Observable Confluence for Constraint Handling Rules

Gregory J. Duck[1], Peter J. Stuckey[1], and Martin Sulzmann[2]

[1] NICTA Victoria Laboratory
Department of Computer Science and Software Engineering
University of Melbourne, 3010, AUSTRALIA
{gjd,pjs}@cs.mu.oz.au
[2] School of Computing, National University of Singapore
S16 Level 5, 3 Science Drive 2, Singapore 117543
sulzmann@comp.nus.edu.sg

Abstract. Constraint Handling Rules (CHRs) are a powerful rule based language for specifying constraint solvers. Critical for any rule based language is the notion of confluence, and for terminating CHRs there is a decidable test for confluence. But many CHR programs that in practice are confluent fail this confluence test. The problem is that the states that illustrate non-confluence are not reachable in practice. In this paper we introduce the notion of observable confluence, a weaker notion of confluence which takes into account whether states are observable. We show for an important class of non-confluent programs arising from Haskell type class programs with functional dependencies, that they are observable confluent.

1 Introduction

Constraint Handling Rules [3] (CHRs) are a powerful rule based language for specifying constraint solvers. Constraint handling rules operate on a global multiset (conjunction) of constraints. A constraint handling rule defines a rewriting from one multiset of constraints to another. There are two kinds of rewriting rules: simplification rules which replace one multiset of constraints by another, and propagation rules which when seeing a multiset of constraints matching some condition, add some more constraints to the global multiset.

Example 1. Consider the following CHR program:[3]

```
r1 @ f(int,bool,float)    <=> true.
r2 @ f(A,B1,C), f(A,B2,D) ==> B1 = B2.
r3 @ f(int,B,C)           ==> B = bool.
```

The first rule has name `r1`. It is a simplification rule. Whenever we see the constraint `f(int,bool,float)` in the global multiset we can replace it by `true` (representing the empty multiset). The second rule (named `r2`) is a propagation

[3] CHRs follow Prolog like notation, where identifiers starting with a lower case letter indicate predicates and function symbols, and identifiers starting with upper case letters indicate variables.

rule: whenever there exists two `f/3` constraints in the global multiset that share the same first argument, we can add the constraint that the second arguments must be identical. The third rule (`r3`) is another propagation rule, whenever we see a `f` constraint with first argument `int` we can enforce that the second argument is `bool`.

The program above results from the translation of type class constraints in Haskell [7] involving functional dependencies [5] to CHRs [4, 8]. Here is the original type class program.

```
class F x y z | x -> y
instance F Int Bool Float
```

Rule `r1` encodes the instance for `F`, that is we can prove their is an instance for `Int Bool Float`. Rule `r2` encodes the functional dependency, it requires that for any two `F` constraints if the first argument is the same, the second must also be the same (since it is functionally defined by the first). Rule `r3` encodes the combination of the instance rule and the functional dependency.

Unlike other rewriting systems, constraint handling rules allow propagation rules. Note that propagation rules do not delete anything from the global multiset that makes them eligible to be executed, they simply add new constraints. For this reason they would seem to trivially lead to non-termination. This is overcome in constraint handling rules by keeping a *history* of propagation rule firings, and preventing a second firing of a propagation rule on the same set of constraints.

A critical issue for any rule based language is the notion of confluence. Confluence enforces that each different possible rewriting sequence leads eventually to the same result. Confluent programs have a deterministic behaviour in terms of an input goal will always lead to the same answer. For terminating CHR programs there is a decidable test for confluence [1]. Unfortunately there are many (terminating) programs which are confluent in practice, but fail to pass the test.

The reason for this arises from the use of propagation rules and histories. The confluence test examines critical states, which are minimal states where two different rule firings are possible. In order to be minimal states, the propagation history is assumed to be as strong as possible, that is, it disallows any propagation rules that could possible fire except the two rules used to generate the critical state itself.

Example 2. Consider the CHR program of Example 1, and a critical state arising from rules `r1` and `r2`.

$$f(int, bool, float), f(int, B2, D)$$

Both rules `r1` and `r2` are applicable to the state and to ensure that it is minimal, we assume a propagation history which disallows the use of `r3` on either of the constraints appearing in the state.

The CHR program of Example 1 is not confluent, because this critical state can lead to 2 different results, depending on which of `r1` and `r2` is first used. This is illustrated by the derivations

$$\begin{array}{ll} & f(int, bool, float), f(int, B2, D) \\ \rightarrowtail_{r1} & f(int, B2, D) \end{array}$$

and

$$\begin{array}{rl} & f(int, bool, float), f(int, B2, D) \\ \rightarrowtail_{r2} & f(int, bool, float), f(int, bool, D), B2 = bool \\ \rightarrowtail_{r1} & f(int, bool, D), B2 = bool \end{array}$$

Two different states result. In one we know that $B2 = bool$ in the other we do not. Note that we cannot apply the rule `r3` to the state $f(int, B2, D)$ because the propagation history disallows this.

The above example illustrates that the program P of Example 1 is non-confluent. But in practice it is not possible to write a goal G which leads to two different answers using P. The reason is that goals always begin with an empty history. The critical state used above to illustrate non-confluence cannot actually occur in practice.

Example 3. Consider the state

$$f(int, bool, float), f(int, B2, D)$$

assuming that propagation history prevent `r3` firing on the second constraint. Thus `r3` must have fired already on the second constraint. This cannot have occurred since this would add the constraint $B2 = bool$ to the store, and it does not appear.

CHRs resulting from the translation of type class constraints in Haskell involving functional dependencies are an important class of programs since the soundness and completeness of type inference for Haskell with type classes and functional dependencies depends on their confluent behaviour [4, 8].

In this paper, we make the following contributions:

- We introduce the notion of *observable confluence* which captures the notion of programs where all the observable behaviour (derivations for goals with originally empty propagation histories) is confluent (Section 2.3).
- We show that for a class of CHRs arising from Haskell type class programs with functional dependencies the current notion of confluence is too limiting (Section 3).
- We provide for a general observable confluence result based on a operational correspondence condition between two sets of CHRs (Section 4).
- We verify that a certain class of non-confluent CHRs resulting from type class programs with functional dependencies are in fact observable confluent (Section 5).

In Section 2 we provide background material on CHRs and also introduce the notion of observable confluence. We conclude in Section 6.

2 Preliminaries

In this paper we study confluence under the *theoretical operational semantics* ω_t of CHRs [2]. The theoretical semantics is equivalent to the original semantics defined in [1] except that

– the original semantics treats simpagation rules as shorthand for a (logically) equivalent simplification rule; and
– the treatment of propagation histories is different.

First we define numbered constraints.

Definition 1 (Numbered Constraints). *A* numbered constraint *is a constraint c paired with an integer i. We write $c\#i$ to indicate a numbered constraint.* □

Sometimes we refer to i as the *identifier* (or simply ID) of the numbered constraint. This numbering serves to differentiate among copies of the same constraint.

Now we define an *execution state*, as follows.

Definition 2 (Execution State). *An* execution state *is a tuple of the form $\langle G, S, B, T\rangle_n^{\mathcal{V}}$ where G is a multiset of constraints, S is a set of numbered constraints, B is a conjunction of built-in constraints, T is a set of sequences of integers, $\mathcal{V}$ is the set of variables and n is an integer. Throughout this paper we use symbol 'σ' to represent an execution state.* □

We call G the *goal*, which contains all constraints to be executed. The CHR constraint *store* S is the set[4] of *numbered* CHR constraints that can be matched with rules in the program P. For convenience we introduce functions $chr(c\#i) = c$ and $id(c\#i) = i$, and extend them to sequences and sets of numbered CHR constraints in the obvious manner.

The *built-in constraint store* B contains any built-in constraint that has been passed to the built-in solver. Since we will usually have no information about the internal representation of B, we treat it as a conjunction of constraints. The *propagation history* T is a set of sequences, each recording the identities of the CHR constraints which fired a rule, and the name of the rule itself (which may be represented as a unique integer, but typically we just use the name of the rule itself). This is necessary to prevent trivial nontermination for propagation rules: a propagation rule is allowed to fire on a set of constraints only if the constraints have not been used to fire the rule before. The set $\mathcal{V}$ contains all variables that appeared in the initial goal. Throughout this paper we will usually omit $\mathcal{V}$ unless it is explicitly required. Finally, the counter n represents the next free integer which can be used to number a CHR constraint.

Throughout we let $vars(A)$ return the variables occurring in any syntactic object A. We use $\exists_A F$ to denote the formula $\exists X_1 \cdots \exists X_n F$ where $\{X_1, \ldots X_n\} = vars(A)$. We use $\bar{\exists}_A F$ to denote the formula $\exists X_1 \cdots \exists X_n F$ where $\{X_1, \ldots X_n\} = vars(F) - vars(A)$.

We use $[H]$ to construct a sequence of length 1 containing element H, $++$ for sequence concatenation, and $\uplus$ for multiset union. We shall sometimes treat multisets as sequences, in which case we nondeterministically choose an order for the objects in the multiset. We use the notation $p(s_1, \ldots, s_n) = p(t_1, \ldots, t_n)$ as shorthand for the constraint $s_1 = t_1 \wedge \cdots \wedge s_n = t_n$, and similarly $S = T$ where S and T are equal length sequences $S \equiv s_1 \ \cdots \ s_n$ and $T = t_1 \ \cdots \ t_n$ as shorthand for $s_1 = t_1 \wedge \cdots \wedge s_n = t_n$.

We define an *initial state* as follows.

[4] Sometimes we treat the store as a multiset.

Definition 3 (Initial State). *Given a goal G, which is a multiset of constraints, the* initial state *with respect to G is* $\langle G, \emptyset, true, \emptyset\rangle_1^{vars(G)}$. □

The theoretical operational semantics ω_t is based on the following three transitions which map execution states to execution states:

Definition 4 (Theoretical Operational Semantics).
1. Solve

$$\langle\{c\} \uplus G, S, B, T\rangle_n^{\mathcal{V}} \rightarrowtail \langle G, S, c \wedge B, T\rangle_n^{\mathcal{V}}$$

where c is a built-in constraint.
2. Introduce

$$\langle\{c\} \uplus G, S, B, T\rangle_n^{\mathcal{V}} \rightarrowtail \langle G, \{c\#n\} \uplus S, B, T\rangle_{(n+1)}^{\mathcal{V}}$$

where c is a CHR constraint.

3. Apply

$$\langle G, H_1 \uplus H_2 \uplus S, B, T\rangle_n^{\mathcal{V}} \rightarrowtail \langle C \uplus G, H_1 \uplus S, B \wedge \theta, T'\rangle_n^{\mathcal{V}}$$

where there exists a (renamed apart) rule r in P of the form

$$H_1' \setminus H_2' \iff g \mid C$$

and $\theta \equiv chr(H_1) = H_1' \wedge chr(H_2) = H_2'$ such that

$$\begin{cases} \mathcal{D} \models B \rightarrow \exists_r(\theta \wedge g) \\ id(H_1) +\!\!+ \; id(H_2) +\!\!+ \; [r] \notin T \end{cases}$$

In the result $T' = T \cup \{id(H_1) +\!\!+ \; id(H_2) +\!\!+ \; [r]\}$. □

A *derivation* is a sequence of states connected by ω_t transitions. We use notation $\sigma_0 \rightarrowtail^* \sigma_1$ to represent a derivation from σ_0 to σ_1. We also define *complete derivation* to be a derivation from an initial state to a *final state* (i.e. a state where no transition is applicable).

2.1 Confluence

In this section we define confluence under the theoretical semantics ω_t.

First we define an auxiliary function *alive*, which decides which part of the propagation history is relevant for confluence.

Definition 5 (Live History). *Function alive is a bijective mapping from a CHR store S and a propagation history to a propagation history defined as follows.*

$$\begin{array}{lll} alive(S, \emptyset) & = \emptyset & \\ alive(S, \{t\} \uplus T) & = alive(S, t) \uplus alive(S, T) & \\ alive(S, t +\!\!+ [\_]) & = \emptyset & \textit{if } \exists i \in t \textit{ such that } \forall c(c\#i \notin S) \\ alive(\_, t) & = \{t\} & \textit{otherwise} \end{array}$$

□

In other words, $alive(S,T)$ is propagation history T where all entries with numbers for deleted (i.e. not alive) constraints have been removed. Interestingly, $alive(S,T)$ can only have entries on propagation rules (otherwise one of the numbers in the entry must be dead).

Definition 6 (Variants). *Two states*

$$\sigma_1 = \langle G_1, S_1, B_1, T_1 \rangle_{i_1}^{\mathcal{V}} \qquad \textit{and} \qquad \sigma_2 = \langle G_2, S_2, B_2, T_2 \rangle_{i_2}^{\mathcal{V}}$$

(from either semantics) are variants *(i.e. $\sigma_1 \approx \sigma_2$) if there exists a renaming ρ on variables not in $\mathcal{V}$ and a mapping ϱ on constraint numbers such that*

1. $\rho \circ \varrho(G_1) = G_2$
2. $\rho \circ \varrho(S_1) = S_2$*;*
3. $\mathcal{D} \models (\bar{\exists}_{\mathcal{V}} \rho(B_1) \leftrightarrow \bar{\exists}_{\mathcal{V}} B_2)$*; and*
4. $\varrho \circ alive(S_1, T_1) = alive(S_2, T_2)$*.*

Otherwise the two states are variants if $\mathcal{D} \models \neg\bar{\exists}_{\emptyset} B_1$ and $\mathcal{D} \models \neg\bar{\exists}_{\emptyset} B_2$ (i.e. both states are $false$). □

Definition 7 (Joinable). *Two states σ_1 and σ_2 are* joinable *if there exists states σ_1' and σ_2' such that $\sigma_1 \rightarrowtail^* \sigma_1'$ and $\sigma_2 \rightarrowtail^* \sigma_2'$ and σ_1' and σ_2' are variants.* □

Definition 8 (Confluence). *A CHR program P is* confluent *if the following holds for all states σ_0, σ_1 and σ_2: If $\sigma_0 \rightarrowtail^* \sigma_1$ and $\sigma_0 \rightarrowtail^* \sigma_2$ then σ_1 and σ_2 are joinable.* □

2.2 Confluence Test

The confluence test for CHRs depends on calculating all critical pairs between rules in the program.

Definition 9 (Critical Pair). *Given two (renamed apart) rules r1 and r2 from program P of the respective forms*

$$H_{1r} \setminus H_{1k} \Longleftrightarrow g_1 \mid B_1.$$
$$H_{2r} \setminus H_{2k} \Longleftrightarrow g_2 \mid B_2.$$

Let $H_1 = H_{1r} \uplus H_{1k}$ and $H_2 = H_{2r} \uplus H_{2k}$. Let $H_1' \subseteq H_1$ and $H_2' \subseteq H_2$, and let θ be a most general unifier of multisets H_1' and H_2'. Given $\theta(H_1) \uplus (H_2 - H_2') = \{h_1, ..., h_m\}$ (for some arbitrary ordering), let $S = \{h_1\#1, ..., h_m\#m\}$. Also define $S_{1k} \subseteq S$ such that $chr(S_{1k}) = \theta(H_{1k})$ and $S_{2k} \subseteq S$ such that $chr(S_{2k}) = H_{2k}$. Let T_∞ be a propagation history such that for all propagation rules $r \in P$ we have that $[i_1, ..., i_k, r] \in T_\infty$ for all permutations of all subsets $\{i_1, ..., i_k\} \in \{1, .., n\}$. Then states

$$\sigma_1 = \langle B_1, S - S_{1k}, g_1 \wedge g_2, T_\infty \rangle_n^{\mathcal{V}}$$
$$\sigma_2 = \langle B_2, S - S_{2k}, g_1 \wedge g_2, T_\infty \rangle_n^{\mathcal{V}}$$

are a critical pair *between rules r1 and r2.* □

Definition 10 (Confluence Test). *Given a terminating CHR program P, if all critical pairs between all rules in P are joinable, then P is confluent. This is known as the* confluence test *for CHRs.* □

It has been shown that the confluence test decides confluence for terminating CHR programs [1]. This relies on the fact that there are finitely many critical pairs for a given program.

2.3 Observable Confluence

In this section we formally define observable confluence with respect to reachability.

Definition 11 (Reachability). *An execution state σ is* reachable *if there exists an initial state $\sigma_0 = \langle G, \emptyset, true, \emptyset\rangle_1$ such that there exists a derivation $\sigma_0 \rightarrowtail^* \sigma$.* □

Definition 12 (Observable Confluence). *A CHR program P is* observable confluent *if the following holds for all states σ_0, σ_1 and σ_2 where σ_0 is a reachable state: If $\sigma_0 \rightarrowtail_\omega^* \sigma_1$ and $\sigma_0 \rightarrowtail_\omega^* \sigma_2$ then σ_1 and σ_2 are joinable.* □

Notice the differences between Definition 12 and Definition 8: we now require that σ_0 is a *reachable* state.

3 Observable Confluence: Examples

Confluence implies observable confluence, so if a CHR program P passes Abdennahder's confluence test, then that program is also observable confluent. In the introduction, we have seen that there exists CHR programs that are observable confluent by Definition 12, yet are not confluent by Definition 8.

Example 4. Here is again the type class program from the introduction.

```
class F x y z | x -> y
instance F Int Bool Float
```

The corresponding CHR program, according to the translation given in [8], is

```
r1 @ f(int,bool,float)    <=> true.
r2 @ f(A,B1,C), f(A,B2,D) ==> B1 = B2.
r3 @ f(int,B,C)           ==> B = bool.
```

This program is not confluent, as we saw in Example 2. Here we illustrate this formally using the full theoretical operational semantics. Consider the following state applicable to rules `r1` and `r2`. σ:

$$\langle \emptyset, \{f(int, bool, float)\#1, f(int, B2, D)\#2\}, true, \{[1, r3], [2, r3]\}\rangle_3$$

The propagation history prevents rule `r3` from firing on either constraint.

The state σ has two distinct derivations:

$$\sigma \rightarrowtail^* \langle \emptyset, \{f(int, bool, D)\#2\}, B2 = bool, \{[1, 2, r2], [1, r3], [2, r3]\}\rangle_3$$
$$\sigma \rightarrowtail^* \langle \emptyset, \{f(int, B2, D)\#2\}, true, \{[1, r3], [2, r3]\}\rangle_3$$

These two final states are non-variant, therefore the program is not confluent.

We claim this program is observable confluent. The state σ is not reachable since the propagation history indicates that rule `r3` has fired on constraint $f(int, B2, T)\#2$. However, if that were the case then we would expect the built-in constraint $B2 = bool$ to appear in the built-in store. This is not the case therefore the state σ cannot be reachable. $\square$

We formally define the class of CHR programs arising from Haskell type class declarations with functional dependencies.

Definition 13 (FD-CHR). *A CHR program P is said to be in the* FD-CHR *class of programs if it is of the form*

$$\begin{array}{l}
\texttt{r1 @ p}(X_1, ..., X_d, X_{d+1}, ..., X_r, ...)\texttt{, p}(X_1, ..., X_d, Y_{d+1}, ..., Y_r, ...) \texttt{ ==>} \\
\qquad\qquad\qquad\qquad\qquad\qquad X_{d+1} = Y_{d+1}, ..., X_r = Y_r\,\texttt{.} \\
\texttt{r2 @ p}(f_1, ..., f_n) \texttt{ <=> } B\texttt{.} \\
\texttt{r3 @ p}(f_1, ..., f_d, Y_1, ..., Y_r, ...) \texttt{ ==> } Y_1 = f_{d+1}, ..., Y_r = f_r\,\texttt{.}
\end{array}$$

where B is an arbitrary conjunction of built-in and CHR constraints and f_i are arbitrary terms such that $vars(f_{d+1}, ..., f_r) \subseteq vars(f_1, ..., f_d)$ We also require P to be terminating. Here the indices $1..d$ represent the domain *and indices $(d + 1)..r$ represent the* range *of the functional dependency. Also note that r is allowed to be less than n.* $\square$

4 Observable Confluence: Formal Result

In this section we present the main result that relates observable confluence to ordinary confluence. These are related through an *operational correspondence*.

Definition 14 (Operational Correspondence). *Let P and P' be a CHR programs. An* operational correspondence *is a function α mapping complete derivations in P to complete derivations in P', and the following conditions for all derivations*

$$\begin{aligned}
D_1 &= (\sigma_i \rightarrowtail_P^* \sigma_{1f}) \\
D_2 &= (\sigma_i \rightarrowtail_P^* \sigma_{2f}) \\
\alpha(D_1) &= (\sigma_{1i} \rightarrowtail_{P'}^* \sigma'_{1f}) \\
\alpha(D_2) &= (\sigma_{2i} \rightarrowtail_{P'}^* \sigma'_{2f})
\end{aligned}$$

are satisfied:

1. *σ_{1i} is an initial state;*
2. *$\sigma_{1i} = \sigma_{2i}$; and*
3. *If $\sigma_{1f} \not\approx \sigma_{2f}$ then $\sigma'_{1f} \not\approx \sigma'_{2f}$.*

$\square$

In other words, an operational correspondence preserves initial states, and preserves non-variance on final states. We can use operational correspondence to show observable confluence.

Theorem 1. *Let P and P' be a CHR programs such that P' is confluent and there exists an operational correspondence α from P to P', then P is observable confluent.*

Proof. By contradiction, assume P is not observable confluent. Therefore there exists an initial state σ_i and two complete derivations

$$D_1 = (\sigma_i \rightarrowtail_P^* \sigma_{1f})$$
$$D_2 = (\sigma_i \rightarrowtail_P^* \sigma_{2f})$$

such that σ_{1f} and σ_{2f} are two non-variant final states. By the operational correspondence we have that

$$\alpha(D_1) = (\sigma'_i \rightarrowtail_{P'}^* \sigma'_{1f})$$
$$\alpha(D_2) = (\sigma'_i \rightarrowtail_{P'}^* \sigma'_{2f})$$

for some initial state σ'_i and non-variant final states σ'_{1f} and σ'_{2f}. This contradicts the confluence of P', since there exists an initial state σ'_i which can be reduced to non-variant final states σ'_{1f} or σ'_{2f}. □

Example 5. Consider the following CHR program P

```
p(X), p(Y) ==> X = Y.
p(1) <=> true.
p(X) ==> X = 1
```

and the CHR program P'

```
p(X) <=> X = 1.
```

Proposition 1. *There exists an operational correspondence α from P to P' as follows:*

$$\alpha(\sigma_0 \rightarrowtail_P^* \langle\emptyset, S, B, T\rangle_n^{\mathcal{V}}) = \sigma_0 \rightarrowtail_P^* \langle\emptyset, S, B, \emptyset\rangle_n^{\mathcal{V}}$$

Note that there may be multiple possible derivations satisfying the RHS. If this is the case then we simply choose one arbitrarily to find a suitable function α.

The program P' is trivially confluent. Assuming Proposition 1 holds, then by Theorem 1 program P is observable confluent. □

5 Observable Confluence: FD-CHR

Using Theorem 1 we can reduce the problem of deciding observable confluence to the problem of finding a suitable P', such that P' is confluent and there exists a operational correspondence from P to P'. In this section we examine the FD-CHR class of programs, and use Theorem 1 to formally show they are observable confluent.

First we define the target program P' as follows.

Definition 15. *Given a program $P \in$ FD-CHR, we define the* correspondence program *$\mathcal{C}(P)$ for P as*

```
r1  @ p(X_1,...,X_d,X_{d+1},...,X_r,...), p(X_1,...,X_d,Y_{d+1},...,Y_r,...) ==>
                                    X_{d+1} = Y_{d+1},...,X_r = Y_r.
r2' @ p(f_1,...,f_n) ==> B.
r3  @ p(f_1,...,f_d,Y_1,...,Y_r,...) ==> Y_1 = f_{d+1},...,Y_r = f_r.
```

□

The only difference is between rules r2 and r2': r2 is a simplification rule and r2' is an equivalent propagation rule. The remaining structure of the program is preserved. Note that $\mathcal{C}(P)$ is terminating since P is terminating.[5]

5.1 $\mathcal{C}(P)$ Confluence

In this section we establish that the class of $\mathcal{C}(P)$ is confluent.

Lemma 1. *For all $P \in$ FD-CHR, $\mathcal{C}(P)$ is confluent.*

Proof. Direct proof. We apply the standard CHR confluence test. Since there are only propagation rules in $\mathcal{C}(P)$, all critical pairs are trivially joinable. Therefore $\mathcal{C}(P)$ is confluent. □

5.2 $\mathcal{C}(P)$ Operational Correspondence

In this section we establish an operational correspondence between P and $\mathcal{C}(P)$.

First we define two auxiliary tests to help make the proofs more concise.

Definition 16. *We define a test $\mathsf{Inst}_B(p(F_1, ...F_n))$ that succeeds if there exists a substitution θ such that*

$$D \models B \rightarrow (p(F_1, ...F_n) = p(\theta.f_1, ..., \theta.f_n))$$

Similarly we define $\mathsf{Inst}_B^{DOM}(p(F_1, ...F_n))$ that succeeds if there exists a substitution θ such that

$$D \models B \rightarrow (p(F_1, ...F_n) = p(\theta.f_1, ..., \theta.f_d, F_{d+1}, ..., F_n)))$$

□

Informally, Inst_B defines the set of all constraints that match r2 (or r2'), and Inst_B^{DOM} defines the set of all constraints that match r3.

Lemma 2. *If*

$$\sigma_0 \rightarrowtail_P^* \langle G, S, B, T\rangle_n$$

then

$$\sigma_0 \rightarrowtail_{\mathcal{C}(P)}^* \langle G, S \uplus S', B, T \cup T'\rangle_n$$

where S' is

$$\{p(F_1^1, ..., F_n^1)\#i_1, ..., p(F_1^m, ..., F_n^m)\#i_m\}$$

for some set of constraint numbers $i_1, ..., i_m$, $\mathsf{Inst}_B(p(F_1^j, ...F_n^j))$ holds for all $j \in 1, \ldots, m$. And for all $t \in T'$, t is of the form $[j, r2']$ for some $j \in i_1, ..., i_m$

[5] We omit a formal proof for space reasons.

Proof. By induction over the derivation steps in P.
Base case: No derivation steps. Then $S' = \emptyset$ and $T' = \emptyset$.
Induction step: Suppose for derivations D_i of length i of the form

$$\sigma_0 \rightarrowtail_P^* \langle G_i, S_i, B_i, T_i \rangle_{n_i}$$

we have that there exists a derivation D'_i of the form

$$\sigma_0 \rightarrowtail_{\mathcal{C}(P)}^* \langle G_i, S_i \uplus S'_i, B_i, T_i \cup T'_i \rangle_{n_i}$$

where S'_i and T'_i satisfy the conditions on S' and T' from above respectively.

Consider all derivations of length $i+1$ constructed from applying an ω_t transition to the final state in D_i. The possible transitions are:

- **Solve**. Then $G_i = \{c\} \uplus G'_i$ for some built-in constraint c. Thus

  $$\langle \{c\} \uplus G'_i, S_i, B_i, T_i \rangle_{n_i} \rightarrowtail_{\mathbf{Solve}} \langle G'_i, S_i, c \wedge B_i, T_i \rangle_{n_i}$$

  and for $\mathcal{C}(P)$

  $$\langle \{c\} \uplus G'_i, S_i \uplus S'_i, B_i, T_i \cup T'_i \rangle_{n_i} \rightarrowtail_{\mathbf{Solve}} \langle G'_i, S_i \uplus S'_i, c \wedge B_i, T_i \cup T'_i \rangle_{n_i}$$

  Thus the induction hypothesis holds for $i+1$ with $S'_{i+1} = S'_i$ and $T'_{i+1} = T'_i$.
- **Introduce**. Then $G_i = \{c\} \uplus G'_i$ for some CHR constraint c. Thus

  $$\langle \{c\} \uplus G'_i, S_i, B_i, T_i \rangle_{n_i} \rightarrowtail_{\mathbf{Introduce}} \langle G'_i, \{c\#n_i\} \uplus S_i, B_i, T_i \rangle_{n_i+1}$$

  and for $\mathcal{C}(P)$

  $$\langle \{c\} \uplus G'_i, S_i \uplus S'_i, B_i, T_i \cup T'_i \rangle_{n_i} \rightarrowtail_{\mathbf{Introduce}} \langle G'_i, \{c\#n_i\} \uplus S_i \uplus S'_i, B_i, T_i \cup T'_i \rangle_{n_i+1}$$

  Thus the induction hypothesis holds for $i+1$ with $S'_{i+1} = S'_i$ and $T'_{i+1} = T'_i$.
- **Apply**. We split this case into two smaller cases:
  1. **Apply** r2. Then $S_i = \{p(F_1, ..., F_n)\#j\} \uplus S$ for some constraint number j and CHR store S where $\mathsf{Inst}_{B_i}(p(F_1, ..., F_n))$. Thus

     $$\langle G_i, \{p(F_1, ..., F_n)\#j\} \uplus S, B_i, T_i \rangle_{n_i} \rightarrowtail_{\mathbf{Apply}} \langle B \wedge G_i, S, B_i \wedge \theta, T_i \rangle_{n_i}$$

     and

     $$\begin{array}{c} \langle G_i, \{p(F_1, ..., F_n)\#j\} \uplus S \uplus S'_i, B_i, T_i \cup T'_i \rangle_{n_i} \rightarrowtail_{\mathbf{Apply}} \\ \langle B \wedge G_i, \{p(F_1, ..., F_n)\#j\} \uplus S \uplus S'_i, B_i \wedge \theta, T_i \cup T'_i \cup \{[j, r2']\} \rangle_{n_i} \end{array}$$

     Thus the induction hypothesis holds for $i+1$ with $S'_{i+1} = S'_i \uplus \{p(F_1, ..., F_n)\#j\}$ and $T'_{i+1} = T'_i \cup \{[j, r2']\}$.
  2. **Apply** r1 $\vee$ r3. Either r1 or r3 is applicable to the constraints in S_i. Thus

     $$\langle G_i, S_i, B_i, T_i \rangle_{n_i} \rightarrowtail_{\mathbf{Apply}} \langle C \uplus G_i, S_i, B_i \wedge \theta, \{t\} \cup T_i \rangle_{n_i}$$

     and using $\mathcal{C}(P)$

     $$\langle G_i, S_i \uplus S'_i, T_i \cup T'_i \rangle_{n_i} \rightarrowtail_{\mathbf{Apply}} \langle C \uplus G_i, S_i \uplus S'_i, B_i \wedge \theta, \{t\} \cup T_i \cup T'_i \rangle_{n_i}$$

     for some θ, C and t. Thus the induction hypothesis holds for $i+1$ with $S'_{i+1} = S'_i$ and $T'_{i+1} = T'_i$.

□

Lemma 3. *Given a complete derivation under* P

$$\sigma_0 \rightarrowtail_P^* \langle \emptyset, S, B, T \rangle_n^{\mathcal{V}} = \sigma_f$$

and the corresponding derivation under $\mathcal{C}(P)$ *(given by Lemma 2)*

$$\sigma_0 \rightarrowtail_{\mathcal{C}(P)}^* \langle \emptyset, S \cup S', B, T \cup T' \rangle_n^{\mathcal{V}} = \sigma_1$$

suppose that $\sigma_1 \rightarrowtail_{\mathcal{C}(P)}^* \sigma_i$ *then* σ_i *is of the form*

$$\langle G_i, S \cup S', B_i, T \cup T' \cup T_i \rangle_n^{\mathcal{V}}$$

such that

1. $\mathcal{D} \models \bar{\exists}_{\mathcal{V}} B \leftrightarrow \bar{\exists}_{\mathcal{V}} B_i$*;*
2. *for all* $c \in G_i$*,* c *is built-in and* $\mathcal{D} \models B_i \rightarrow c$*;*
3. *for all* $t \in T_i$ *we have that* $id(t) \cap id(S') \neq \emptyset$*.*

Proof. By induction over the derivation steps in $\sigma_1 \rightarrowtail_{\mathcal{C}(P)}^* \sigma_i$
Base case: $i = 1$, thus $G_i = \emptyset$, $B_i = B$ and $T_i = \emptyset$ satisfies the conditions.
Induction step: Suppose that for $i - 1$ we have that

$$\sigma_{i-1} = \langle G_{i-1}, S \cup S', B_{i-1}, T \cup T' \cup T_{i-1} \rangle_n^{\mathcal{V}}$$

is of the required form.

We consider all ω_t transition steps applicable to σ_{i-1}.

- **Solve**. Then $G_{i-1} = c \wedge G'_{i-1}$ for some built-in constraint c, thus

  $$\sigma_{i-1} \rightarrowtail_{\mathbf{Solve}} \langle G'_{i-1}, S \cup S', c \wedge B_{i-1}, T \cup T' \cup T_{i-1} \rangle_n^{\mathcal{V}}$$

  We see that $G_i = G'_{i-1} \subset G_{i-1}$ and $T_i = T_{i-1}$ satisfies conditions (2) and (3) respectively. For condition (1): Since $\mathcal{D} \models \bar{\exists}_{\mathcal{V}} B \leftrightarrow \bar{\exists}_{\mathcal{V}} B_{i-1}$ and $\mathcal{D} \models B \rightarrow c$ we have that $B_i = (c \wedge B_{i-1})$ satisfies $\mathcal{D} \models \bar{\exists}_{\mathcal{V}} B \leftrightarrow \bar{\exists}_{\mathcal{V}} B_i$. Thus condition (1) is satisfied.
- **Introduce**. This transition is not applicable since there are no CHR constraints in G_{i-1}.
- **Apply**. A rule is applied to the constraints in $S \uplus S'$. Thus

  $$\sigma_{i-1} \rightarrowtail_{\mathbf{Apply}} \langle C \uplus G_{i-1}, S \cup S', B_{i-1} \wedge \theta, T \cup T_{i-1} \cup \{t\} \rangle_n$$

  for some entry t, rule body C and matching substitution θ.
  Now $D \models B \rightarrow \exists_r \theta$ and $D \models \bar{\exists}_{\mathcal{V}} B \leftrightarrow \bar{\exists}_{\mathcal{V}} B_{i-1}$ hence $D \models \bar{\exists}_{\mathcal{V}} B \leftrightarrow \bar{\exists}_{\mathcal{V}}(B_{i-1} \wedge \theta)$ satisfying condition (1). Let $M \subseteq S \uplus S'$ be the *matching*, then $M \cap S' \neq \emptyset$ otherwise σ_f is not a final state. Thus $id(t) \cap id(S') \neq \emptyset$ and hence $T_i = T_{i-1} \cup \{t\}$ satisfies condition (3).
  Next consider $G_i = C \uplus G_{i-1}$. For condition (2) to hold, we need to show that for all $c \in C$ we have that c is a built-in constraint, and $\mathcal{D} \models B_{i-1} \wedge \theta \rightarrow c$. There are three cases to consider:

1. **Apply** r1. Then $M = \{p(F_1, ..., F_n)\#j, p(T_1, ..., T_n)\#k\}$, for some constraint numbers j, k. C is of the form:

$$\{X_{d+1} = Y_{d+1}, ..., X_r = Y_r\} \tag{1}$$

Obviously C is all built-in constraints as required. Also, $\theta \equiv (\bigwedge_{l=1}^{n} F_l = X_l) \wedge (\bigwedge_{l=1}^{d} T_l = X_l) \wedge (\bigwedge_{l=d+1}^{n} T_l = Y_l)$ and $D \models B_{i-1} \rightarrow \exists\bar{X}\exists\bar{Y}\theta$.
Since $M \cap S' \neq \emptyset$ we can assume w.l.o.g. $p(F_1, ..., F_n)\#k \in S'$. Therefore $\mathsf{Inst}_{B_{i-1}}(p(F_1, ..., F_n))$ holds.
From $D \models B_{i-1} \rightarrow \exists\bar{X}\exists\bar{Y}\theta$ we project out $\bar{X}$ and $\bar{Y}$ to derive

$$D \models B_{i-1} \rightarrow \bigwedge_{l=1}^{d} F_l = T_l \tag{2}$$

I.e. the domain arguments must coincide. Therefore $\mathsf{Inst}_{B_{i-1}}^{DOM}(p(T_1, ..., T_n))$ must hold, and r3 is applicable to this rule. Since σ_f is a final state, rule r3 must have already been applied to this constraint, and thus we can conclude $\mathsf{Inst}_{B_{i-1}}(p(T_1, ..., T_n))$ holds.
Since $vars(f_{d+1}, ..., f_r) \subseteq vars(f_1, ..., f_d)$, and by (2) we have that

$$D \models B_{i-1} \rightarrow \bigwedge_{l=d+1}^{r} F_l = T_l$$

Therefore

$$D \models (B_{i-1} \wedge \theta) \rightarrow \bigwedge_{l=d+1}^{r} X_l = Y_l$$

I.e. the equations of C are already implied by $B_i = (B_{i-1} \wedge \theta)$ and thus condition (2) is satisfied.
2. **Apply** r2'. Then $M = \{p(F_1, ..., F_n)\#j\} \subseteq S'$ (since $M \cap S' \neq \emptyset$) for some constraint number j,
Since $M \cap S' \neq \emptyset$ then by Lemma 2 there exists an entry $[j, r2'] \in T'$, thus **Apply** is not applicable so we can exclude this case.
3. **Apply** r3. Then $M = \{p(F_1, ..., F_n)\#j\} \subseteq S'$ (since $M \cap S' \neq \emptyset$) for some constraint number j, and hence $\mathsf{Inst}_{B_{i-1}}(p(F_1, ..., F_n))$ holds.
Now $\theta \equiv (\bigwedge_{l=1}^{d} F_l = f_l) \wedge (\bigwedge_{l=d+1}^{n} F_l = Y_l)$ and C is of the form

$$\{Y_{d+1} = f_{d+1}, ..., Y_r = f_r\}$$

Since $vars(f_{d+1}, ..., f_r) \subseteq vars(f_1, ..., f_d)$, we have that clearly $\mathcal{D} \models (B_{i-1} \wedge \theta) \rightarrow c$ for all $c \in C$ and thus condition (2) holds.

In either case the state σ_i satisfies the required conditions.

□

Lemma 4. *There exists an operational correspondence between $P \in$ FD-CHR and $\mathcal{C}(P)$.*

Proof. Direct proof. By Lemma 3 and termination of $\mathcal{C}(P)$, for a complete derivation D in P

$$\sigma_0 \rightarrowtail_P^* \sigma_f = \langle \emptyset, S, B, T \rangle_n^{\mathcal{V}}$$

there exists a complete derivation D' in $\mathcal{C}(P)$

$$\sigma_0 \rightarrowtail_{\mathcal{C}(P)}^* = \langle \emptyset, S \uplus S', B_i, T \cup T' \cup T_i \rangle_n^{\mathcal{V}}$$

where S', B_i, T' and T_i satisfy the conditions of Lemmas 2 and 3.

Define $\alpha(D) = D'$. Next we check that α satisfies the conditions for Definition 14. Given the complete derivations

$$\begin{aligned} D_1 &= (\sigma_0 \rightarrowtail_P^* \sigma_{1f}) \\ D_2 &= (\sigma_0 \rightarrowtail_P^* \sigma_{2f}) \\ \alpha(D_1) &= (\sigma_{1i} \rightarrowtail_{P'}^* \sigma'_{1f}) \\ \alpha(D_2) &= (\sigma_{2i} \rightarrowtail_{P'}^* \sigma'_{2f}) \end{aligned}$$

we have that

1. σ_{1i} is an initial state since $\sigma_{1i} = \sigma_0$;
2. $\sigma_{1i} = \sigma_{2i}$ since also $\sigma_{2i} = \sigma_0$; and
3. if $\sigma_{1f} \not\approx \sigma_{2f}$ then $\sigma'_{1f} \not\approx \sigma'_{2f}$. We show this condition is satisfied by contradiction. Assume that $\sigma_{1f} \not\approx \sigma_{2f}$ but $\sigma'_{1f} \approx \sigma'_{2f}$.
   Let

$$\begin{aligned} \sigma_{1f} &= \langle \emptyset, S_1, B_1, T_1 \rangle_n^{\mathcal{V}} \\ \sigma_{2f} &= \langle \emptyset, S_2, B_2, T_2 \rangle_m^{\mathcal{V}} \\ \sigma'_{1f} &= \langle \emptyset, S_1 \uplus S'_1, B_{1i}, T_1 \cup T'_1 \cup T_{1i} \rangle_n^{\mathcal{V}} \\ \sigma'_{2f} &= \langle \emptyset, S_2 \uplus S'_2, B_{2i}, T_2 \cup T'_2 \cup T_{2i} \rangle_m^{\mathcal{V}} \end{aligned}$$

   where S'_1, S'_2, B_{1i}, B_{2i}, T'_1, T'_2, T_{1i} and T_{2i} satisfy the conditions of Lemmas 2 and 3 in the obvious manner.
   Given Definition 6, for all renamings ρ on variables not in $\mathcal{V}$ and mappings ϱ on constraint numbers, there are four cases the consider:
   (a) *Goals:* $\rho \circ \varrho(\emptyset) \neq \emptyset$ gives an immediate contradiction.
   (b) *CHR Stores:* Given that $\rho \circ \varrho(S_1) \neq S_2$ then, w.l.o.g., there exists a numbered constraint $c\#j \in \rho \circ \varrho(S_1)$ such that $c\#j \notin S_2$.
   Thus if $\rho \circ \varrho(S_1 \uplus S'_1) = (S_2 \uplus S'_2)$ we have that $c\#j \in S_2 \uplus S'_2$, therefore $c\#j \in S'_2$. Thus c must satisfy $\mathsf{Inst}_{B_{2i}}(c)$, and hence rule r2 is applicable to $c\#j$ in state σ_{1f}. Therefore σ_{1f} cannot be a final state which is a contradiction.
   (c) *Built-in Stores:* Given that $\mathcal{D} \models (\bar{\exists}_{\mathcal{V}} \rho(B_1) \not\leftrightarrow \bar{\exists}_{\mathcal{V}} B_2)$ assume that $\mathcal{D} \models (\bar{\exists}_{\mathcal{V}} \rho(B_{1i}) \leftrightarrow \bar{\exists}_{\mathcal{V}} B_{2i})$.
   We immediately arrive at a contradiction since $\mathcal{D} \models \bar{\exists}_{\mathcal{V}} B_1 \leftrightarrow \bar{\exists}_{\mathcal{V}} B_{1i}$ and $\mathcal{D} \models \bar{\exists}_{\mathcal{V}} B_2 \leftrightarrow \bar{\exists}_{\mathcal{V}} B_{2i}$.

(d) *Histories:* Let $T_{1a} = \varrho \circ alive(S_1, T_1)$ and $T_{2a} = alive(S_2, T_2)$. Given that $T_{1a} \neq T_{2a}$, w.l.o.g., there exists a propagation history entry t such that $t \in T_{1a}$ but $t \notin T_{2a}$.
Let

$$T'_{1a} = \varrho \circ alive(S_1 \uplus S'_1, T_1 \uplus T'_1 \uplus T_{1i})$$
$$T'_{2a} = alive(S_2 \uplus S'_2, T_2 \uplus T'_2 \uplus T_{2i})$$

By assumption, $T'_{1a} = T'_{2a}$. By Definition 5 we see that $T_{1a} \subseteq T'_{1a}$, thus $t \in T'_{1a}$. Since $T'_{1a} = T'_{2a}$ we have that $t \in T'_{2a}$.
Expanding out the shorthand notation, the above is equivalent to

$$t \in \varrho \circ alive(S_1, T_1)$$
$$t \notin alive(S_2, T_2)$$
$$t \in alive(S_2 \uplus S'_2, T_2 \uplus T'_2 \uplus T_{2i})$$

There are three cases to consider:

i. $t \in T'_2$. Then $t = [j, r2']$ for some j. We immediately arrive at a contradiction since rule r2' is not present in program P, yet the propagation history for σ_{1f} (namely T_1) mentions it.

ii. $t \in T'_{2i}$. Then by Lemma 3 we have that $id(t) \cap id(S'_2) \neq \emptyset$. Thus there exists a $c\#j \in S'_2$ such that $j \in t$. Therefore $\mathsf{Inst}_{B_{2i}}(c)$ holds. Since $t \in \varrho \circ alive(S_1, T_1)$ we have that $c\#j' \in S_1$ where $\varrho(j') = j$. Since $\mathcal{D} \models \bar{\exists}_{\mathcal{V}} B_{1i} \leftrightarrow \bar{\exists}_{\mathcal{V}} B_{2i}$ we have that $\mathsf{Inst}_{B_{1i}}(c)$ also holds, thus constraint c is applicable to rule r2, therefore σ_{1f} is not a final state. This is a contradiction.

iii. $t \in T_2$. If $id(t) \subset id(S_2)$ we instantly reach a contradiction, since this implies $t \in alive(S_2, T_2)$. Therefore $id(t) \cap id(S'_2) \neq \emptyset$, however this leads to the same contradiction as preceding case.

Each case leads to a contradiction, therefore if $\sigma_{1f} \not\approx \sigma_{2f}$ then $\sigma'_{1f} \not\approx \sigma'_{2f}$.

Therefore α is an operational correspondence between P and $\mathcal{C}(P)$. □

5.3 Main result

Corollary 1. *All programs $P \in$ FD-CHR are observable confluent.*

Proof. Directly follows from Theorem 1 and Lemmas 1 and 4. □

6 Conclusion and Future Work

We have investigated observable confluence for a class of non-confluent CHRs which arise when building type inferencer for type class programs with functional dependencies [8]. Our results guarantee that all reachable states during the type class inference process are confluent. Thus, we obtain completeness of inference for a larger set of type class programs. For simplicity we considered the cases arising from one instance and one functional dependency, the results extend straightforwardly for program arising from many instance declarations and functional dependencies (assuming the instances satisfy the functional dependencies).

In future work, we plan to consider further classes of CHRs which are observable confluent. There are a number of other situations where weaker notions of observable confluence are also important. We briefly sketch one of these cases.

Lam and Sulzmann [6] are using CHRs for agent-oriented programming. These CHRs fail the confluence test. However, in practice CHRs are only applied to constraint stores which satisfy certain invariants of the agent world. CHR applications maintain these invariants. Hence, we should obtain observable confluence for all initial states which satisfy a certain condition. For example, consider the following two simplified CHRs modelling the block world, a standard example of an agent world.

```
g1 @ get(X), empty   <=> rhs1
g2 @ get(X), hold(Y) <=> rhs2
```

The critical pair `get(X), empty, hold(Y)` is not be joinable depending on the right-hand sides. Hence, the above CHRs are non-confluent. However, we know that the CHRs for the block world obey the invariant that `empty` and `hold(Y)` will never appear in any constraint store at the same time. Hence, we argue that the CHRs are observable confluent with respect to the condition that initial constraint store does not contain `empty` and `hold(Y)`.

References

1. S. Abdennadher. Operational semantics and confluence of constraint propagation rules. In *Proc. of CP'97*, LNCS, pages 252–266. Springer-Verlag, 1997.
2. G.J. Duck, M. García de la Banda, P.J. Stuckey, and C. Holzbaur. The refined operational semantics for constraint handling rules. In B. Demoen and V. Lifschitz, editors, *Proceedings of the 20th International Conference on Logic Programming*, LNCS, pages 120–136. Springer-Verlag, 2004.
3. T. Frühwirth. Constraint handling rules. In *Constraint Programming: Basics and Trends*, LNCS. Springer-Verlag, 1995.
4. P. J. Stuckey G. J. Duck, S. Peyton-Jones and M. Sulzmann. Sound and decidable type inference for functional dependencies. In *Proc. of ESOP'04*, volume 2986 of *LNCS*, pages 49–63. Springer-Verlag, 2004.
5. M. P. Jones. Type classes with functional dependencies. In *Proc. of ESOP'00*, volume 1782 of *LNCS*. Springer-Verlag, 2000.
6. E. S. L. Lam and M. Sulzmann. Representing linear logic agents in CHR. http://www.comp.nus.edu.sg/˜ sulzmann, May 2006.
7. S. Peyton Jones, editor. *Haskell 98 Language and Libraries: The Revised Report*. Cambridge University Press, 2003.
8. M. Sulzmann, G.J. Duck, S. Peyton-Jones, and P.J. Stuckey. Understanding functional dependencies via constraint handling rules. *Journal of Functional Programming*, page to appear, 2006.