Persistent Constraints in Constraint Handling Rules

Hariolf Betz, Frank Raiser, and Thom Frühwirth

Faculty of Engineering and Computer Sciences, Ulm University, Germany
firstname.lastname@uni-ulm.de

Abstract. In the most abstract definition of its operational semantics, the declarative and concurrent programming language CHR is trivially non-terminating for a significant class of programs. Common refinements of this definition, in closing the gap to real-world implementations, compromise on declarativity and/or concurrency. Building on recent work and the notion of persistent constraints, we introduce an operational semantics avoiding trivial non-termination without compromising on its essential features.

1 Introduction

Constraint Handling Rules [1] (CHR) is a declarative, multiset- and rule-based programming language suitable for concurrent execution and powerful program analyses. Several operational semantics have been proposed for CHR [1], situated between an abstract and an implementation level.

The most abstract operational semantics – constituting the basis for most other variants – is called the "very abstract" operational semantics and denoted as ω_{va}. It is firmly rooted in first-order logic, defining a state transition system but providing no execution model. Hence, it is oblivious to termination issues, unfavorably causing the class of rules known as *propagation rules* to induce *trivial non-termination*.

The de-facto standard in avoiding trivial non-termination is set by the operational semantics ω_t [1], providing the basis for most available CHR implementations. In ω_t, every propagation rule is applicable only once to a specific combination of constraints, thus avoiding trivial non-termination. This is realized by keeping a *propagation history* – also called *token store* – in the program state.

On the downside, token stores break with declarativity: Two states that differ only in their token stores may exhibit different operational behaviour while sharing the same logical reading. Therefore, we consider token stores as *non-declarative elements* in program states. The propagation history also hinders effective concurrent execution of CHR programs, as it has to be distributed adequately.

With concurrency in mind, [2] defines operational semantics based on sets rather than multisets, which effectively avoids trivial non-termination without

recurring to token stores. With nine transition rules, however, the resulting state transition system is unusually complex, thus reducing clarity and complicating formal proofs. Furthermore, abandoning multiset semantics is a severe break with existing approaches and the presence of non-declarative elements remains. Notably, the authors of [2] reckon that any "reasonable [multiset-based] semantics" for CHR requires a propagation history. This work is proof to the contrary.

Recent work on linear logical algorithms [3] and the close relation of CHR to linear logic [4] suggest a novel approach: we introduce the notion of *persistent constraints* to CHR, a concept reminiscent of "banged" propositions in linear logic. Persistent constraints provide a finite representation of the results of any number of propagation rule firings. Furthermore, we explicitly define our state transition system as irreflexive. It shows that, in combination, these ideas solve the problem of trivial non-termination.

Building on earlier work in [5], we thus develop the operational semantics $\omega_!$ for CHR in this work. As opposed to existing approaches, it achieves a high degree of declarativity whilst preserving the potential of ω_{va} for effective concurrent execution. Its state transition system requires only two rules, such that each transition step corresponds to a CHR rule application, thus facilitating formal reasoning over programs.

In Section 2 we introduce CHR and present its operational semantics ω_{va} and ω_t. We then introduce $\omega_!$ and discuss its properties in Section 3, before comparing it to other approaches in Section 4. Finally, in Section 5 we conclude and consider possible directions of future work.

2 Preliminaries

This section introduces CHR and its two most important operational semantics. A complete listing of available operational semantics is given in [6]. In this work, we concentrate on the so-called *very abstract* operational semantics ω_{va} and *theoretical* operational semantics ω_t. A *refined* variant of the latter – introduced in [7] and denoted as ω_r – reduces several sources of non-determinism and is the de-facto standard for CHR implementations.

The very abstract operational semantics ω_{va} is the semantics with the simplest state definition and fewest restrictions on rule applications. We introduce it in Section 2.2 before presenting its refinement into the theoretical operational semantics ω_t in Section 2.3.

2.1 The Syntax of CHR

Constraint Handling Rules distinguishes between two kinds of constraints: *user-defined* (or simply CHR) constraints and *built-in* constraints. The latter are processed by a predefined solver implementing a complete and decidable constraint theory $\mathcal{CT}$.

CHR itself is an advanced rule-based rewriting language. Its eponymous rules are of the form

$$r@\ H_1 \backslash H_2 \Leftrightarrow G \mid B_c, B_b$$

where H_1 and H_2 are multisets of user-defined constraints, called the *kept head* and *removed head*, respectively. The *guard* G is a conjunction of built-in constraints and the *body* consists of a conjunction of built-in constraints B_b and a multiset of user-defined constraints B_c. The *rule name* r is optional and may be omitted along with the @ symbol.

Intuitively speaking, a rule is applicable, if a part of the current state can be matched to all head constraints such that the guard is satisfied. Application of a rule removes from the state the constraints matched to H_2 and adds the guards and the body constraints to the state. In this work, we put special emphasis on the class of rules where $H_2 = \emptyset$, called *propagation rules*. Propagation rules can be written alternatively as $H_1 \Rightarrow G \mid B_c, B_b$.

2.2 Very Abstract Operational Semantics

The very abstract operational semantics ω_{va} [1] is the most general specification of an operational semantics for CHR. Its state definition only contains one component and the transition system is given by a single rule.

Definition 1 (ω_{va}-State).
A ω_{va}-state $\sigma_{va} = \langle C \rangle$ *is a conjunction* C *of built-in and CHR constraints.*

The only allowed state transition in ω_{va} is the application of a CHR rule.

Definition 2 (ω_{va}-Transition). *Let* $r \ @ \ H_1 \setminus H_2 \Leftrightarrow G \mid B$ *be an instance of a rule* $r \in P$ *with new local variables* $\bar{x}$ *and* $\mathcal{CT} \models \forall(\mathbb{G} \rightarrow \exists \bar{x}.G)$. *Then*

$$\langle H_1 \wedge H_2 \wedge \mathbb{G} \rangle \rightarrowtail_{\omega_{va}} \langle H_1 \wedge G \wedge B \wedge \mathbb{G} \rangle$$

Note that the above definition, based on instantiation of rules, requires all arguments of constraints in H_1 and H_2 to be variables. Ground terms can be realized by an equality constraint in the guard, and similarly, multiple occurrences of the same variable are not allowed, but have to be realized via guard constraints. This restriction simplifies the formulation of ω_{va}, but it also makes for less elegant programs. Most derived operational semantics – including ω_t, ω_{set}, and $\omega_!$ discussed herein – avoid this restriction.

An inherent problem of ω_{va} is its behavior with respect to propagation rules: If a state can fire a propagation rule once, it can do so again and again, ad infinitum. In the literature, this problem is referred to as *trivial non-termination* of propagation rules.

2.3 Theoretical Operational Semantics

The theoretical operational semantics ω_t [1, 6] is based on the idea of using a token store to avoid trivial non-termination. Under ω_t, a propagation rule can only be applied once to each combination of constraints matching the head. Hence, the token store keeps a history of fired propagation rules, which is based on constraint identifiers.

Definition 3 (Identified CHR Constraints).
An identified CHR constraint $c\#i$ *is a CHR constraint* c *associated with a unique integer* i*, the* constraint identifier*. We introduce the functions* $\mathrm{chr}(c\#i) = c$ *and* $\mathrm{id}(c\#i) = i$*, and extend them to sequences and sets of identified CHR constraints in the obvious manner.*

The definition of a CHR state in ω_t is more complicated, because identified constraints are distinguished from unidentified constraints and the token store is added [1].

Definition 4 (ω_t-State).
A ω_t-state *is a tuple of the form* $\langle \mathbb{G}, \mathbb{S}, \mathbb{B}, \mathbb{T} \rangle_n^{\mathbb{V}}$ *where the* goal (store) $\mathbb{G}$ *is a multiset of constraints, the* CHR (constraint) store $\mathbb{S}$ *is a set of numbered CHR constraints, the* built-in (constraint) store $\mathbb{B}$ *is a conjunction of built-in constraints. The* propagation history *(or* token store*)* $\mathbb{T}$ *is a set of tuples* (r, I)*, where* r *is the name of a propagation rule and* I *is an ordered sequence of the identifiers of constraints that matched the head constraints of* r *in a previous application of* r*. Finally, the set* $\mathbb{V}$ *of* global variables *contains the variables that occur in the initial goal.*

This state definition entails a more complicated transition system, consisting of the following three types of transitions:

Definition 5 (ω_t-Transitions).

1. **Solve.** $\langle \{c\} \uplus \mathbb{G}, \mathbb{S}, \mathbb{B}, \mathbb{T} \rangle_n^{\mathbb{V}} \rightarrowtail_{\omega_t} \langle \mathbb{G}, \mathbb{S}, \mathbb{B}', \mathbb{T} \rangle_n^{\mathbb{V}}$
   where c *is a built-in constraint and* $\mathcal{CT} \models \forall((c \wedge \mathbb{B}) \leftrightarrow B')$.
2. **Introduce.** $\langle \{c\} \uplus \mathbb{G}, \mathbb{S}, \mathbb{B}, \mathbb{T} \rangle_n^{\mathbb{V}} \rightarrowtail_{\omega_t} \langle \mathbb{G}, \{c\#n\} \cup \mathbb{S}, \mathbb{B}, \mathbb{T} \rangle_{n+1}^{\mathbb{V}}$
   where c *is a CHR constraint.*
3. **Apply.** $\langle \mathbb{G}, H_1 \cup H_2 \cup \mathbb{S}, \mathbb{B}, \mathbb{T} \rangle_n^{\mathbb{V}} \rightarrowtail_{\omega_t} \langle B \uplus \mathbb{G}, H_1 \cup \mathbb{S}, \mathrm{chr}(H_1) = H_1' \wedge \mathrm{chr}(H_2) = H_2' \wedge G \wedge \mathbb{B}, \mathbb{T} \cup \{(r, \mathrm{id}(H_1) + \mathrm{id}(H_2))\} \rangle_n^{\mathbb{V}}$
   where $r \ @ \ H_1' \setminus H_2' \Leftrightarrow G \mid B$ *is a fresh variant of a rule in* P *with fresh variables variables* $\bar{x}$ *such that* $\mathcal{CT} \models \exists(\mathbb{B}) \wedge \forall(\mathbb{B} \rightarrow \exists \bar{x}(\mathrm{chr}(H_1) = H_1' \wedge \mathrm{chr}(H_2) = H_2' \wedge G))$ *and* $(r, \mathrm{id}(H_1) + \mathrm{id}(H_2)) \notin \mathbb{T}$.

By construction, ω_t restricts the number of applications of a propagation rule for each given combination of head constraints to one. This stands in contrast to its declarative reading as well as its execution under ω_{va}, where a propagation rule may be applied any number of times. The ω_t-state also contains non-declarative elements: the set of identified CHR constraints, the propagation history, and the integer n used for identification.

3 Operational Semantics with Persistent Constraints

We now introduce our proposal for an operational semantics $\omega_!$ with persistent constraints. It is based on three important ideas:

1. In ω_{va}, the body of a propagation rule can be generated any number of times given that the corresponding head constraints are present in the store. In order to give consideration to this theoretical behavior while avoiding trivial non-termination, we introduce those body constraints as so-called *persistent constraints*. A persistent constraint can be regarded as a finite representation of a very large, though unspecified number of identical constraints. For a proper distinction, constraints that are non-persistent are henceforth called *linear* constraints.
2. Not only the bodies of propagation rules can be generated indefinitely many times in ω_{va}. Consider the following program:

$$\begin{aligned} &\text{r1} \ @ \ a(X) \Longrightarrow b(X) \\ &\text{r2} \ @ \ b(X) \ \Leftrightarrow \ c(X) \end{aligned}$$

   If executed with a goal $a(0)$, this program can generate an arbitrary number of constraints of the form $b(0)$. As a consequence of this, it can also generate arbitrarily many constraints $c(0)$. To take these indirect consequences of propagation rules into accout, we introduce a rule's body constraints as persistent, whenever its removed head can be matched completely with persistent constraints.
3. As a persistent constraint represents an arbitrary number of identical constraints, we consider several occurences of a persistent constraint as idempotent. We now adapt our execution model such that a transition takes place only if the post-transition state is not equivalent to the pre-transition state. By the thus irreflexive transition system, we avoid trivial non-termination of propagation rules.

To realize the first two ideas, we adapt the definion of states in $\omega_!$ with respect to ω_t: The goal store $\mathbb{G}$ of ω_t-states is split into a store $\mathbb{L}$ of linear constraints and a store $\mathbb{P}$ of persistent constraints. The components $\mathbb{B}$ and $\mathbb{V}$ of ω_t-states are retained, but the token-related components $\mathbb{S}, \mathbb{T}$, and n are eliminated.

Definition 6 ($\omega_!$-State).
A $\omega_!$-state *is a tuple of the form* $\langle \mathbb{L}, \mathbb{P}, \mathbb{B}, \mathbb{V} \rangle$*, where* $\mathbb{L}$ *and* $\mathbb{P}$ *are multisets of CHR constraints called the* linear (CHR) store *and* persistent (CHR) store*, respectively.* $\mathbb{B}$ *is a conjunction of built-in constraints and* $\mathbb{V}$ *is a set of variables.*

We define the notion of *strictly local variables* which we will apply below.

Definition 7 (Strictly local variables). *Let* $\sigma = \langle \mathbb{L}, \mathbb{P}, \mathbb{B}, \mathbb{V} \rangle$ *be an* $\omega_!$ *state. Then we call the variables occurring in* $\mathbb{B}$ *but not in* $\mathbb{L}$*,* $\mathbb{P}$*, or* $\mathbb{V}$ *the* strictly local variables *of* σ*.*

To realize the third idea, we adapt the equivalence relation between $\omega_!$-states. The following definition of state equivalence is based on [5], adding condition 5 to handle idempotence of persistent constraints.

Definition 8 (Equivalence of $\omega_!$-States).
Equivalence between $\omega_!$*-states is the smallest equivalence relation* $\equiv$ *over* $\omega_!$*-states that satisfies the following conditions:*

1. (Equality as Substitution) *Let X be a variable, t be a term and $\doteq$ the syntactical equality relation.*

$$\langle \mathbb{L}, \mathbb{P}, X \doteq t \wedge \mathbb{B}, \mathbb{V} \rangle \equiv \langle \mathbb{L}\,[X/t]\,, \mathbb{P}\,[X/t]\,, X \doteq t \wedge \mathbb{B}, \mathbb{V} \rangle$$

2. (Transformation of the Constraint Store) *If $\mathcal{CT} \models \exists \bar{s}.\mathbb{B} \leftrightarrow \exists \bar{s}'.\mathbb{B}'$ where $\bar{s}, \bar{s}'$ are the strictly local variables of $\mathbb{B}, \mathbb{B}'$, respectively, then:*

$$\langle \mathbb{L}, \mathbb{P}, \mathbb{B}, \mathbb{V} \rangle \equiv \langle \mathbb{L}, \mathbb{P}, \mathbb{B}', \mathbb{V} \rangle$$

3. (Omission of Non-Occurring Global Variables) *If X is a variable that does not occur in $\mathbb{L}$, $\mathbb{P}$ or $\mathbb{B}$ then:*

$$\langle \mathbb{L}, \mathbb{P}, \mathbb{B}, \{X\} \cup \mathbb{V} \rangle \equiv \langle \mathbb{L}, \mathbb{P}, \mathbb{B}, \mathbb{V} \rangle$$

4. (Equivalence of Failed States)

$$\langle \mathbb{L}, \mathbb{P}, \bot, \mathbb{V} \rangle \equiv \langle \mathbb{L}', \mathbb{P}', \bot, \mathbb{V}' \rangle$$

5. (Contraction)

$$\langle \mathbb{L}, P \uplus P \uplus \mathbb{P}, \mathbb{B}, \mathbb{V} \rangle \equiv \langle \mathbb{L}, P \uplus \mathbb{P}, \mathbb{B}, \mathbb{V} \rangle$$

Based on this definition of state equivalence, we define CHR as a rewrite system over equivalence classes of states. Let Σ be the set of all $\omega_!$-states, then the transition relation $\rightarrowtail_{\omega_!}$ satisfies $\rightarrowtail_{\omega_!} \subseteq (\Sigma/\equiv) \times (\Sigma/\equiv)$. Note that we use the term *state* interchangably to denote $\omega_!$-states *per se*, as well as equivalence classes over such states. As discussed above, we require that a post-transition state τ needs to be different to the pre-transition state σ, thus making the transition relation irreflexive.

Definition 9 ($\omega_!$-Transitions).

$$\frac{\begin{array}{c} r \;@\; (H_1^l \uplus H_1^p) \backslash (H_2^l \uplus H_2^p) \Leftrightarrow G \mid B_c, B_b \quad H_2^l \neq \emptyset \quad \sigma \neq \tau \\ \sigma = [\langle H_1^l \uplus H_2^l \uplus \mathbb{L}, H_1^p \uplus H_2^p \uplus \mathbb{P}, G \wedge \mathbb{B}, \mathbb{V} \rangle] \end{array}}{\rightarrowtail_{\omega_!} [\langle H_1^l \uplus B_c \uplus \mathbb{L}, H_1^p \uplus H_2^p \uplus \mathbb{P}, G \wedge \mathbb{B} \wedge B_b, \mathbb{V} \rangle] = \tau}$$

$$\frac{\begin{array}{c} r \;@\; (H_1^l \uplus H_1^p) \backslash H_2^p \Leftrightarrow G \mid B_c, B_b \quad \sigma \neq \tau \\ \sigma = [\langle H_1^l \uplus \mathbb{L}, H_1^p \uplus H_2^p \uplus \mathbb{P}, G \wedge \mathbb{B}, \mathbb{V} \rangle] \end{array}}{\rightarrowtail_{\omega_!} [\langle H_1^l \uplus \mathbb{L}, H_1^p \uplus H_2^p \uplus B_c \uplus \mathbb{P}, G \wedge \mathbb{B} \wedge B_b, \mathbb{V} \rangle] = \tau}$$

Note that in a concurrent environment, the second inference rule can be executed without any restrictions: As persistent constraints cannot be removed by other rule applications every process can independently use them to fire rules. The first inference rule can be executed concurrently, if is is guaranteed, that rule applications do not interfere, in the manner described in [8].

3.1 Termination Behavior

Our proposed operational semantics $\omega_!$ results in a termination behavior different from ω_t and ω_{va}. Compared to ω_{va}, the problem of trivial non-termination is solved in $\omega_!$. In comparison with ω_t, we find that there exist programs that terminate under $\omega_!$ but not under ω_t, and vice versa.

Example 1. Consider the following straightforward CHR program for computing the transitive hull of a graph represented by edge constraints $e/2$:

$$\mathrm{t} \ @ \ e(X,Y), e(Y,Z) \Longrightarrow e(X,Z)$$

Due to the presence of propagation rules, this program is not terminating under ω_{va}. Under ω_t, termination depends on the initial goal: It is shown in [9] that this program terminates for acyclic graphs. However, goals containing graphs with cycles, like $\langle (e(1,2), e(2,1)), \emptyset, \top, \emptyset \rangle_0^{\emptyset}$, result in nontermination.

When executed under $\omega_!$, the previous goal terminates after computing the transitive hull.

$$\begin{aligned} & \langle \{e(1,2), e(2,1)\}, \emptyset, \top, \emptyset \rangle \\ \rightarrowtail_{\omega_!}^{t} \ & \langle \{e(1,2), e(2,1)\}, \{e(1,1)\}, \top, \emptyset \rangle \\ \rightarrowtail_{\omega_!}^{*} \ & \langle \{e(1,2), e(2,1)\}, \{e(1,1), e(1,2), e(2,1), e(2,2)\}, \top, \emptyset \rangle \not\rightarrowtail_{\omega_!} \end{aligned}$$

In fact, we can show that the above program terminates under $\omega_!$ for all possible inputs.

Proposition 1. *Under $\omega_!$, the transitive hull program terminates for every possible input.*

Proof. The only rule t propagates e constraints, which are necessarily persistent. The propagated constraints contain only the arguments X, Z, already recieved as parameters. Hence, no new arguments are introduced. Any given initial state contains only a finite number of arguments. Therefore, only finitely many different e constraints can be built from these arguments. As rule application is irreflexible, the computation therefore has to stop after a finite number of transition steps. □

Example 2. Consider the following exemplary CHR program:

$$\begin{aligned} \mathrm{r1} \ @ \ a & \Longrightarrow b \\ \mathrm{r2} \ @ \ c(X), b & \Leftrightarrow c(X+1) \end{aligned}$$

The above program terminates under ω_t and ω_r: There can only be a finite number of a-constraints in the initial goal, hence rule $r1$ only creates a finite number of b-constraints. This, in turn, allows only a finite number of increments being made by rule $r2$.

In contrast, our proposed semantics $\omega_!$ results in the above program being non-terminating, as the following infinite derivation shows:

$$\begin{aligned} & \langle \{a, c(X)\}, \emptyset, \top, \{X\} \rangle \\ \rightarrowtail_{\omega_!}^{r1} \ & \langle \{a, c(X)\}, \{b\}, \top, \{X\} \rangle \\ \rightarrowtail_{\omega_!}^{r2} \ & \langle \{a, c(X+1)\}, \{b\}, \top, \{X\} \rangle \\ \rightarrowtail_{\omega_!}^{r2} \ & \langle \{a, c(X+2)\}, \{b\}, \top, \{X\} \rangle \rightarrowtail_{\omega_!}^{r2} \ldots \end{aligned}$$

3.2 Limitations of the current approach

The approach specified in this work entails a significant discrepance w.r.t. ω_{va} when fresh variables are introduced in rule bodies. For example, consider the following program:

$$\begin{array}{ll} \text{r1} \ @ \ a & \Longrightarrow b(X) \\ \text{r2} \ @ \ b(X), b(X) & \Leftrightarrow \ c \end{array}$$

If executed with the initial goal a, this program would cause the following infinite derivation under ω_{va}:

$$\begin{array}{l} \quad \langle a \rangle \\ \rightarrowtail^{r1}_{\omega_{va}} \ \langle a \wedge b(X') \rangle \\ \rightarrowtail^{r1}_{\omega_{va}} \ \langle a \wedge b(X') \wedge b(X'') \rangle \rightarrowtail^{r1}_{\omega_{va}} \ \ldots \end{array}$$

The variables $X', X'', \ldots$ each are explicitly distinct from each other and from the variable X which occurs in the rule body. Thus, it is not possible to derive the constraint c from goal a under ω_{va}.

Under $\omega_!$, however, the following derivation is possible:

$$\begin{array}{l} \quad \langle \{a\}, \emptyset, \top, \emptyset \rangle \\ \rightarrowtail^{r1}_{\omega_!} \ \langle \{a\}, \{b(X')\}, \top, \emptyset \rangle \equiv \langle \{a\}, \{b(X'), b(X')\}, \top, \emptyset \rangle \\ \rightarrowtail^{r2}_{\omega_!} \ \langle \{a\}, \{b(X'), c\}, \top, \emptyset \rangle \equiv \langle \{a\}, \{b(X'), b(X'), c\}, \top, \emptyset \rangle \end{array}$$

Therefore, the current formulation of the operational semantics $\omega_!$ for CHR is only applicable to range-restricted programs, i.e. rules that do not introduce new variables in their bodies.

4 Related Work

In [2] the set-based semantics ω_{set} has been introduced. Its development was, amongst other considerations, driven by the intention to eliminate the propagation history. Besides addressing the problem of trivial non-termination in a novel manner, it reduces non-determinism in a way closely resembling ω_r.

Similarly to ω_t, a propagation rule is only fired once for a possible matching in ω_{set}. Unlike ω_t, however, additional single firings are possible in ω_{set}. These depend on the further development of the built-in store. Nonetheless, there remains a limit on the number of rule firings.

Our approach to eliminate trivial non-termination consists of the combination of two essential components: an irreflexive state transition system and persistent constraints. Using irreflexivity for termination is a straightforward consequence of adding persistent constraints. The separation of propositions, or constraints, into linear and persistent ones was inspired by the work on linear logical algorithms in [3]. CHR differs significantly from linear logical algorithms, because of its support for built-in constraints, their underlying constraint theory and interaction with user-defined constraints.

Figure 1 relates the different operational semantics for CHR in a hierarchical order. At the root, we have the abstract semantics ω_{va} from which the other

semantics are derived. The operational semantics ω_t introduces token stores to solve the trivial non-termination problem. Numerous extensions of it have been published [6], as indicated by a dotted elements in the figure. In particular, the operational semantics ω_r [7] is an important specialization of ω_t, as it is the foundation of most existing CHR implementations. Again, numerous extensions apply to ω_r.

In the right-hand column, we have placed ω_{set}, which, is another specialization of ω_{va}. Having identified shortcomings of the token store approach, the authors of [2] give a set-based operational semantics for CHR instead.

By placing our approach into the middle column, we emphasize that it is a distinct approach to the trivial non-termination problem. The remaining entries in Figure 1 under the category of persistent semantics indicate that $\omega_!$ – analogously to ω_t – can serve as the basis for a multitude of extensions and implementation-specific variants.

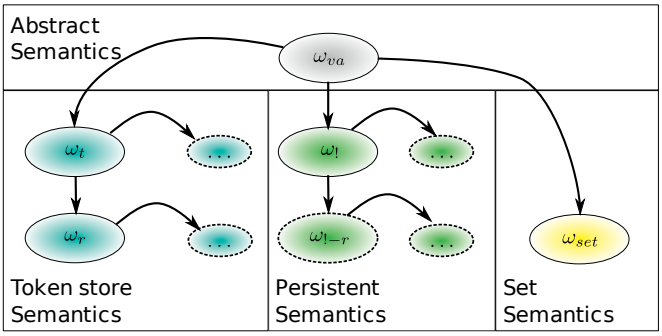

Fig. 1. Relations between Operational Semantics of CHR

The benefits of $\omega_!$ in comparison with the other cited approaches are summarized in Figure 2. In the following, we discuss the different evaluation criteria and the corresponding results given in Figure 2.

Termination on propagation rules: While forming the basis for all other semantics, ω_{va} itself is a theoretical construct, made impractical by its trivial non-termination on propagation rules. Derived semantics apply various strategies to avoid this problem, as outlined above.

Effective concurrency: In ω_t and ω_r, the necessity to distribute token stores constitutes an impediment to effective concurrent execution. We deem ω_{va}, ω_{set}, and $\omega_!$ effective bases for concurrent execution, as they do not introduce auxiliary elements causing inference between rule applications.

Declarative states: In ω_t, ω_r, and ω_{set}, program states contain elements that have no correspondence in the declarative reading. States in $\omega_!$ and ω_{va} avoid such non-declarative elements, thus simplifying proofs of program properties.

Number of transition rules: To varying degrees, the transition systems of the investigated operational semantics encompass concrete execution strategies. Especially in the cases of ω_r and ω_{set}, this makes for a large number

of transition rules at the expense of clarity and simplicity of proofs. Second only to ω_{va}, our system $\omega_!$ consists of only two inference rules. More importantly, each transition step corresponds to an application of a CHR rule, thus simplifying proofs of program properties.

Preservation of multiset semantics: It should be noted that the multiset semantics is a key feature of CHR, although strictly speaking it is in contrast with the paradigm of declaritivity w.r.t. first-order logic. It is already present in the constitutive semantics ω_{va} and is effectively made use of in many programs. In this respect, ω_{set} exerts a strong break with the tradition of CHR that $\omega_!$ avoids.

Reduced non-determinism: The refined semantics ω_r and the set-based semantics ω_{set} significantly reduce the inherent non-determinism of CHR: Firstly, they determine that rules are to be tried for applicability in textual order. Secondly, they fix the order in which CHR constraints are tried to match to the rules. Our semantics $\omega_!$, along with ω_{va} and ω_t, is distinctly non-deterministic. Nonetheless, it leaves open the possibility of restricting non-determinism, analogously to ω_r reducing the non-determinism of ω_t. However, this comes at the cost of additional transition rules and possibly introducing non-declarative elements into states.

	ω_{va}	ω_t	ω_r	ω_{set}	$\omega_!$
Termination on propagation rules:	-	+	+	+	+
Effective concurrency:	+	-	-	+	+
Declarative states:	+	-	-	-	+
Number of transition rules:	1	3	7	9	2
Preservation of multiset semantics:	+	+	+	-	+
Reduced non-determinism:	-	-	+	+	-

Fig. 2. Comparison of the different operational semantics

5 Conclusion and Future Work

For this work, we investigated the extent to which several desirable features are found in the most prominent operational semantics of CHR. As Figure 2 shows, each semantics displays certain limitations for specific fields of application. Inspired by linear logic, we introduced the concept of persistent constraints to CHR. Building on earlier work in [5], we proposed a novel operational semantics $\omega_!$ that provides a better trade-off between the most significant features.

The transition system of $\omega_!$ consists of two rules, such that each transition directly corresponds to a CHR rule application. Its irreflexive formulation straightforwardly solves the trivial non-termination problem. Furthermore, all elements

of $\omega_!$-states correspond to the declarative reading of states. Both properties facilitate formal proofs of program properties and hence are advantageous for program analysis.

Concerning concurrency, $\omega_!$ inherits the suitability for concurrent execution from ω_{va}. In this respect, persistent constraints have a clear advantage over token stores: As they do not hinder rule applications, their distribution is not critical and less synchronization is required.

Our proposed operational semantics $\omega_!$ displays a termination behavior different from the commonly used operational semantics ω_t. The classes of programs terminating under $\omega_!$ and ω_t do not contain each other. Hence, either semantics may be more favorable, depending on the application. Also, in its current formulation, $\omega_!$ is only applicable to range-restricted CHR programs – a limitation we plan to address in the future.

Furthermore, we intend to formulate and prove clear statements on the soundness and completeness of our semantics with respect to ω_{va} and to further investigate the differing termination behavior between $\omega_!$ and other semantics. Finally, as ω_t is the basis for numerous extensions to CHR [6], we plan to investigate the effect of building these extensions on $\omega_!$ instead.

References

1. Frühwirth, T.: Constraint Handling Rules. Cambridge University Press (2009)
2. Sarna-Starosta, B., Ramakrishnan, C.: Compiling Constraint Handling Rules for efficient tabled evaluation. In Hanus, M., ed.: 9th Intl. Symp. Practical Aspects of Declarative Languages, PADL. Volume 4354 of Lecture Notes in Computer Science., Nice, France, Springer-Verlag (jan 2007) 170–184
3. Simmons, R.J., Pfenning, F.: Linear logical algorithms. In Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I., eds.: Automata, Languages and Programming, 35th International Colloquium, ICALP 2008. Volume 5126 of Lecture Notes in Computer Science., Springer-Verlag (2008) 336–347
4. Betz, H., Frühwirth, T.: A linear-logic semantics for constraint handling rules. In van Beek, P., ed.: Principles and Practice of Constraint Programming, 11th International Conference, CP 2005. Volume 3709 of Lecture Notes in Computer Science., Sitges, Spain, Springer-Verlag (October 2005) 137–151
5. Raiser, F., Betz, H., Frühwirth, T.: Equivalence of CHR states revisited. In Raiser, F., Sneyers, J., eds.: 6th International Workshop on Constraint Handling Rules (CHR). (2009) 34–48
6. Sneyers, J., Van Weert, P., Schrijvers, T., De Koninck, L.: As time goes by: Constraint Handling Rules – A survey of CHR research between 1998 and 2007. Accepted by *Journal of Theory and Practice of Logic Programming* (2008)
7. Duck, G.J., Stuckey, P.J., García de la Banda, M., Holzbaur, C.: The refined operational semantics of Constraint Handling Rules. In Demoen, B., Lifschitz, V., eds.: Logic Programming, 20th International Conference, ICLP 2004. Volume 3132 of Lecture Notes in Computer Science., Saint-Malo, France, Springer-Verlag (September 2004) 90–104
8. Sulzmann, M., Lam, E.S.L.: Parallel execution of multi-set constraint rewrite rules. In Antoy, S., Albert, E., eds.: Proceedings of the 10th International ACM SIG-

PLAN Conference on Principles and Practice of Declarative Programming (PPDP), Valencia, Spain, ACM (July 2008) 20–31
9. Pilozzi, P., Schreye, D.D.: Proving termination by invariance relations. In Hill, P.M., Warren, D.S., eds.: 25th International Conference Logic Programming, ICLP. Volume 5649 of Lecture Notes in Computer Science., Pasadena, CA, USA, Springer-Verlag (July 2009) 499–503