Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 253 (2009) 91–111
www.elsevier.com/locate/entcs

# Strong Joinability Analysis for Graph Transformation Systems in CHR

Frank Raiser[1] Thom Frühwirth[2]

*Institute for Software Engineering and Compiler Construction*
*Ulm University*

**Abstract**

The notion of confluence is prevalent in graph transformation systems (GTS) as well as constraint handling rules (CHR). This work presents a generalized embedding of GTS in CHR that allows to consider strong derivations in confluence analyses. Confluence of a terminating CHR program is decidable, whereas confluence of a terminating GTS is undecidable. We show that observable confluence in CHR coincides with a sufficient criterion for confluence of the embedded GTS. For this purpose the automatic confluence check for CHR can be reused.

*Keywords:* Graph Transformation Systems, Constraint Handling Rules, Confluence

## 1 Introduction

Constraint handling rules (CHR) [6] allow for rapid prototyping and efficient implementation of constraint-based algorithms. Besides constraint reasoning, CHR have been used for various tasks including theorem proving, parsing, and multiset rewriting [6].

Graph transformation systems (GTS) are used to describe complex structures and systems in a concise, readable, and easily understandable way. They have applications ranging from implementations of programming languages over model transformations to graph-based models of computation [3,5].

In this work we present an embedding of graph transformation systems in CHR allowing us to perform strong derivations on partially defined graphs. This behavior provides the basis for the analysis of strong joinability of critical pairs presented in this work. Deciding strong joinability of critical pairs comes for free as a result of the observable confluence check of the corresponding CHR program containing

[1] frank.raiser@uni-ulm.de
[2] thom.fruehwirth@uni-ulm.de


doi:10.1016/j.entcs.2009.10.019

the embedded GTS. This work is a revised and extended version of [10]. Its main improvements are a more succinct encoding of GTS in CHR and a stronger correspondence between strong joinability in GTS and confluence in CHR.

We begin with the introduction of the necessary notions of graph transformation systems and CHR in Sect. 2. Section 3 then presents our proposed encoding of a GTS in CHR, for which Sect. 4 proves soundness and completeness. Finally, Sect. 5 introduces observable confluence and its application as a sufficient criterion for confluence of an embedded GTS, before we conclude in Sect. 6.

## 2 Preliminaries

In this section we introduce the required formalisms for graph transformation systems and constraint handling rules.

### *2.1 Graph Transformation System (GTS)*

The following definitions for graphs and graph transformation systems have been adapted from [5].

A *graph* $G = (V, E, \mathrm{src}, \mathrm{tgt})$ consists of a set $V$ of nodes, a set $E$ of edges and two morphisms $\mathrm{src}, \mathrm{tgt} : E \to V$ specifying source and target of an edge, respectively. A *type graph* $TG$ is a graph with unique labels for all nodes and edges.

For the purpose of simplicity, we avoid an additional label morphism in favor of identifying variable names with labels. For multiple graphs we refer to the node set $V$ of a graph $G$ as $V_G$ and analogously for edge sets and the $\mathrm{src}, \mathrm{tgt}$ morphisms. We further define the degree of a node as $\deg : V \to \mathbb{N}, v \mapsto \#\{e \in E \mid \mathrm{src}(e) = v\} + \#\{e \in E \mid \mathrm{tgt}(e) = v\}$. As there are often multiple graphs containing the same node due to inclusion morphisms we use $\deg_G(v)$ to specify the degree of a node $v$ with respect to the graph $G$. When the context graph is clear the subscript is omitted.

A *typed graph* $G$ is a tuple $(V, E, \mathrm{src}, \mathrm{tgt}, \mathrm{type}, TG)$ where $(V, E, \mathrm{src}, \mathrm{tgt})$ is a graph, $TG$ a type graph, and type a morphism with $\mathrm{type} = (\mathrm{type}_V, \mathrm{type}_E)$ and $\mathrm{type}_V : V \to TG_V, \mathrm{type}_E : E \to TG_E$. The type morphism is a graph morphism, therefore, it has to satisfy the following condition: $\forall e \in E : \mathrm{type}_V(\mathrm{src}(e)) = \mathrm{src}_{TG}(\mathrm{type}_E(e)) \wedge \mathrm{type}_V(\mathrm{tgt}(e)) = \mathrm{tgt}_{TG}(\mathrm{type}_E(e))$

A *Graph Transformation System* (GTS) is a tuple consisting of a type graph and a set of graph production rules. A *graph production rule* – also simply called *rule* if the context is clear – is a tuple $p = (L \xleftarrow{l} K \xrightarrow{r} R)$ of typed graphs $L, K$, and $R$ with inclusion morphisms $l : K \to L$ and $r : K \to R$.

We distinguish two kinds of typed graphs: *rule graphs* and *host graphs*. Rule graphs are the graphs $L, K, R$ of a graph production rule $p$ and host graphs are graphs to which the graph production rules can be applied. We, furthermore, make use of graph transformations based on the double-pushout approach (DPO) as defined in [5]. Most notably, we require a so-called match morphism $m : L \to G$ to apply a rule $p$ to a typed host graph $G$. The transformation yielding the typed

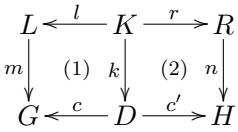



Fig. 1. Double-pushout approach

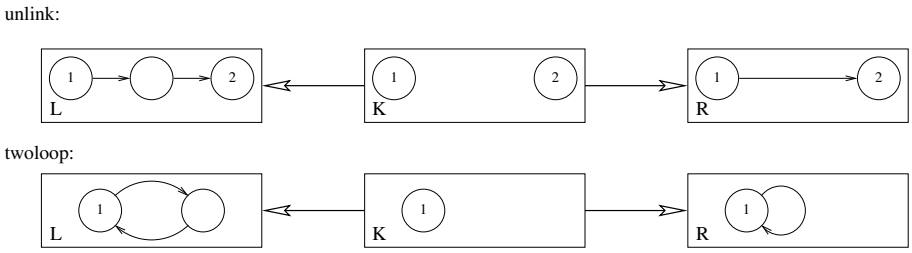



Fig. 2. Graph transformation system for recognizing cyclic lists

graph $H$ is written as $G \overset{p,m}{\Longrightarrow} H$. $H$ is given mathematically by constructing $D$ as shown in Figure 1, such that (1) and (2) are pushouts. Intuitively, the graph $L$ on the left-hand side is matched as a subgraph of $G$ and its occurrence in $G$ is then replaced by the right-hand side graph $R$. The intermediate graph $K$ is the context graph containing those items in $L$ that are preserved by the rule.

A graph production rule $p$ can only be applied to a host graph $G$ if the following gluing condition is satisfied. The *gluing condition* [5] is based on the set of gluing points $GP = l(K)$, the set of identification points $IP = \{v \in V_L \mid \exists w \in V_L, w \neq v : m(v) = m(w)\} \cup \{e \in E_L \mid \exists f \in E_L, e \neq f : m(e) = m(f)\}$, and the set of dangling points $DP = \{v \in V_L \mid \exists e \in E_G \setminus m(E_L) : \mathrm{src}_G(e) = m(v) \vee \mathrm{tgt}_G(e) = m(v)\}$ and it is defined as $IP \cup DP \subseteq GP$.

**Example 2.1** Figure 2 shows two graph production rules which make up a graph transformation system for detecting cyclic lists. The basic idea of the *unlink* rule is to remove intermediate nodes of the list, while the *twoloop* rule replaces the cyclic list consisting of two nodes by a single node with a loop. To detect if a host graph is a cyclic list the GTS is applied to the host graph until exhaustion. The host graph then is a cyclic list if and only if the final graph consists of a single node with a loop [3].

Note that the example makes use of the type graph consisting only of a single node with a loop. Furthermore, we use a shorthand notation that only shows the morphisms $l$ and $r$ implicitly by the labels of the nodes which are mapped onto each other. Nodes and edges which are removed or added in the graphs $L$ or $R$ are not labeled, as there is no node or edge in $K$ which is mapped to them.

In general, the DPO approach allows for the match morphism $m$ to be non-injective. For injective match morphisms the set $IP$ of identification points is guaranteed to be $\emptyset$. For the remainder of this work we only consider injective match morphisms, as non-injective ones can be simulated as follows: given a rule $p = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$ and a non-injective match morphism $m$ it holds $\forall v, w \in V_L, v \neq w$ with $m(v) = m(w)$ that the rule is only applicable, if $v, w \in l(V_K)$, i.e. only nodes which

are not removed by the rule application are allowed to be matched non-injectively – otherwise $IP \not\subseteq GP$. Therefore, it is possible to add another rule $p'$ which is derived from $p$ by merging the nodes $v$ and $w$ into a node $v_w$ in all three graphs of the rule. Thus, the non-injective matching with $m(v) = m(w)$ can be simulated by injectively matching $v_w$ to $m(v_w)$ where $m(v_w)$ is the same node in $G$ as $m(v)$. The same argumentation holds for edges, analogously. Therefore, we can restrict ourselves to injective match morphisms by extending the set of rules with new rules for all possible merges of nodes and edges in the graph $K$. This simplifies the generic gluing condition to $DP \subseteq GP$.

In Sect. 5 we also require the following definition of a track morphism. Intuitively, the track morphism is defined for a node or edge, if it is not removed by the rule application.

**Definition 2.2** [Track Morphism] Given $G \Longrightarrow H$ the *track morphism* $\mathrm{tr}_{G \Rightarrow H} : G \to H$ is the partial graph morphism defined by

$$\mathrm{tr}_{G \Rightarrow H}(x) = \begin{cases} c'(c^{-1}(x)) & \text{if } x \in c(D), \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Here $c : D \to G$ and $c' : D \to H$ are the morphisms in the lower row of Fig. 1 and $c^{-1} : c(D) \to D$ maps each item $c(x)$ to $x$.

The track morphism of a derivation $\Delta : G_0 \Rightarrow^* G_n$ is defined by $\mathrm{tr}_\Delta = \mathrm{id}_{G_0}$ if $n = 0$ and $\mathrm{tr}_\Delta = \mathrm{tr}_{G_1 \Rightarrow^* G_n} \circ \mathrm{tr}_{G_0 \Rightarrow G_1}$ otherwise, where $\mathrm{id}_{G_0}$ is the identity morphism on $G_0$.

### *2.2 Constraint Handling Rules (CHR)*

This section presents the syntax and operational semantics of constraint handling rules [6]. Constraints are first-order predicates which we separate into *built-in constraints* and *user-defined constraints*. Built-in constraints are provided by the constraint solver while user-defined constraints are defined by a CHR program. In this work we consider a subset of CHR where *Simplification* rules are of the form

$$\mathit{Rulename} \ @ \ H_1, \ldots, H_i \Leftrightarrow B_1, \ldots, B_k$$

where *Rulename* is an optional unique identifier of a rule, the *head* $H = H_1, \ldots, H_i$ is a non-empty conjunction of user-defined constraints, and the *body* $B = B_1, \ldots, B_k$ is a conjunction of built-in and user-defined constraints. Note that we make sloppy use of the terms conjunction, sequence, and multiset with respect to $H_1, \ldots, H_i$ and $B_1, \ldots, B_k$.

The operational semantics is based on an underlying *constraint theory* $\mathcal{CT}$ for the built-in constraints and a *state*, which is a tuple $\langle G, C, \mathcal{V} \rangle$ where $G$ is a goal store, i.e. a multiset of user-defined constraints, $C$ is a conjunction of built-in constraints, and $\mathcal{V}$ is the set of global variables-of-interest [6].

A simplification rule of the form $r \ @ \ H \Leftrightarrow B$ is *applicable* to a state $\langle E \wedge G, C, \mathcal{V} \rangle$ if $CT \models \forall (C \to \exists \overline{x}(H = E))$ where $\overline{x}$ are the variables in $H$ and $=$ is syntactic equality. We then define the following state transition for its application: $\langle E \wedge G, C, \mathcal{V} \rangle \mapsto^r \langle B_u \wedge G, (H = E) \wedge C \wedge B_b, \mathcal{V} \rangle$ where $B = B_u \cup B_b$ with $B_u$ being user-defined and $B_b$ being built-in constraints. We use $\mapsto$ when the applied rule is

not of interest, and as usual, $\mapsto^*$ denotes the reflexive-transitive closure of the $\mapsto$ relation.

Given a simplification rule $p \ @ \ H \Leftrightarrow B$ and a state $S = \langle E \cup G, C, \mathcal{V} \rangle$ such that $p$ is applicable to $S$ we define for the involved match $\eta(p, S) = (E, C \wedge (H = E))$.

When comparing different states for confluence we make use of an equivalence relation $\equiv$ on CHR states [6]. This equivalence accounts for different syntactical representations, including renaming of local variables, equality substitutions, and logically equivalent built-in stores.

**Example 2.3** The following two rules are part of a CHR handler for the boolean `and` constraint. The `and` constraint is ternary here with the implicit meaning that $\texttt{and}(X, Y, Z)$ holds iff $X \wedge Y = Z$.

$$r_1 \ @ \ \texttt{and}(X, X, Z) \quad \Leftrightarrow \quad Z = X$$
$$r_2 \ @ \ \texttt{and}(X, Y, 1) \quad \Leftrightarrow \quad X = 1, Y = 1$$

For a CHR program consisting of these two rules we can consider an initial state $\langle \texttt{and}(0, 0, N) \cup \texttt{and}(A, B, C), C = 1, \{N, A, B, C\} \rangle$ as input, resulting in the following computation. The underlined constraints are matched to one of the rule heads and removed by the rule application.

$\langle \underline{\texttt{and}(0, 0, N)} \cup \texttt{and}(A, B, C), C = 1, \{N, A, B, C\} \rangle$
$\mapsto^{r_1} \langle \underline{\texttt{and}(A, B, C)}, C = 1 \wedge (X = 0 \wedge Z = N \wedge Z = X), \{N, A, B, C\} \rangle$
$\mapsto^{r_2} \langle \underline{\emptyset}, C = 1 \wedge (X = 0 \wedge Z = N \wedge Z = X) \wedge (X' = A \wedge Y' = B \wedge C = 1 \wedge X' = 1 \wedge Y' = 1), \{N, A, B, C\} \rangle$

As this example shows the built-in store can include redundant information when the above transition definition is applied directly. CHR implementations simplify the built-in store with respect to the variables of interest using the built-in solver for the constraint theory $\mathcal{CT}$. This yields the following simplification of the final state above: $\langle \emptyset, N = 0 \wedge A = 1 \wedge B = 1 \wedge C = 1, \{N, A, B, C\} \rangle$. This state is equivalent to the final state above, i.e. the two states are contained in the $\equiv$ relation.

**Example 2.4** An important property of the equivalence relation $\equiv$ between CHR states is equivalence modulo renaming of local variables. In this work we make use of this property to deal with graph isomorphism in CHR. Without going into details on the encoding of graphs in CHR yet, consider the following states $\sigma_1, \sigma_2$, and $\sigma_3$:
$\sigma_1 = \langle \text{node}(N, 1) \cup \text{node}(M, 1) \cup \text{edge}(E, N, M), \top, \{N\} \rangle$
$\sigma_2 = \langle \text{node}(N, 1) \cup \text{node}(M', 1) \cup \text{edge}(E', N, M'), \top, \{N\} \rangle$
$\sigma_3 = \langle \text{node}(N', 1) \cup \text{node}(N, 1) \cup \text{edge}(\hat{E}, N', N), \top, \{N\} \rangle$

The variable $N$ is a global variable in all these states and the remaining variables are local. Therefore, $\sigma_1 \equiv \sigma_2$ as they differ only by renaming of local variables. This is similar to considering isomorphism between two graphs, each consisting of two nodes connected by an edge. However, in CHR we can also consider these graphs in a different way, as it holds that $\sigma_3 \not\equiv \sigma_1$ although the graph described by $\sigma_3$ is an isomorphic graph. This is due to the global variable $N$ occurring as a source of the edge in $\sigma_1$, but as a target in $\sigma_3$. This distinction is the basis of our strong joinability analysis.

# 3 Representation of Graphs in CHR

In order to embed a GTS in CHR, we have to encode its graph production rules as CHR rules and provide a conjunction of goal constraints corresponding to the host graph. To this end, we provide a correspondence between graphs and their representation by CHR constraints given by the constructions in Sect. 3.1. Section 3.2 presents the encoding of the rules of the GTS for recognizing cyclic lists and a complete example derivation.

## *3.1 CHR Encoding of a GTS*

For encoding a GTS in CHR we first determine the constraints needed for encoding the rules and host graph. At this point we require the GTS to be typed, so we can directly infer the necessary constraints from the corresponding type graph as explained in Def. 3.1. Note that this is not a restriction though, as every untyped graph can be typed over the type graph consisting of a single node with a loop.

**Definition 3.1** [Type Graph Encoding] For a type graph $TG$ we define the set $\mathcal{C}$ of required constraints to encode graphs typed over $TG$ as the smallest set including $v/2 \in \mathcal{C}$ for $v \in V_{TG}$ and $e/3 \in \mathcal{C}$ for $e \in E_{TG}$.

We assume all nodes and edges of the type graph $TG$ to be uniquely labeled such that the introduced constraints have unique names as well. Note that when annotating host graphs with these labels they can occur multiple times, i.e. their uniqueness is restricted to the type graph only.

**Definition 3.2** [Typed Graph Encoding] For a typed graph $G$ based on a type graph $TG$ the set of constraints encoding $G$ is defined differently for host and rule graphs. We define the following mappings for the encoding for an infinite set of variables VARS:

- $[\text{type}_G(x)]$ denotes the corresponding constraint name for encoding a node or edge of the given type.
- $\text{var} : G \to \text{VARS}, x \mapsto X_x$ such that $X_x$ is a unique variable associated to $x$, i.e. var is injective for the set of all graph nodes and edges.
- $\text{dvar} : V_G \to \text{VARS}, x \mapsto X_x$ such that $X_x$ is a unique variable associated to $x$, i.e. dvar is injective for the set of all graph nodes and edges.

Using these mappings we define the following encoding of graphs:

$$\texttt{chr}_G(\tau, x) = \begin{cases} [\text{type}_G(x)](\text{var}(x), \deg_G(x)) & \text{if } x \in V_G \wedge \tau = \text{host} \\ [\text{type}_G(x)](\text{var}(x), \text{dvar}(x)) & \text{if } x \in V_G \wedge \tau = \text{keep} \\ [\text{type}_G(x)](\text{var}(x), \text{var}(\text{src}(x)), \text{var}(\text{tgt}(x))) & \text{if } x \in E_G \end{cases}$$

We use the notations $\texttt{chr}(\text{host}, G) = \{\texttt{chr}_G(\text{host}, x) \mid x \in G\}$ and $\texttt{chr}(\text{keep}, G) = \{\texttt{chr}_G(\text{keep}, x) \mid x \in G\}$. Furthermore, we omit the index $G$ if the context is clear. For a node $v$ encoded with $\texttt{chr}(\text{keep}, v)$ we call $\text{dvar}(v)$ its *degree variable*.

Section 4 discusses the importance of degree variables with respect to the encoded GTS. Intuitively, nodes using these cannot be removed by a rule application. These nodes prove to be vital for the strong joinability analysis presented in Sect. 5.

**Example 3.3** [cont] For our example of the GTS for recognizing cyclic lists every node in the typed graph has the same type, just like every edge has the same type. Based on this we need the following constraints: $\mathrm{node}/2, \mathrm{edge}/3$

The host graph $G$ with a cyclic list consisting of exactly two nodes is encoded in $\mathtt{chr}(\mathrm{host}, G)$ as $\mathrm{node}(N_1, 2), \mathrm{node}(N_2, 2), \mathrm{edge}(E_1, N_1, N_2), \mathrm{edge}(E_2, N_2, N_1)$.

The same graph $G$ occurring as a rule graph is encoded in $\mathtt{chr}(\mathrm{keep}, G)$ as follows: $\mathrm{node}(N_1, D_1), \mathrm{node}(N_2, D_2), \mathrm{edge}(E_1, N_1, N_2), \mathrm{edge}(E_2, N_2, N_1)$.

We can now encode a complete graph production rule based on these definitions:

**Definition 3.4** [GTS Rule in CHR] For a graph production rule $p = (L \xleftarrow{l} K \xrightarrow{r} R)$ from a GTS we define $\rho(p) = (C_L, C_R)$ with

- $C_L = \{\mathtt{chr}(\mathrm{keep}, x) \mid x \in K\} \cup \{\mathtt{chr}(\mathrm{host}, x) \mid x \in L \setminus K\}$
- $C_R = \{\mathtt{chr}(\mathrm{host}, x) \mid x \in R \setminus K\} \cup \{\mathtt{chr}(\_, e) \mid e \in E_K\}$
  $\cup \{\mathtt{chr}(\mathrm{keep}, v'), \mathrm{var}(v) = \mathrm{var}(v'), \mathrm{dvar}(v') = \mathrm{dvar}(v) - \deg_L(v) + \deg_R(v) \mid v \in V_K\}$

The rule $p$ is then encoded in CHR using $\rho(p) = (C_L, C_R)$ and in abuse of notation we use $\rho(p)$ for the CHR rule $p \mathbin{@} C_L \Leftrightarrow C_R$ as well as for the tuple $(C_L, C_R)$.

### *3.2 Example Computation*

Soundness and completeness of the above encoding is shown in Sect. 4, however, to ease the understanding, we present a complete computation here for our cyclic list example. The following two rules are the CHR encoding of the rules from Fig. 2:

*unlink* @ $\mathrm{node}(N_1, D_1), \mathrm{node}(N, 2), \mathrm{node}(N_2, D_2),$
$\mathrm{edge}(E_1, N_1, N), \mathrm{edge}(E_2, N, N_2)$
$\Leftrightarrow$
$\mathrm{node}(N_1', D_1'), N_1' = N_1, D_1' = D_1{+}1{-}1,$
$\mathrm{node}(N_2', D_2'), N_2' = N_2, D_2' = D_2{+}1{-}1, \mathrm{edge}(E, N_1, N_2)$

*twoloop* @ $\mathrm{node}(N_1, D_1), \mathrm{node}(N, 2), \mathrm{edge}(E_1, N_1, N), \mathrm{edge}(E_2, N, N_1)$
$\Leftrightarrow$
$\mathrm{node}(N_1', D_1'), N_1' = N_1, D_1' = D_1{+}2{-}2, \mathrm{edge}(E, N_1, N_1)$

The following state $S$ is the encoding of a simple cycle consisting of three nodes. To demonstrate strong computations the degree of the third node is left uninstantiated:

$S = \langle \mathrm{node}(N_1, 2) \cup \mathrm{node}(N_2, 2) \cup \mathrm{node}(N_3, D_3) \cup \mathrm{edge}(E_1, N_1, N_2)$
$\cup\, \mathrm{edge}(E_2, N_2, N_3) \cup \mathrm{edge}(E_3, N_3, N_1), \top, \{N_1, N_2, N_3, E_1, E_2, E_3, D_3\}\rangle$

Rule *unlink* can then be applied to the state $S$ resulting in the following state $S'$:

$S' = \langle \mathrm{node}(N_1, 2) \cup \mathrm{node}(N_3, D_3') \cup \mathrm{edge}(E, N_1, N_3) \cup \mathrm{edge}(E_3, N_3, N_1),$
$D_3' = D_3{+}1{-}1, \{N_1, N_2, N_3, E_1, E_2, E_3, D_3\}\rangle$

Finally, rule *twoloop* can be applied to $S'$ to remove node $N_1$, resulting in the following final state $S''$:
$S'' = \langle \text{node}(N_3, D_3'') \cup \text{edge}(E', N_3, N_3), D_3' = D_3{+}1{-}1$
$\wedge\, D_3'' = D_3'{+}2{-}2, \{N_1, N_2, N_3, E_1, E_2, E_3, D_3\}\rangle$

As can be seen from the state $S''$ the built-in store contains a chain of degree adjustments for nodes with initially uninstantiated degree and the node $N_3$ remains throughout the whole computation. These properties are investigated more thoroughly in the following section.

## 4 Soundness and Completeness

In this section we show soundness and completeness of our encoding. Whereas in [9] we showed soundness and completeness only for an encoding corresponding to $\texttt{chr}(\text{host}, G)$ we generalize these results in this work for an encoding based on $\texttt{chr}(\text{keep}, G)$. The following definitions specify these strictly more generic host graph encodings, as well as some properties of our encoding used throughout the remainder of this section.

We then discuss in Sect. 4.1 that CHR rule application respects the gluing condition, before Sect. 4.2 shows that rule applicability of GTS and CHR coincide. Finally, Sect. 4.3 combines these results to prove soundness and completeness.

In Sect. 3.2 the example shows that during the CHR computations we may encounter states which are not a direct encoding of a host graph. Nevertheless, these states represent a graph $G$ without explicitly specifying node degrees. In order to uniformly argue on all of these states we introduce an invariant on states which, intuitively, is satisfied when a state is an encoding of a graph.

**Definition 4.1** [Invariant] An *invariant* $\mathcal{I}(S)$ is a property such that for all $S_0$ and $S_1$, we have that if $S_0 \to S_1$ (or $S_0 \equiv S_1$) and $\mathcal{I}(S_0)$ holds then $\mathcal{I}(S_1)$ holds.

**Definition 4.2** [Graph Invariant] The *graph invariant* $\mathcal{G}(S)$ with $S = \langle E, C, \mathcal{V}\rangle$ holds iff there exist a graph $G$ and a conjunction $C'$ of constraints that are of the form $\text{dvar}(v) = \deg_G(v)$, such that $\langle E, C \wedge C', \emptyset\rangle \equiv \langle \texttt{chr}(\text{host}, G), \top, \emptyset\rangle$. For a state $S$ for which $\mathcal{G}(S)$ holds with a graph $G$ we say $S$ is a *$\mathcal{G}$-state based on $G$*.

The fact, that $\mathcal{G}$ is an invariant is shown in Cor. 4.12 using other results from this section which only make use of the definition of $\mathcal{G}$, but do not require it to be an invariant. The following definition allows us to argue directly on those nodes of a $\mathcal{G}$-state based on $G$ for which the state has uninstantiated degree variables:

**Definition 4.3** [Strong Nodes] For a CHR state $S = \langle \texttt{chr}(\text{keep}, G), C, \mathcal{V}\rangle$ which is a $\mathcal{G}$-state based on $G$ we define the set of *strong nodes* as $\mathcal{S}(S) = \{v \in V_G \mid \text{dvar}(v) = \deg_G(v) \notin C\}$.

An important feature of strong nodes is that they cannot be removed by any rule, because the uninstantiated degree variable cannot match the constant degree used in rule heads for nodes that are deleted. This property is used in Sect. 5 where

overlaps of rules are investigated and strong nodes are responsible for enforcing strong joinability.

Next we show how a matching in one formalism can be transferred to the other formalism:

**Definition 4.4** [GTS Match Implies CHR Match] Let $G$ be a host graph, $p = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$ a GTS rule, and $m$ a match morphism such that $G \overset{p,m}{\Longrightarrow} G'$. Furthermore, let $S = \langle \mathtt{chr}(\text{keep}, G), C, \mathcal{V} \rangle$ be a $\mathcal{G}$-state based on $G$ and $\rho(p) = (C_L, C_R)$.

Then $m$ implies the CHR match $\eta(\rho(p), S) = (\tilde{G}, Eq)$ with
$\tilde{G} = \{\mathtt{chr}(\text{keep}, x) \mid x \in m(L)\}$
$Eq = C \wedge \{\text{var}(v) = \text{var}(m(v)) \mid v \in V_L\} \wedge \{\text{var}(e) = \text{var}(m(e)) \mid e \in E_L\}$
$\wedge \{\text{dvar}(v) = \text{dvar}(m(v)) \mid v \in V_K\}$.

**Definition 4.5** [CHR Match Implies GTS Match] Let $S = \langle \mathtt{chr}(\text{keep}, G), C, \mathcal{V} \rangle$ be a $\mathcal{G}$-state based on $G$, $\rho(p)$ be the CHR rule for $p = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$ , and $S \mapsto S'$ using rule $\rho(p)$ with match $\eta(p, S) = (\tilde{G}, Eq)$.

Then $\eta(p, S)$ implies the injective GTS match morphism $m : L \to G$ with
$v \mapsto v'$ with $\text{var}(v) = \text{var}(v') \in Eq \wedge [\text{type}_L(v)](\text{var}(v'), \_) \in \tilde{G}$
$e \mapsto e'$ with $\text{var}(e) = \text{var}(e') \in Eq \wedge [\text{type}_L(e)](\text{var}(e'), \_, \_) \in \tilde{G}$.

Note that the implied CHR match from Def. 4.4 matches all constraints in the head of the corresponding CHR rule and the implied match $m$ from Def. 4.5 always corresponds to an injective total graph morphism.

### *4.1 Gluing Condition*

As applicability of GTS rules is tied to satisfaction of the gluing condition we first ensure that our encoding given in Sect. 3 adheres to this restriction as well. It follows from the definition of a dangling edge, that one exists if and only if $DP \not\subseteq GP$.

**Lemma 4.6 (Dangling Edges)** *If the application of rule $p = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$ to $G$ using match $m$ violates the gluing condition, such that $DP \not\subseteq GP$, then the corresponding CHR rule $\rho(p) = (C_L, C_R)$ is not applicable to a $\mathcal{G}$-state based on $G$ using the match implied by Def. 4.4.*

**Proof.** As $DP \not\subseteq GP$ there exists a dangling edge. Let $e \in E_G$ be a dangling edge which is adjacent to $v_G \in V_G$, such that for a $v \in V_L \setminus V_K : m(v) = v_G$. Due to Definition 3.4, $[\text{type}_L(v)](\text{var}(v), k) \in C_L$ with $k = \deg_L(v)$. This means that there are $k$ edges adjacent to the node $v$ in the rule graph $L$. When matching this rule graph injectively to the host graph $G$ we need to identify each of these edges with an edge in $E_G$ adjacent to $m(v) = v_G$. By the definition of a dangling edge, the edge $e$ is not among those $k$ edges as $e \in E_G \setminus m(E_L)$. Therefore, we have $\deg_G(v_G) = l > k$.

The constraint corresponding to $v_G$ in the goal is $[\text{type}_G(v_G)](\text{var}(v_G), l)$, or $[\text{type}_G(v_G)](\text{var}(v_G), \text{dvar}(v_G))$ depending on whether the degree variable is instantiated in $C$ or not. In the first case a match is impossible due to $l \neq k$ and in the

latter case a match is impossible, as a variable cannot be matched to a constant in CHR. Therefore, the rule is not applicable as the gluing condition is violated. □

### *4.2 Applicability*

Next we show that applicability of GTS rules and the corresponding rules encoded in CHR coincides. The following two lemmata show that the implied matchings are sufficient for the corresponding rule applicability:

**Lemma 4.7 (GTS Rule Applicability)** *Let $\rho(p) = (C_L, C_R)$ be applicable to a $\mathcal{G}$-state based on $G$ then $p = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$ is applicable to $G$ using the implied match morphism $m$ from Def. 4.5.*

**Proof.** Let $m$ be the match implied by Def. 4.5. For $p$ to be applicable to $G$ the gluing condition has to be satisfied. As $m$ is injective this is equivalent to showing the non-existence of a dangling edge. However, if the application of $p$ to $G$ using match $m$ violates the gluing condition, Lemma 4.6 states that $\rho(p)$ will not be applicable using this match, which is a contradiction. Therefore, the gluing condition is satisfied, no dangling edge exists, and $p$ is applicable to $G$. □

**Lemma 4.8 (Graph Rule Applicability)** *Let $p = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$ , $G \overset{p,m}{\Longrightarrow} G'$, and let $S = \langle \texttt{chr}(\text{keep}, G), C, \mathcal{V}\rangle$ be a $\mathcal{G}$-state based on $G$.*

*If $m(V_L \setminus V_K) \cap \mathcal{S}(S) = \emptyset$, then $\rho(p) = (C_L, C_R)$ is applicable to $S$ using the implied match $\eta(p, S) = (\tilde{G}, Eq)$ from Def. 4.4.*

**Proof.** For the CHR rule to be applicable we have to show that $C$ implies a possible match. We show this individually for nodes and edges:

Consider a node $v \in V_K$. Then there is a constraint $[\text{type}_L(v)](\text{var}(v), \text{dvar}(v)) \in C_L$. As $m$ is a graph morphism it holds that $\text{type}_L(v) = \text{type}_G(m(v))$. We also have $[\text{type}_G(m(v))](\text{var}(m(v)), \text{dvar}(m(v))) \in \tilde{G}$, $\text{var}(v) = \text{var}(m(v)) \in Eq$, and $\text{dvar}(v) = \text{dvar}(m(v)) \in Eq$. The node identifier variables $\text{var}(v)$ and $\text{var}(m(v))$ can always be matched as requested by $Eq$. If $\text{dvar}(m(v)) = \deg_G(m(v)) \in C$ a match is possible with $\text{dvar}(v) = \deg_G(m(v))$. Otherwise, the match only requires $\text{dvar}(v) = \text{dvar}(m(v))$ which is also possible.

Next consider a node $v \in V_L \setminus V_K$. Then there is a constraint $[\text{type}_L(v)](\text{var}(v), \deg_L(v)) \in C_L$. Again $\text{type}_L(v) = \text{type}_G(m(v))$ holds and we have $[\text{type}_G(m(v))](\text{var}(m(v)), \text{dvar}(m(v))) \in \tilde{G}$. As the applicability of $m$ ensures the gluing condition is satsified we know that $\deg_L(v) = \deg_G(m(v))$ and due to $\text{dvar}(m(v)) = \deg_G(m(v)) \in C$ the match is possible by identifying the node identifier variables $\text{var}(v)$ and $\text{var}(m(v))$.

For an edge $e \in E_L$ we have $[\text{type}_L(\text{src}_L(e))](\text{var}(\text{src}_L(e)), \_)$, $[\text{type}_L(\text{tgt}_L(e))](\text{var}(\text{tgt}_L(e)), \_)$, $[\text{type}_L(e)](\text{var}(e), \text{var}(\text{src}(e)), \text{var}(\text{tgt}(e))) \in C_L$. Due to $m$ being a graph morphism, there exist constraints $[\text{type}_L(\text{src}_L(e))](\text{var}(m(\text{src}_L(e))), \_) \in \tilde{G}$ and $[\text{type}_L(\text{tgt}_L(e))](\text{var}(m(\text{tgt}_L(e))), \_) \in \tilde{G}$. The matchings $\text{var}(\text{src}(e)) = \text{var}(m(\text{src}(e)))$ and $\text{var}(\text{tgt}(e)) = \text{var}(m(\text{tgt}(e)))$ are possible by the previous argumentation on nodes.

There further exists an edge $m(e) \in G$ with
$\text{src}_G(m(e)) = m(\text{src}_L(e)), \text{tgt}_G(m(e)) = m(\text{tgt}_L(e))$ that is represented by
$[\text{type}_L(e)](\text{var}(m(e)), \text{var}(m(\text{src}(e))), \text{var}(m(\text{tgt}(e)))) \in \tilde{G}$ which can be matched to the corresponding constraint in $C_L$ with $\text{var}(e) = \text{var}(m(e))$. □

With the above lemmata it can be shown that applicability directly coincides with respect to states that fully encode a host graph:

**Theorem 4.9 (Applicability For Host Graphs)** *A graph production rule $p = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$ is applicable to a typed host graph $G$ if and only if $\rho(p)$ is applicable to $S = \langle \texttt{chr}(\text{host}, G), \top, \mathcal{V} \rangle$.*

### *4.3 Soundness and Completeness*

In order to argue on the relationship between computations in CHR and the corresponding GTS derivations w.r.t. a defined track morphism we define strong derivations:

**Definition 4.10** [Strong Derivation] A GTS derivation $G \overset{p,m}{\Longrightarrow} G'$ using $p = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$ is *strong with respect to $S \subset V_G$* if $m(L \setminus K) \cap S = \emptyset$.

Def. 4.10 implies that the track morphism is defined $\forall v \in S$. Together with the soundness result below this allows us to consider strong derivations. The basic notion behind these is that the initial state $S$ contains only partial instantiations of degree variables. Then all rule applications correspond to strong derivations with respect to $\mathcal{S}(S)$, and hence, the track morphism is defined $\forall v \in \mathcal{S}(S)$ over all the involved rule applications, because the final state still contains all constraints corresponding to nodes in $\mathcal{S}(S)$.

**Theorem 4.11 (Soundness)** *Let $\rho(p) = (C_L, C_R)$ be applicable to a state $S = \langle \texttt{chr}(\text{keep}, G), C, \mathcal{V} \rangle$ where $\mathcal{G}(S)$ holds with $\eta(p, S) = (\tilde{G}, Eq)$, such that $S \mapsto S'$.*

*Then $p = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$ is applicable to $G$ using the implied match morphism $m$ from Def. 4.5 such that $G \overset{p,m}{\Longrightarrow} G'$ is strong w.r.t. $\mathcal{S}(S)$. Furthermore, $S' \equiv \langle \texttt{chr}(\text{keep}, G'), C', \mathcal{V} \rangle$ and $\mathcal{G}(S')$ holds.*

**Proof.** We have to show that $\mathcal{G}(S')$ holds, i.e. that $S' \equiv \langle \texttt{chr}(\text{keep}, G'), C', \mathcal{V} \rangle$ for the graph $G'$ with $G \overset{p,m}{\Longrightarrow} G'$ using the implied match $m$ from Def. 4.5.

To show that $S'$ is the encoding of the graph $G'$ we show that its construction is analogous to the GTS construction of $G'$:
First the nodes and edges in $m(L)$ get deleted from $G$, but nodes and edges in $m(K)$ are kept. For a node $v \in V_K$ we have $\texttt{chr}(\text{keep}, v) \in C_L$ and $\texttt{chr}(\text{keep}, v'), \text{var}(v) = \text{var}(v') \in C_R$. Hence, the node constraint corresponding to $m(v)$ is removed and a variant of it introduced by the rule application. For a node $v \in V_L \setminus V_K$ instead $\texttt{chr}(\text{host}, v) \in C_L$ and no variant of it is in $C_R$ which results in the removal of that node. Analogously, edges in $m(L) \setminus m(K)$ are removed, while edges in $m(K)$ are kept.

Next for the GTS transformation we add nodes and edges in $R \setminus K$. As for every $x \in R \setminus K$ we have $\mathtt{chr}(\text{host}, x) \in C_R$ the corresponding constraints are also added to $S'$, taking into account the matching $Eq$. As the nodes and edges not matched to $L$ remain unchanged in both systems all nodes and edges of $G'$ are also represented in $S'$. Furthermore, no additional constraints are in $S'$, as all constraints in $C_R$ are directly modeling nodes or edges from $R$.

It is now shown, that $S'$ contains the constraints in $\mathtt{chr}(\text{keep}, G')$. For $\mathcal{G}(S')$ to hold we require a graph $G'$ and equality constraints $C''$, with $\langle \mathtt{chr}(\text{keep}, G'), C' \wedge C'', \emptyset \rangle \equiv \langle \mathtt{chr}(\text{host}, G'), \top, \emptyset \rangle$. The corresponding graph is cleary the graph $G'$ from the GTS rule application, so it remains to investigate if $C'$ can be extended with equality constraints $C''$ such that $\mathcal{G}(S')$ holds.

As all node degrees are instantiated to constants in $\mathtt{chr}(\text{host}, G')$ there are two cases: either the corresponding degree variable is instantiated in $S'$ as well, or it can easily be instantiated to the correct degree through an equality constraint in $C''$. There are three cases for how the degree variable can be instantiated in $S'$ depending on whether the node corresponds to a node $v \in K$, $v \in R \setminus K$, or $v \notin R$. For the last case the node was unaffected by the transformation and as $\mathcal{G}(S)$ holds its degree can be instantiated correctly via $C''$. If $v \in R \setminus K$ then $\mathtt{chr}(\text{host}, v) \in C_R$, i.e. the corresponding node constraint in $S'$ contains the correct degree $\deg_R(v)$ in $C$. Finally, if $v \in K$, then $\text{dvar}(v) = \text{dvar}(v') - n + m \in C_R$ with $v' \in V_G$. Before the application of $p$ there are $n + c$ edges adjacent to $v'$ and after its application $m + c$ edges, hence, the degree of $v'$ changes by $m + c - (n + c) = m - n$ which is correctly represented in $\text{dvar}(v)$. Therefore, the instantiation needs to instantiate $\text{dvar}(v') = \deg_G(v')$ which is possible, as $\mathcal{G}(S)$ holds.

To prove the strongness consider a $v \in \mathcal{S}(S)$ with $v \in V_G$. If $v \notin m(L)$ the node is unaffected by the transformation, hence, we assume $v \in m(L)$. Let $v \in m(L) \setminus m(K)$, then there exists a node $v' \in V_L \setminus V_K$ with $m(v') = v$ with $\text{var}(v) = \text{var}(v') \in Eq$ and $[\text{type}_L(v')](\text{var}(v'), \deg_L(v')) \in C_L$. However, as $\text{dvar}(v) = \deg_G(v) \notin C$ this is a contradiction to the CHR rule application using this match. Therefore, $v \in m(K)$ or $v \notin m(L)$. □

From this soundness result it follows directly that $\mathcal{G}$ is indeed an invariant:

**Corollary 4.12 ($\mathcal{G}$ is an Invariant)** *For CHR programs consisting of rules encoding a GTS the graph invariant $\mathcal{G}$ is an invariant according to Definition 4.1.*

As uninstantiated degree variables inhibit the application of rules that remove the corresponding nodes we can only have completeness if the removed elements are not among the set of strong nodes. When considering a $\mathtt{chr}(\text{host}, G)$ encoding, unrestricted soundness and completeness is given as the following strongness condition is always satisfied.

**Theorem 4.13 (Completeness)** *Let $p = (L \xleftarrow{l} K \xrightarrow{r} R)$ , $G \overset{p,m}{\Longrightarrow} G'$, and let $S = \langle \mathtt{chr}(\text{keep}, G), C, \mathcal{V} \rangle$ be a $\mathcal{G}$-state based on $G$.*

*If $m(V_L \setminus V_K) \cap \mathcal{S}(S) = \emptyset$, then $\rho(p) = (C_L, C_R)$ is applicable to $S$ using the implied match $\eta(p, S)$ from Def. 4.4. Furthermore, for $S \mapsto S'$ using this match*

$S' \equiv \langle \texttt{chr}(\text{keep}, G'), C', \mathcal{V} \rangle$ *and* $\mathcal{G}(S')$ *holds.*

**Proof.** The applicability of $\rho(p) = (C_L, C_R)$ to $S$ using the implied match $\eta(p, S)$ from Def. 4.4 follows from Lemma 4.8.

Therefore, it remains to be shown that the result of applying $\rho(p)$ to $S$ is equivalent to $\langle \texttt{chr}(\text{keep}, G'), C', \mathcal{V} \rangle$. This is done analogously to the proof of Thm. 4.11 by comparing the GTS construction of $G'$ and the operational semantics of the CHR rule application. Similarly, $C' = C \wedge C''$ where $C''$ consists of equality constraints grounding the corresponding degree variables. Note that we assume that the resulting graph $G'$ uses the same nodes as the graph $G$ for nodes which have not been removed instead of explicitly dealing with an isomorphic graph and performing an additional variable renaming.

Finally, the satisfaction of $\mathcal{G}(S')$ follows from Cor. 4.12. □

# 5 Confluence

Both graph transformation systems and constraint handling rules provide the notion of a confluence property. This property guarantees that any derivation made for an initial state results in the same final state no matter which applicable rules are applied. This section introduces the necessary definitions used for GTS and CHR confluence before comparing the two notions. It is shown how automatic observable confluence checking in CHR can be reused to yield a decidable sufficient criterion for confluence of a GTS encoded in CHR.

Note that for the remainder of this section a CHR program always assumes a program consisting only of rules encoding a GTS as explained above. Furthermore, all CHR programs, and therefore graph transformation systems, are assumed to be terminating.

## *5.1 Preliminaries*

**Definition 5.1** [GTS Confluence]A GTS is called *confluent* if, for all typed graph transformations $G \overset{*}{\Longrightarrow} H_1$ and $G \overset{*}{\Longrightarrow} H_2$, there is a typed graph $X$ together with typed graph transformations $H_1 \overset{*}{\Longrightarrow} X$ and $H_2 \overset{*}{\Longrightarrow} X$. *Local confluence* means that this property holds for all pairs of direct typed graph transformations $G \Rightarrow H_1$ and $G \Rightarrow H_2$ [5].

Newman's general result for rewriting systems implies that it is sufficient to consider local confluence for terminating graph transformation systems. To verify local confluence we particularly need to study critical pairs and their joinability, according to the following definition based on [5,8].

**Definition 5.2** [Joinability of Critical GTS Pair] Let $r_1 = (L_1 \overset{l}{\leftarrow} K_1 \overset{r}{\rightarrow} R_1), r_2 = (L_2 \overset{l}{\leftarrow} K_2 \overset{r}{\rightarrow} R_2)$ be two GTS rules. A pair $P_1 \overset{r_1,m_1}{\Longleftarrow} G \overset{r_2,m_2}{\Longrightarrow} P_2$ of direct typed graph transformations is called a *critical GTS pair* if it is parallel dependent, and minimal in the sense that the pair $(m_1, m_2)$ of matches $m_1 : L_1 \to G$ and $m_2 : L_2 \to G$ is jointly surjective.

A pair $P_1 \overset{r_1,m_1}{\Longleftarrow} G \overset{r_2,m_2}{\Longrightarrow} P_2$ of direct typed graph transformations is called *parallel independent* if $m_1(L_1) \cap m_2(L_2) \subseteq m_1(K_1) \cap m_2(K_2)$, otherwise it is called *parallel dependent*.

A critical GTS pair $P_1 \overset{r_1,m_1}{\Longleftarrow} G \overset{r_2,m_2}{\Longrightarrow} P_2$ is called *joinable* if there exists a typed graph $X$ together with typed graph transformations $P_1 \overset{*}{\Longrightarrow} X_1 \simeq X_2 \overset{*}{\Longleftarrow} P_2$. It is *strongly joinable* if there is an isomorphism $f : X_1 \to X_2$ such that for each node $v$, for which $\mathrm{tr}_{G \Rightarrow P_1}(v)$ and $\mathrm{tr}_{G \Rightarrow P_2}(v)$ are defined, the following holds:

(i) $\mathrm{tr}_{G \Rightarrow P_1 \Rightarrow X_1}(v)$ and $\mathrm{tr}_{G \Rightarrow P_2 \Rightarrow X_2}(v)$ are defined and

(ii) $f_V(\mathrm{tr}_{G \Rightarrow P_1 \Rightarrow X_1}(v)) = \mathrm{tr}_{G \Rightarrow P_2 \Rightarrow X_2}(v)$

A similar notion of confluence has been developed for CHR [6]:

**Definition 5.3** [CHR Confluence] A CHR program is called *confluent* if for all states $S, S_1$, and $S_2$: If $S \mapsto^* S_1$ and $S \mapsto^* S_2$, then $S_1$ and $S_2$ are joinable. Two states $S_1$ and $S_2$ are called *joinable* if there exist states $T_1 \equiv T_2$ such that $S_1 \mapsto^* T_1$ and $S_2 \mapsto^* T_2$.

Analogous to a GTS, the confluence property for terminating CHR programs is determined by local confluence which can be checked through critical pairs:

**Definition 5.4** [Joinability of Critical CHR Pair] Let $r_1$ be a simplification rule and $r_2$ be a (not necessarily different) rule whose variables have been renamed apart. Let $H_i \uplus A_i$ be the head, $G_i$ be the guard, and $B_i$ be the body of rule $r_i (i = 1, 2)$, then an *overlap* $\sigma_{\mathcal{CP}}$ of $r_1$ and $r_2$ is $\sigma_{\mathcal{CP}} = \langle H_1 \cup A_1 \cup H_2, (A_1 = A_2) \wedge G_1 \wedge G_2, \mathcal{V} \rangle$, provided $A_1$ and $A_2$ are non-empty conjunctions, $\mathcal{V} = \mathrm{vars}(H_1 \cup A_1 \cup H_2 \cup A_2 \cup G_1 \cup G_2)$ and $CT \models \exists((A_1 = A_2) \wedge G_1 \wedge G_2)$.

Let $S_1 = \langle B_1 \cup H_2, (A_1 = A_2) \wedge G_1 \wedge G_2, \mathcal{V} \rangle$ and $S_2 = \langle B_2 \cup H_1, (A_1 = A_2) \wedge G_1 \wedge G_2, \mathcal{V} \rangle$. Then the tuple $\mathcal{CP} = (S_1, S_2)$ is a *critical CHR pair* of $r_1$ and $r_2$. A critical CHR pair $(S_1, S_2)$ is *joinable* if $S_1$ and $S_2$ are joinable.

### *5.2 Critical Pair Properties*

After defining the different notions of confluence we now further investigate the difference between critical GTS pairs and critical CHR pairs for CHR programs encoding a GTS. The following lemma shows that there exists a corresponding CHR overlap for each critical GTS pair. Therefore, by examining the overlaps and using the previous soundness result we can transfer joinability results to the critical GTS pair.

**Lemma 5.5 (Overlap for Critical GTS Pair)** *If $P_1 \overset{r_1,m_1}{\Longleftarrow} G \overset{r_2,m_2}{\Longrightarrow} P_2$ is a critical GTS pair, then there exists an overlap $\sigma_{\mathcal{CP}}$ of $\rho(r_1) = (C_L^1, C_R^1)$ and $\rho(r_2) = (C_L^2, C_R^2)$ which is a $\mathcal{G}$-state based on $G$ and a critical CHR pair $(S_1, S_2)$ such that $S_1$ is a $\mathcal{G}$-state based on $P_1$ and $S_2$ is a $\mathcal{G}$-state based on $P_2$.*

**Proof.** Let $M = m_1(L_1) \cap m_2(L_2)$. We then define the following sets of constraints

with $k(x) = \texttt{chr}(\text{keep}, x)$:

$$\begin{array}{lll} H_1 = & \{k(x) \mid x \in L_1 \wedge m_1(x) \notin M\}, & H_2 = \{k(x) \mid x \in L_2 \wedge m_2(x) \notin M\} \\ A_1 = & \{k(x) \mid x \in L_1 \wedge m_1(x) \in M\}, & A_2 = \{k(x) \mid x \in L_2 \wedge m_2(x) \in M\} \\ C_1 = & \{\text{dvar}(v) = \deg_{L_1}(v) \mid v \in V_{L_1} \setminus V_{K_1}\} & \\ C_2 = & \{\text{dvar}(v) = \deg_{L_2}(v) \mid v \in V_{L_2} \setminus V_{K_2}\} & \end{array}$$

Let $\mathcal{V} = \text{vars}(H_1 \cup H_2 \cup A_1 \cup A_2)$. By combining $H_1$ and $C_1$ to a state $S = \langle H_1, C_1, \mathcal{V}\rangle$ we know that $S \equiv S' = \langle \{\texttt{chr}(\text{keep}, x) \mid x \in K_1 \wedge m_1(x) \notin M\} \cup \{\texttt{chr}(\text{host}, x) \mid x \in L_1 \setminus K_1 \wedge m_1(x) \notin M\}, \top, \mathcal{V}\rangle = \langle H_1', \top, \mathcal{V}\rangle$. Similarly, $\langle A_1, C_1, \mathcal{V}\rangle \equiv \langle \{\texttt{chr}(\text{keep}, x) \mid x \in K_1 \wedge m_1(x) \in M\} \cup \{\texttt{chr}(\text{host}, x) \mid x \in L_1 \setminus K_1 \wedge m_1(x) \in M\}, \top, \mathcal{V}\rangle = \langle A_1', \top, \mathcal{V}\rangle$. Analogously, we define $H_2'$ and $A_2'$.

It follows from the definition of $C_L$, that $H_1' \cup A_1'$ matches $C_L^1$ and analogously, $H_2' \cup A_2'$ matches $C_L^2$. As $M \neq \emptyset$ $A_1'$ and $A_2'$ are non-empty. To investigate if $CT \models \exists(A_1' = A_2')$ we take a closer look at the equality constraints imposed by $A_1' = A_2'$:

$$\begin{array}{rl} (A_1' = A_2') & = \{\text{var}(v_1) = \text{var}(v_2) \mid v_1 \in V_{L_1} \wedge v_2 \in V_{L_2}, m_1(v_1) = m_2(v_2)\} \\ \cup & \{\text{dvar}(v_1) = \text{dvar}(v_2) \mid v_1 \in V_{K_1} \wedge v_2 \in V_{K_2} \wedge m_1(v_1) = m_2(v_2)\} \\ \cup & \{\text{dvar}(v_1) = \deg_{L_2}(v_2) \mid v_1 \in V_{K_1} \wedge v_2 \in V_{L_2} \setminus V_{K_2} \wedge m_1(v_1) = m_2(v_2)\} \\ \cup & \{\text{dvar}(v_2) = \deg_{L_1}(v_1) \mid v_1 \in V_{L_1} \setminus V_{K_1} \wedge v_2 \in V_{K_2} \wedge m_1(v_1) = m_2(v_2)\} \\ \cup & \{\text{var}(e_1) = \text{var}(e_2) \mid e_1 \in E_{K_1} \wedge e_2 \in E_{K_2} \wedge m_1(e_1) = m_2(e_2)\} \end{array}$$

The above equality constraints can easily be satisfied and hence the only remaining problematic case is when two node constraints with constant degrees are overlapped. However, the degree of $m_1(v_1) = m_2(v_2)$ equals the degree of $v_1$ and the degree of $v_2$ due to the gluing condition being satisfied, such that this case can only occur with equal constant degrees.

Hence, $\sigma_{\mathcal{CP}} = \langle H_1' \cup A_1' \cup H_2', A_1' = A_2', \mathcal{V}\rangle$ is an overlap of $\rho(r1)$ and $\rho(r_2)$ with the critical CHR pair $(\langle C_R^1 \cup H_2', A_1' = A_2', \mathcal{V}\rangle, \langle C_R^2 \cup H_1', A_1' = A_2', \mathcal{V}\rangle$. □

If we try to directly transfer the confluence property of a GTS to the corresponding CHR program, we cannot succeed however, as in general there are too many critical CHR pairs that could cause the CHR program to be non-confluent. The following example provides a rule, which only has one critical GTS pair, but for which the corresponding CHR rule has three critical CHR pairs.

**Example 5.6** Consider a graph production rule for removing a loop from a node and its corresponding CHR rule:

$R@\,\text{node}(N, D), \text{edge}(E, N, N) \Leftrightarrow \text{node}(N, D'), D' = D - 2$

For investigating confluence one must overlap this rule with itself which yields the following three CHR overlap states:

(i) $\langle \text{node}(N, D) \cup \text{edge}(E, N, N) \cup \text{edge}(E', N', N'), N = N', \mathcal{V}\rangle$

(ii) $\langle \text{node}(N, D) \cup \text{node}(N', D') \cup \text{edge}(E, N, N), N = N', \mathcal{V}\rangle$

(iii) $\langle \text{node}(N, D) \cup \text{edge}(E, N, N), \top, \mathcal{V}\rangle$

State (i) is not critical, because the corresponding pair of graph transformations is parallel independent, and hence, directly joinable by applying the rule again. State (ii) is an invalid state as it has multiple encodings of the same node and state (iii) is the encoding of the corresponding critical pair for the graph production rule.

As we want to rule out invalid states, we use the following notion of observable confluence presented in [4]. It is based on restricting confluence investigations to states that satisfy an invariant. Based on these invariants, observable confluence (or $\mathcal{I}$-confluence) is defined as follows:

**Definition 5.7** [Observable Confluence] A CHR program $P$ is $\mathcal{I}$-confluent with respect to invariant $\mathcal{I}$ if the following holds for all states $S_0, S_1$, and $S_2$ where $\mathcal{I}(S_0)$ holds: If $S_0 \rightarrow^* S_1$ and $S_0 \rightarrow^* S_2$ then $S_1$ and $S_2$ are joinable.

In order to use the graph invariant $\mathcal{G}$ for the notion of observable confluence, we have to investigate the properties of this invariant. We introduce the following definitions from [4]. As overlap states themselves may not satisfy the invariant we have to examine all possible extensions that satisfy it [4].

**Definition 5.8** [Extension, Valid Extension] A state $\sigma = \langle G, B, \mathcal{V}\rangle$ can be *extended* by another state $\sigma_e = \langle G_e, B_e, \mathcal{V}_e\rangle$ as follows: $\sigma \lhd \sigma_e = \langle G \uplus G_e, B \wedge B_e, \mathcal{V}_e\rangle$. We say that $\sigma_e$ is an *extension* of $\sigma$. A *valid extension* $\sigma_e = \langle G_e, B_e, \mathcal{V}_e\rangle$ of a state $\sigma = \langle G, B, \mathcal{V}\rangle$ is an extension such that $v \in vars(G \cup B) \wedge v \notin \mathcal{V} \Rightarrow v \notin vars(G_e \cup B_e \cup \mathcal{V}_e)$.

In the following we discuss states that do not satisfy an invariant $\mathcal{I}$ and extensions of those states, such that the result satisfies the invariant $\mathcal{I}$. In this context, we refer to $\Sigma_e^{\mathcal{I}}(\sigma)$ as the set of all extensions $\sigma_e$ of the state $\sigma$ such that $\mathcal{I}(\sigma \lhd \sigma_e)$ holds.

To reduce the number of extensions that have to be investigated only minimal extensions w.r.t. a partial order $\prec_\sigma$ on extensions [4] are considered. $\mathcal{M}_e^{\mathcal{I}}(\sigma)$ denotes the set of these minimal extensions of a state $\sigma$ and is used in the following decision criterion of $\mathcal{I}$-local-confluence.

**Lemma 5.9 (Deciding $\mathcal{I}$-Local-Confluence [4])** *Let $\prec_{\sigma_{\mathcal{CP}}}$ be well-founded for all overlaps $\sigma_{\mathcal{CP}}$, then: $P$ is $\mathcal{I}$-local-confluent iff for all critical pairs $\mathcal{CP} = (\sigma_1, \sigma_2)$ with overlap $\sigma_{\mathcal{CP}}$, and for all $\sigma_e \in \mathcal{M}_e^{\mathcal{I}}(\sigma_{\mathcal{CP}})$, we have that $(\sigma_1 \lhd \sigma_e, \sigma_2 \lhd \sigma_e)$ is joinable.*

Although, in our programs built-in constraints + and − occur, we can consider $\prec_{\sigma_{\mathcal{CP}}}$ well-founded, as $\sigma_\emptyset$ is always smaller than any other extension. The following discussion shows that either $\mathcal{M}_e^{\mathcal{G}}(\sigma_{\mathcal{CP}}) = \{\sigma_\emptyset\}$ or $\Sigma_e^{\mathcal{G}}(\sigma_{\mathcal{CP}}) = \mathcal{M}_e^{\mathcal{G}}(\sigma_{\mathcal{CP}}) = \emptyset$. This means, that for all elements $\sigma_e \in \Sigma_e^{\mathcal{G}}(\sigma_{\mathcal{CP}})$ we have $\sigma_\emptyset \preceq_{\sigma_{\mathcal{CP}}} \sigma_e$, and hence, $\prec_{\sigma_{\mathcal{CP}}}$ is well-founded. Whether $\sigma_\emptyset$ is the minimal element depends solely on $\mathcal{G}(\sigma_{\mathcal{CP}})$ holding as the following lemma shows.

**Lemma 5.10 (No Minimal Elements)** *If $\mathcal{G}(\sigma_{\mathcal{CP}})$ is violated for an overlap $\sigma_{\mathcal{CP}}$ then no extension $\sigma_e$ exists such that $\mathcal{G}(\sigma_{\mathcal{CP}} \lhd \sigma_e)$ is satisfied, i.e. $\Sigma_e^{\mathcal{G}}(\sigma_{\mathcal{CP}}) = \mathcal{M}_e^{\mathcal{G}}(\sigma_{\mathcal{CP}}) = \emptyset$.*

**Proof.** We proof this by a structural analysis of the overlap which gives the different possibilities for $\mathcal{G}(\sigma_{\mathcal{CP}})$ to be violated. W.l.o.g. the overlap stems from the two rules $\rho(r_1) = (C_L^1, C_R^1)$ and $\rho(r_2) = (C_L^2, C_R^2)$ with the corresponding rule graphs $L_1, L_2, K_1, K_2, R_1$, and $R_2$.

First consider a node being overlapped. Let $[\text{type}_{L_1}(v_1)](\text{var}(v_1), D_1) \in C_L^1$ and $[\text{type}_{L_2}(v_2)](\text{var}(v_2), D_2) \in C_L^2$ be overlapped with $\text{type}_{L_1}(v_1) = \text{type}_{L_2}(v_2)$. The equality constraint $\text{var}(v_1) = \text{var}(v_2) \in \sigma_{\mathcal{CP}}$ resembles the merging of the two graph nodes $v_1$ and $v_2$. However, for the degree equalities different possibilities exist:

- $D_1$ and $D_2$ are constants: Then $D_1 = D_2 = \deg_{L_1}(v_1) = \deg_{L_2}(v_2) = k$, as the overlap is impossible otherwise. Then $\sigma_{\mathcal{CP}}$ contains only one constraint $[\text{type}_{L_1}(v_1)](\text{var}(v_1), \deg_{L_1}(v_1))$. As in $L_1$ and $L_2$ the nodes each have $k$ adjacent edges, all constraints corresponding to adjacent edges in both rule graphs have to be contained in the overlap as well. If at least one such constraint is not part of the overlap then $\sigma_{\mathcal{CP}}$ contains more than $k$ constraints corresponding to edges adjacent to $v_1 = v_2$. As the degree for the node is a constant it cannot be changed by any extension and the additional edge constraints cannot be removed either. Therefore, no extension can correct the degree inconsistency in such a case.
- $D_1$ and $D_2$ are variable: In this case the overlap is possible without any problems. Depending on the number of overlapped adjacent edge constraints the degree variables can always be instantiated with the correct degree, thus satisfying the invariant $\mathcal{G}$.
- w.l.o.g. $D_1 = k$ and $D_2$ is a variable: this means $D_2 = k \in \sigma_{\mathcal{CP}}$, therefore, all edge constraints of $C_L^2$ of edges adjacent to $v_2$ have to be overlapped with edge constraints of $C_L^1$ corresponding to edges adjacent to $v_1$. If there is such an edge constraint from $C_L^2$ which is not contained in the overlap, then $\sigma_{\mathcal{CP}}$ contains more than $k$ edge constraints corresponding to edges adjacent to $v_1$. Again the degree of $v_1$ is specified as the constant $k$ in $\sigma_{\mathcal{CP}}$, and thus, an extension cannot correct this degree inconsistency. If however, all these edge constraints are contained in the overlap $\mathcal{G}$ is satisified again, as there are exactly $k$ such edge constraints coming from $C_L^1$.

Finally, consider an edge being overlapped:
Let $[\text{type}_{L_1}](\text{var}(e_1), \text{var}(\text{src}(e_1)), \text{var}(\text{tgt}(e_1))) \in C_L^1$ and
$[\text{type}_{L_2}](\text{var}(e_2), \text{var}(\text{src}(e_2)), \text{var}(\text{tgt}(e_2))) \in C_L^2$, then
$\text{var}(e_1) = \text{var}(e_2) \wedge \text{var}(\text{src}(e_1)) = \text{var}(\text{src}(e_2)) \wedge \text{var}(\text{tgt}(e_1)) = \text{var}(\text{tgt}(e_2)) \in \sigma_{\mathcal{CP}}$. By Def. 3.4 we have constraints $[\text{type}_{L_1}(\text{src}(e_1))](\text{var}(\text{src}(e_1)), \_) \in C_L^1$ and $[\text{type}_{L_2}(\text{src}(e_2))](\text{var}(\text{src}(e_2)), \_) \in C_L^2$. If these two constraints are not part of the overlap the corresponding equality constraint in $\sigma_{\mathcal{CP}}$ results in a single graph node being represented by two constraints. This is a violation of $\mathcal{G}$, as $\texttt{chr}(\text{host}, G)$ contains exactly one constraint for each node. This violation cannot be fixed by an extension, as the conflicting additional node constraint cannot be removed. Analogously, the two node constraints corresponding to $\text{tgt}(e_1)$ and $\text{tgt}(e_2)$ have to be contained in the overlap.

Therefore, an overlap $\sigma_{\mathcal{CP}}$ which violates the invariant $\mathcal{G}$ has to violate it due to one of the above reasons for which it cannot be extended by an extension $\sigma_e$ such that $\mathcal{G}(\sigma_{\mathcal{CP}} \lhd \sigma_e)$ is satisfied. □

Combining these two results yields the criterion in Cor. 5.11 for deciding $\mathcal{G}$-local-confluence. Note that this decision criterion is essentially the same criterion as used for traditional local confluence, except that the invariant $\mathcal{G}$ restricts the set of investigated overlaps.

**Corollary 5.11 (Deciding $\mathcal{G}$-Local-Confluence)** *$P$ is $\mathcal{G}$-local-confluent if and only if for all critical pairs $\mathcal{CP} = (\sigma_1, \sigma_2)$ with overlap $\sigma_{\mathcal{CP}}$, for which $\mathcal{G}(\sigma_{\mathcal{CP}})$ holds, $\mathcal{CP}$ is joinable.*

Next we transfer joinability of critical CHR pairs to strong joinability in GTS:

**Lemma 5.12 ($\mathcal{G}$-Confluence Implies Strong Joinability)** *Given a CHR program for a terminating GTS that is $\mathcal{G}$-confluent, then all critical GTS pairs are strongly joinable.*

**Proof.** Let $P_1 \overset{r_1,m_1}{\Longleftarrow} G \overset{r_2,m_2}{\Longrightarrow} P_2$ be a critical GTS pair. Let $r_i = (L_i \leftarrow K_i \rightarrow R_i)$ and $\rho(r_i) = (C_L^i, C_R^i)$ for $i = 1, 2$.

By Lemma 5.5 there exists an overlap $\sigma_{\mathcal{CP}}$ which is a $\mathcal{G}$-state based on $G$. As the critical pair $(S_1, S_2)$ created by the overlap $\sigma_{\mathcal{CP}}$ is joinable we have the computations $\sigma_{\mathcal{CP}} \mapsto S_1 \mapsto^* T_1$ and $\sigma_{\mathcal{CP}} \mapsto S_2 \mapsto^* T_2$ with $T_1 \equiv T_2$. From Thm. 4.11 we know that there exist corresponding GTS transformations $G \overset{r_1,m_1}{\Longrightarrow} P_1 \Longrightarrow^* X_1 \simeq X_2 {}^* \Longleftarrow P_2 \overset{r_2,m_2}{\Longleftarrow} G$. The isomorphism between $X_1$ and $X_2$ follows from $T_1 \equiv T_2$. Hence, the critical GTS pair is joinable.

To see that it is strongly joinable consider the set $\mathcal{S}(\sigma_{\mathcal{CP}})$. Every node $v$ for which $\mathrm{tr}_{G \Rightarrow P_1}(v)$ and $\mathrm{tr}_{G \Rightarrow P_2}(v)$ are defined is a node which is not deleted by either $r_1$ or $r_2$. As $m_1$ and $m_2$ are jointly surjective w.l.o.g. there exists a node $v' \in V_{L_1}$ of rule $r_1$ with $m(v') = v$. As the node is not removed we know $v' \in V_{K_1}$, and therefore, $[\mathrm{type}_{K_1}(v')](\mathrm{var}(v'), \mathrm{dvar}(v')) \in C_L^1$. Either the node is not part of the overlap, or if it is overlapped with a node $v'' \in V_{L_2}$ such that $m(v') = m(v'')$, then we also know that $v'' \in V_{K_2}$ due to the defined track morphism. Therefore, we always have the node constraint $[\mathrm{type}_{K_1}(v')](\mathrm{var}(v), \mathrm{dvar}(v)) \in \sigma_{\mathcal{CP}}$ and $v \in \mathcal{S}(\sigma_{\mathcal{CP}})$. As this node cannot be removed during the transformation a variant of this constraint with adjusted degree is also present in $T_1$ and $T_2$. These two variant constraints are uniquely determined, as $\mathrm{var}(v) \in \mathcal{V}$ by Def. 5.4, and hence, they both have to use $\mathrm{var}(v)$ for the node identifier variable. This means we still have to show for such a node $v$ that the two conditions from Def. 5.2 are satisfied:

(i) $\mathrm{tr}_{G \Rightarrow P_1 \Rightarrow X_1}(v)$ and $\mathrm{tr}_{G \Rightarrow P_2 \Rightarrow X_2}(v)$ are defined:
By Thm. 4.11 we know that the GTS transformations are strong w.r.t. $\mathcal{S}(\sigma_{\mathcal{CP}})$. As $v \in \mathcal{S}(\sigma_{\mathcal{CP}})$ this implies $v \in m(K) \vee v \notin m(L)$ for each of the applied rules, i.e. the node remains during the transformation and hence the track morphisms are defined as in Def. 2.2.

(ii) $f_V(\mathrm{tr}_{G \Rightarrow P_1 \Rightarrow X_1}(v)) = \mathrm{tr}_{G \Rightarrow P_2 \Rightarrow X_2}(v)$:

An isomorphism $f'$ between $T_1$ and $T_2$ exists, because $T_1 \equiv T_2$. Consider the constraints in $T_1$ and $T_2$ which are the encoding of node $v$ in $\sigma_{\mathcal{CP}}$ and let them use the degree variables $\mathrm{dvar}(v_1)$ and $\mathrm{dvar}(v_2)$ (with the corresponding chain of constraints $\mathrm{dvar}(v_i) = \mathrm{dvar}(v_i') - n' + m' = \ldots = \mathrm{dvar}(v) - n + m$ for $i = 1, 2$ that have been accumulated during the computation). Then there exist corresponding nodes $\mathrm{tr}_{G \Rightarrow P_1 \Rightarrow X_1}(v) = v_1 \in V_{X_1}$ and $\mathrm{tr}_{G \Rightarrow P_2 \Rightarrow X_2}(v) = v_2 \in V_{X_2}$ and the isomorphism $f'$ between $T_1$ and $T_2$, which equalizes $\mathrm{var}(v_1)$ and $\mathrm{var}(v_2)$, implies an isomorphism $f$ with $f_V(v_1) = v_2$.

□

The reverse direction holds as well, i.e. strong joinability of critical GTS pairs implies that the corresponding CHR-GTS program is $\mathcal{G}$-confluent.

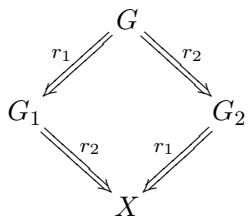
**Lemma 5.13 (Strong Joinability Implies $\mathcal{G}$-Confluence)** *If all critical GTS pairs of a terminating GTS are strongly joinable, then the corresponding CHR program is $\mathcal{G}$-confluent.*

**Proof.** Consider an overlap $\sigma_{\mathcal{CP}}$ for the critical CHR pair $(\sigma_1, \sigma_2)$. W.l.o.g. $\mathcal{G}(\sigma_{\mathcal{CP}})$ holds according to Cor. 5.11. Therefore, $\sigma_{\mathcal{CP}}$ is a $\mathcal{G}$-state based on $G$ and $\sigma_1, \sigma_2$ correspond to graphs $G_1, G_2$. Consider now $G_1 \overset{r_1,m_1}{\Longleftarrow} G \overset{r_2,m_2}{\Longrightarrow} G_2$.

We now show, that either the critical CHR pair is easily joinable, or it corresponds to a critical GTS pair and can thus be joined, because all critical GTS pairs are strongly joinable.

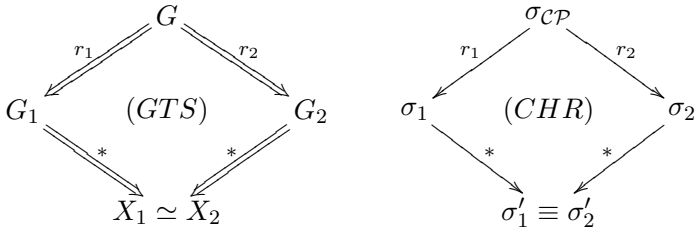
First, we want to point out that $G$ is minimal by the definition of the CHR overlap, i.e. every occuring node and edge is part of a match, hence, $m_1$ and $m_2$ are jointly surjective.

Next, we distinguish two cases: First, let $G_1 \overset{r_1,m_1}{\Longleftarrow} G \overset{r_2,m_2}{\Longrightarrow} G_2$ be parallel independent. Therefore, the second rule can be applied after the first, because none of the required nodes or edges has been removed. The following diagram depicts this situation:




By Thm. 4.11 we can apply the corresponding rules to $\sigma_{\mathcal{CP}}$ in order to join the critical CHR pair, because $\mathcal{S}(\sigma_{\mathcal{CP}})$ contains only nodes and edges not deleted by $r_1$ and $r_2$.

Secondly, let $G_1 \overset{r_1,m_1}{\Longleftarrow} G \overset{r_2,m_2}{\Longrightarrow} G_2$ be parallel dependent. It follows that $m(L_1) \cap m(L_2) \not\subseteq m(K_1) \cap m(K_2)$. However, this is now a critical GTS pair, and hence, strongly joinable as depicted on the left of the following diagram:




The right part of the diagram shows the situation for the critical CHR pair which is joinable by Thm. 4.13. This is possible, because $\forall v \in \mathcal{S}(\sigma_{\mathcal{CP}})$ we know that $\mathrm{tr}_{G \Rightarrow G_1}(v)$ and $\mathrm{tr}_{G \Rightarrow G_2}(v)$ are defined, thus by Def. 5.2, $v$ is never removed and still present in $X_1$ and $X_2$. Finally, the morphism implied by $X_1 \simeq X_2$ gives us $\sigma'_1 \equiv \sigma'_2$.

Therefore, for all overlaps $\sigma_{\mathcal{CP}}$ with $\mathcal{G}(\sigma_{\mathcal{CP}})$ holding we know that the corresponding critical CHR pair is joinable, and hence, by Cor. 5.11 that the CHR program is $\mathcal{G}$-local-confluent. As it is terminating it is $\mathcal{G}$-confluent as well. □

**Theorem 5.14 (Strong Joinability iff $\mathcal{G}$-Confluence)** *For a terminating GTS all critical GTS pairs are strongly joinable if and only if the corresponding CHR program is $\mathcal{G}$-confluent.*

In practical terms Theorem 5.14 effectively means that the automatic confluence check for terminating CHR programs [2,6] can be reused to prove confluence of a terminating GTS encoded as a CHR program. Due to the earlier results presented in this section we can apply the standard confluence checker only to those overlaps satisfying the invariant $\mathcal{G}$. The possible causes for an overlap to not satisfy $\mathcal{G}$ are duplicate node constraints or inconsistent degrees which can easily be checked. If all critical CHR pairs stemming from these overlaps are joinable we know by Cor. 5.11 that the CHR program is $\mathcal{G}$-confluent, and hence by Theorem 5.14, that the GTS is confluent. As no modification is needed for the confluence checker itself this means that by a restriction of inputs to the confluence checker we can decide $\mathcal{G}$-confluence and in turn get a sufficient criterion for GTS confluence for free.

## 6 Conclusion

In [9] we have shown that constraint handling rules (CHR) provide an elegant way for embedding graph transformation systems (GTS). The resulting rules are concise and directly related to the corresponding graph production rules. We presented a generalization of this encoding. It allows to model strong derivations that are used to analyze strong joinability.

The combination of our work with the research on observable confluence [4] resulted in a direct application of the CHR confluence check to decide $\mathcal{G}$-confluence. Invalid overlaps introduced by the CHR encoding of a GTS can elegantly be handled by considering $\mathcal{G}$-confluence which reduces the confluence analysis to the essential overlaps that yield strong joinability of critical GTS pairs.

The connection between CHR and GTS provides room for further research. This work only considers typed graphs, but could be extendend to support typed

*attributed* graphs as well. As our generalized encoding allows computations on partially defined graphs this allows considering derivations as being applicable to the corresponding set of fully defined graphs.

Furthermore, it seems possible to transfer other results from CHR to GTS and vice versa. The approaches used for termination analysis of CHR [7] and GTS [5] seem to be distinct, such that both may profit from applying the approaches from the other formalism. Similarly, CHR provides a strong result on operational equivalence [1] that may provide a decidable criterion for equivalence of embedded graph transformation systems.

Finally, the notion of graphs with interfaces appears to be similar to the notion of global variables in CHR. It might, therefore, be possible to identify a subset of GTS that uses graphs with interfaces in which confluence of a terminating GTS is decidable.

## References

[1] Abdennadher, S. and T. Frühwirth, *Operational equivalence of CHR programs and constraints*, in: J. Jaffar, editor, *Principles and Practice of Constraint Programming, CP 1999*, Lecture Notes in Computer Science **1713** (1999), pp. 43–57.

[2] Abdennadher, S., T. Frühwirth and H. Meuss, *Confluence and semantics of constraint simplification rules*, Constraints **4** (1999), pp. 133–165.

[3] Bakewell, A., D. Plump and C. Runciman, *Specifying pointer structures by graph reduction.*, in: J. L. Pfaltz, M. Nagl and B. Böhlen, editors, *Applications of Graph Transformations with Industrial Relevance, Second International Workshop, AGTIVE 2003, Revised Selected and Invited Papers*, Lecture Notes in Computer Science **3062** (2003), pp. 30–44.

[4] Duck, G. J., P. J. Stuckey and M. Sulzmann, *Observable confluence for constraint handling rules*, in: V. Dahl and I. Niemelä, editors, *Logic Programming, 23rd International Conference, ICLP 2007*, Lecture Notes in Computer Science **4670** (2007), pp. 224–239.

[5] Ehrig, H., K. Ehrig, U. Prange and G. Taentzer, "Fundamentals of Algebraic Graph Transformation," Springer-Verlag, 2006.

[6] Frühwirth, T., "Constraint Handling Rules," Cambridge University Press, 2009, to appear.

[7] Pilozzi, P. and D. De Schreye, *Termination analysis of CHR revisited*, in: T. Schrijvers, F. Raiser and T. Frühwirth, editors, *Constraint Handling Rules, 5th Workshop, CHR 2008*, Hagenberg, Austria, 2008, pp. 35–50.

[8] Plump, D., *Confluence of graph transformation revisited*, in: A. Middeldorp, V. van Oostrom, F. van Raamsdonk and R. C. de Vrijer, editors, *Processes, Terms and Cycles*, Lecture Notes in Computer Science **3838** (2005), pp. 280–308.

[9] Raiser, F., *Graph Transformation Systems in CHR*, in: V. Dahl and I. Niemelä, editors, *Logic Programming, 23rd International Conference, ICLP 2007*, Lecture Notes in Computer Science **4670** (2007), pp. 240–254.

[10] Raiser, F. and T. Frühwirth, *Strong joinability analysis for graph transformation systems in CHR*, in: *5th International Workshop on Computing with Terms and Graphs, TERMGRAPH'09, Pre-Proceedings*, 2009.