sufficient and necessary criterion for confluence modulo equivalence w.r.t. an invariant has been presented and formally proven (c.f. Lemmas 4 to 6 and theorem 2). For this purpose, the set of *compatible* equivalence relations (c.f. Definitions 13 to 15) has been identified to behave well with this confluence criterion for CHR. When an equivalence relation has been shown to be compatible and maintains the invariant, it can be used directly for any program. In practice, it seems to be desirable that the equivalence relation maintains the invariant.

The approach is directly applicable for a non-trivial example (c.f. Example 8). It has been tested for other examples which indicates that the defined class of equivalence relations is actually meaningful. Decidability of the α property is maintained. For some invariants, the set of minimal extensions can be infinitely large and decidability is lost. Although the β property leads to an infinite number of states that have to be considered in general, the proofs are simplified tremendously, as only states equivalent to the finite number of rule states have to be considered.

In many cases it may suffice to only use an equivalence relation without an invariant. The problems originating from invariants are inexistent for those cases and our approach yields a sufficient and necessary criterion for confluence modulo equivalence without invariants.

For the future, we want to investigate how our approach can be unified with the meta-level approach [9] or other proof methods in the context of confluence such as case splitting. Furthermore, it could be interesting how non-confluent programs can be completed such that they become confluent modulo equivalence.

Acknowledgements

The authors would like to thank Henning Christiansen and Maja H. Kirkeby for the valuable discussions and ideas for future work. Additionally, the authors would like to thank the anonymous reviewers for their valuable comments.

References

1. Frühwirth, T.: Constraint Handling Rules. Cambridge University Press (2009)

2. Abdennadher, S., Frühwirth, T., Meuss, H.: On confluence of Constraint Handling Rules. In Freuder, E.C., ed.: Principles and Practice of Constraint Programming — CP96. Volume 1118 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (1996) 1–15
3. Abdennadher, S.: Operational semantics and confluence of constraint propagation rules. In Smolka, G., ed.: CP '97: Proc. Third Intl. Conf. Principles and Practice of Constraint Programming. Volume 1330 of Lecture Notes in Computer Science., Springer-Verlag (1997) 252–266
4. Abdennadher, S., Frühwirth, T., Meuss, H.: Confluence and semantics of constraint simplification rules. Constraints **4**(2) (1999) 133–165
5. Duck, G.J., Stuckey, P.J., Sulzmann, M.: Observable confluence for Constraint Handling Rules. In Dahl, V., Niemelä, I., eds.: ICLP '07. Volume 4670 of Lecture Notes in Computer Science., Springer-Verlag (September 2007) 224–239
6. Duck, G.J., Stuckey, P.J., Sulzmann, M.: Observable confluence for Constraint Handling Rules. In Schrijvers, T., Frühwirth, T., eds.: CHR '06, K.U.Leuven, Department of Computer Science, Technical report CW 452 (July 2006) 61–76
7. Raiser, F.: Graph Transformation Systems in Constraint Handling Rules: Improved Methods for Program Analysis. PhD thesis, Ulm University, Germany (2010)
8. Christiansen, H., Kirkeby, M.H.: Confluence modulo equivalence in Constraint Handling Rules. In Proietti, M., Seki, H., eds.: Logic-Based Program Synthesis and Transformation: 24th International Symposium, LOPSTR 2014, Canterbury, UK, September 9-11, 2014. Revised Selected Papers, Springer International Publishing (2015) 41–58
9. Christiansen, H., Kirkeby, M.H.: On proving confluence modulo equivalence for Constraint Handling Rules. Formal Aspects of Computing **29**(1) (2017) 57–95
10. Raiser, F., Betz, H., Frühwirth, T.: Equivalence of CHR states revisited. In Raiser, F., Sneyers, J., eds.: 6th International Workshop on Constraint Handling Rules (CHR), KULCW, Technical report CW 555 (July 2009) 33–48
11. Huet, G.: Confluent reductions: Abstract properties and applications to term rewriting systems. Journal of the ACM (JACM) **27**(4) (1980) 797–821
12. Ohlebusch, E.: Church-rosser theorems for abstract reduction modulo an equivalence relation. In Nipkow, T., ed.: Rewriting Techniques and Applications, Berlin, Heidelberg, Springer Berlin Heidelberg (1998) 17–31

A Proofs

A.1 Merge Compatibility of $\approx^{\mathrm{S}}$

The multi-set equivalence relation $\approx^{\mathrm{S}}$ from Example 7 is $\diamond$-compatible.

Proof. **Congruence Relation** Let $[\sigma], [\sigma'], [\rho], [\rho']$ be CHR states. From the definition it follows that if $[\sigma] \approx^{\mathrm{S}} [\sigma']$, the number of constraints (and in particular *mset* constraints) is equivalent. This means that if there is no *mset* constraint in $[\sigma]$ or $[\rho]$, then there is none in $[\sigma']$ or $[\rho']$. We use induction over the number n of constraints in the constraint store of $[\sigma]$. Since $\diamond_{\mathbb{V}}$ is commutative (c.f. [7, p. 51 sqq.]), this can be done w.l.o.g.

Base Case ($n = 0$) Let $[\sigma] = [\langle \emptyset; \mathbb{B}_\sigma; \mathbb{V}_\sigma \rangle]$. Since $[\sigma] \approx^{\mathrm{S}} [\sigma']$, we get $[\sigma'] = [\langle \emptyset; \mathbb{B}_\sigma; \mathbb{V}_\sigma \rangle]$. Let $[\rho] = [\langle \mathbb{G}_\rho; \mathbb{B}_\rho; \mathbb{V}_\rho \rangle] \approx^{\mathrm{S}} [\rho'] = [\langle \mathbb{G}_{\rho'}; \mathbb{B}_{\rho'}; \mathbb{V}_\rho \rangle]$. Then for all $\mathbb{V} : [\sigma] \diamond_{\mathbb{V}} [\rho] = [\langle \mathbb{G}_\rho; \mathbb{B}_\sigma \wedge \mathbb{B}_\rho; (\mathbb{V}_\sigma \cup \mathbb{V}_\rho) \setminus \mathbb{V} \rangle] \approx^{\mathrm{S}} [\langle \mathbb{G}_{\rho'}; \mathbb{B}_\sigma \wedge \mathbb{B}_{\rho'}; (\mathbb{V}_\sigma \cup \mathbb{V}_\rho) \setminus \mathbb{V} \rangle] = [\sigma'] \diamond_{\mathbb{V}} [\rho']$.

Induction Step ($n \to n+1$) We add a constraint c to the constraint store of a state $[\delta]$ with n constraints. If c is not a *mset* constraint, it is clear that the proposition holds. Let $c = mset(L)$ and $[\sigma] = [\langle \{c\} \uplus \mathbb{G}_\sigma; \mathbb{B}_\sigma; \mathbb{V}_\sigma \rangle]$. Since $[\sigma] \approx^{\mathrm{S}} [\sigma']$, we get by definition that

$$[\sigma'] = [\langle \{c'\} \uplus \mathbb{G}_{\sigma'}; \mathbb{B}_\sigma; \mathbb{V}_\sigma \rangle]$$

for a constraint $c' = mset(L')$ and L is a permutation of L'. Let

$$\begin{aligned}[\delta] :=& [\langle \mathbb{G}_\sigma; \mathbb{B}_\sigma; \mathbb{V}_\sigma \rangle] \\ \approx^{\mathrm{S}}& [\langle \mathbb{G}_{\sigma'}; \mathbb{B}_\sigma; \mathbb{V}_\sigma \rangle] =: [\delta']\end{aligned}$$

and

$$\begin{aligned}[\rho] :=& [\langle \mathbb{G}_\rho; \mathbb{B}_\rho; \mathbb{V}_\rho \rangle] \\ \approx^{\mathrm{S}}& [\langle \mathbb{G}_{\rho'}; \mathbb{B}_{\rho'}; \mathbb{V}_{\rho'} \rangle] =: [\rho'].\end{aligned}$$

By using the induction hypothesis, it follows that $[\delta] \diamond_{\mathbb{V}} [\rho] \approx^{\mathrm{S}} [\delta'] \diamond_{\mathbb{V}} [\rho']$ and hence

$$\begin{aligned}&[\sigma] \diamond_{\mathbb{V}} [\rho] \\ =& [\langle \{c\} \uplus \mathbb{G}_\sigma \uplus \mathbb{G}_\rho; \mathbb{B}_\sigma \wedge \mathbb{B}_\rho; (\mathbb{V}_\sigma \cup \mathbb{V}_\rho) \setminus \mathbb{V} \rangle] \\ \approx^{\mathrm{S}}& [\langle \{c'\} \uplus \mathbb{G}_{\sigma'} \uplus \mathbb{G}_{\rho'}; \mathbb{B}_\sigma \wedge \mathbb{B}_{\rho'}; (\mathbb{V}_\sigma \cup \mathbb{V}_{\rho'}) \setminus \mathbb{V} \rangle] \\ =& [\sigma'] \diamond_{\mathbb{V}} [\rho']\end{aligned}$$

by definition of $\diamond_{\mathbb{V}}$ and $\approx^{\mathrm{S}}$.

Split Property The property is proven by induction.

Base Case ($n = 0$) Let $[\sigma] = [\langle \emptyset; \mathbb{B}; \mathbb{V} \rangle] = [\sigma_1] \diamond_{\mathbb{V}} [\sigma_2]$ and $[\sigma] \approx^{\mathrm{S}} [\rho]$. If $[\sigma]$ has an empty goal store, it does not contain any *mset* constraints and hence $[\sigma] = [\rho]$. There is a trivial split $[\rho] = [\sigma_1] \diamond_{\mathbb{V}} [\sigma_2]$.

Induction Step ($n \to n+1$) Let $[\sigma] = [\langle \{c\} \uplus \mathbb{G}_\sigma; \mathbb{B}_\sigma; \mathbb{V}_\sigma \rangle] = [\sigma_1] \diamond_{\mathbb{V}} [\sigma_2]$ for a constraint c and $[\sigma] \approx^{\mathrm{S}} [\rho]$. There are two cases:

1. c is not a *mset* constraint. Therefore, $[\sigma] = [\rho]$. There is a trivial split $[\rho] = [\sigma_1] \diamond_{\mathbb{V}} [\sigma_2]$.
2. $c = mset(L)$ for some L. Then, since $[\sigma] \approx^{\mathrm{S}} [\rho]$, there is a $c' = mset(L')$ such that $[\rho] = [\langle \{c'\} \uplus \mathbb{G}_\rho; \mathbb{B}_\rho; \mathbb{V}_\rho \rangle]$ where L is a permutation of L' and

$$\begin{aligned}[\delta_\sigma] :=& [\langle \mathbb{G}_\sigma; \mathbb{B}_\sigma; \mathbb{V}_\sigma \rangle] \\ \approx^{\mathrm{S}}& [\langle \mathbb{G}_\rho; \mathbb{B}_\rho; \mathbb{V}_\rho \rangle] =: [\delta_\rho].\end{aligned}$$

From the induction hypothesis it follows that if $[\delta_\sigma] = [\delta_\sigma^1] \diamond_{\mathbb{V}} [\delta_\sigma^2]$, then there are $[\delta_\rho^1]$ and $[\delta_\rho^2]$, such that $[\delta_\rho] = [\delta_\rho^1] \diamond_{\mathbb{V}} [\delta_\rho^2]$. The constraint c from $[\sigma]$ can now be part of either $[\sigma_1]$ or $[\sigma_2]$. Let w.l.o.g. c be part of $[\sigma_1]$, i.e.

$$[\sigma_1] = [\langle \{c\} \uplus \mathbb{G}_{\delta_\sigma^1}; \mathbb{B}_{\delta_\sigma^1}; \mathbb{V}_{\delta_\sigma^1} \rangle]$$

where

$$[\delta_\sigma^1] = [\langle \mathbb{G}_{\delta_\sigma^1}; \mathbb{B}_{\delta_\sigma^1}; \mathbb{V}_{\delta_\sigma^1} \rangle].$$

In this case, $[\sigma_2] = [\delta_\sigma^2]$.
Analogously, we have that

$$[\rho_1] = [\langle \{c\} \uplus \mathbb{G}_{\delta_\rho^1}; \mathbb{B}_{\delta_\rho^1}; \mathbb{V}_{\delta_\rho^1} \rangle]$$

where

$$[\delta_\rho^1] = [\langle \mathbb{G}_{\delta_\rho^1}; \mathbb{B}_{\delta_\rho^1}; \mathbb{V}_{\delta_\rho^1} \rangle].$$

Since $[\delta_\rho^1] \approx^{\mathrm{S}} [\delta_\sigma^1]$ by induction hypothesis and L is a permutation of L', by definition of $\approx^{\mathrm{S}}$ in Example 7 it is clear that $[\sigma_i] \approx^{\mathrm{S}} [\rho_i]$ for $i = 1, 2$. □

A.2 Proof for α Property Test

In this section, the α property test from Lemma 5 is proven.

Let $\mathcal{P}$ be a CHR program, $\mathcal{I}$ an invariant, $\approx$ a congruence relation and let $\mathcal{M}^{\mathcal{I}}([\sigma])$ be well-defined for all overlaps σ, then: $\mathcal{P}$ has the α property with respect to $\mathcal{I}$ and $\approx$ if and only if for all overlaps σ with critical pairs (σ_1, σ_2) and all $[\sigma_{\mathrm{m}}] \in \mathcal{M}^{\mathcal{I}}([\sigma])$ holds $([\sigma_1] \diamond [\sigma_{\mathrm{m}}] \downarrow^{\approx} [\sigma_2] \diamond [\sigma_{\mathrm{m}}])$.

Proof. The α property test coincides with the invariant-based confluence test first presented for CHR in [7, p. 86, lemma 14.11]. However, the proof has to be adapted in the last step as joinability now allows states to join modulo an equivalence relation.

"$\Rightarrow$": This follows directly from definition 4 and lemma 4.

"$\Leftarrow$": Let $[\sigma], [\sigma_1]$ and $[\sigma_2]$ be CHR states where $\mathcal{I}([\sigma])$ holds and $[\sigma] \mapsto_{r_1} [\sigma_1]$ for some rule r_1 and $[\sigma] \mapsto_{r_2} [\sigma_2]$ for some rule r_2. By definition 11, there exists an overlap state $\sigma_{\mathrm{o}} = \langle \_; \_; \mathbb{V} \rangle$ of rule r_1 and r_2 where $\mathbb{V}$ contains all variables from r_1 and r_2 such that for some $[\delta] := [\langle \mathbb{G}; \mathbb{B}; \mathbb{V}' \rangle]$ it holds that $[\sigma] = [\sigma_{\mathrm{o}}] \diamond_{\mathbb{V}} [\delta]$. The variables $\mathbb{V}$ from the rules r_1 and r_2 are not part of $[\sigma]$ and are therefore removed by the merging $\diamond_{\mathbb{V}}$. Due to monotonicity (c.f. lemma 2), we have that

- $[\sigma_{\mathrm{o}}] \mapsto_{r_1} [\sigma_1']$ with $[\sigma_1] = [\sigma_1'] \diamond_{\mathbb{V}} [\delta]$, and
- $[\sigma_{\mathrm{o}}] \mapsto_{r_2} [\sigma_2']$ with $[\sigma_2] = [\sigma_2'] \diamond_{\mathbb{V}} [\delta]$.

If no such overlap exists, the two rule applications are independent and therefore trivially joinable.

As $\mathcal{I}([\sigma])$ holds and w.l.o.g. $[\sigma]$ only consists of global variables, we have that $[\delta] \in \Sigma^{\mathcal{I}}([\sigma_{\mathrm{o}}])$. Therefore there is a element in the set of minimal extensions that is less than or equal to $[\delta]$, i.e. $\exists [\sigma_{\mathrm{m}}] \in \mathcal{M}^{\mathcal{I}}([\sigma_{\mathrm{o}}]).[\sigma_{\mathrm{m}}] \lhd [\delta]$. This means that there is a minimal element $[\sigma_{\mathrm{m}}]$ in the set of extensions of the overlap state $[\sigma_{\mathrm{o}}]$ that extend $[\sigma_{\mathrm{o}}]$ such that the invariant holds.

It follows by definition of $\lhd$ that $\exists [\delta'].[\delta] = [\sigma_{\mathrm{m}}] \diamond [\delta']$ and hence $[\sigma] = [\sigma_{\mathrm{o}}] \diamond_{\mathbb{V}} ([\sigma_{\mathrm{m}}] \diamond [\delta'])$. By lemma 1, we get $[\sigma] = ([\sigma_{\mathrm{o}}] \diamond [\sigma_{\mathrm{m}}]) \diamond_{\mathbb{V}} [\delta']$. Analogously,

since $\mathcal{I}([\sigma_i]), i = 1, 2$ holds due to the definition of an invariant, we find that $[\sigma_i] = ([\sigma'_i] \diamond [\sigma_\mathrm{m}]) \diamond_\mathbb{V} [\delta']$ for $i = 1, 2$.

At this point the proof differs from confluence without an equivalence relation. By the precondition we now only have that $([\sigma'_1] \diamond [\sigma_\mathrm{m}]) \downarrow^{\approx} ([\sigma'_2] \diamond [\sigma_\mathrm{m}])$, i.e. modulo an equivalence relation. Since $[\sigma_1] = ([\sigma'_1] \diamond [\sigma_\mathrm{m}]) \diamond_\mathbb{V} [\delta']$ and $[\sigma_2] = ([\sigma'_2] \diamond [\sigma_\mathrm{m}]) \diamond_\mathbb{V} [\delta']$, we can apply lemma 4 due to reflexivity of $\approx$ (i.e. $[\delta'] \approx [\delta']$) and get $([\sigma_1] \downarrow^{\approx} [\sigma_2])$.