TLP **11** (6): 841–879, 2011. © Cambridge University Press 2010
doi:10.1017/S147106841000044X First published online 29 July 2010

Concurrent goal-based execution of Constraint Handling Rules

EDMUND S. L. LAM
School of Computing, National University of Singapore,
S16 Level 5, 3 Science Drive 2, Singapore 117543
(*e-mail:* lamsoonl@comp.nus.edu.sg)

MARTIN SULZMANN
Informatik Consulting Systems AG, Sonnenbergstr. 13, 70184 Stuttgart, Germany
(*e-mail:* martin.sulzmann@gmail.com)

submitted 5 December 2008; revised 18 January 2010; accepted 31 May 2010

Abstract

We introduce a systematic, concurrent execution scheme for Constraint Handling Rules (CHR) based on a previously proposed sequential goal-based CHR semantics. We establish strong correspondence results to the abstract CHR semantics, thus guaranteeing that any answer in the concurrent, goal-based CHR semantics is reproducible in the abstract CHR semantics. Our work provides the foundation to obtain efficient, parallel CHR execution schemes.

KEYWORDS: multiset rewriting, constraints, concurrency

1 Introduction

Rewriting is a powerful discipline to specify the semantics of programming languages and to perform automated deduction. There are numerous flavors of rewriting such as term, graph rewriting etc. Our focus here is on exhaustive, forward chaining, multiset constraint rewriting as found in Constraint Handling Rules (CHR) (Frühwirth 1998), which are used in a multitude of applications such as general purpose constraint programming, type system design, agent specification and planning, etc. (Frühwirth 2006). Rewriting steps are specified via CHR rules which replace a multiset of constraints matching the left-hand side of a rule (also known as rule head) by the rule's right-hand side (also known as rule body).

CHR support a very fine-grained form of concurrency. CHR rules can be applied concurrently if the rewriting steps they imply do not interfere with each other. An interesting feature of CHR is that the left-hand side of a CHR rule can have a mix of simplified and propagated constraint patterns. This provides the opportunity for further concurrency. We can execute CHR rules concurrently as long as only their propagated parts overlap.

https://doi.org/10.1017/S147106841000044X Published online by Cambridge University Press

The fact that the abstract CHR semantics is highly concurrent has so far not been exploited in any major CHR implementation. Existing implementations are specified by highly deterministic semantics (Duck *et al*. 2004; De Koninck *et al*. 2008), which support efficient and systematic but inherently single-threaded execution schemes (Duck 2005; Schrijvers 2005). Our goal is to develop a systematic, yet concurrent, semantics which can be efficiently executed in parallel on a multicore architecture. In the CHR context, there is practically no prior work which addresses this important issue.

Specifically, we make the following contributions:

- We develop a novel goal-based concurrent CHR semantics.
- We verify that our semantics respects the abstract CHR semantics by establishing precise correspondence results.
- We examine which existing sequential CHR optimizations carry over to the concurrent setting.

Section 4 contains the details. A concrete parallel implementation derived from our concurrent semantics is studied elsewhere (Sulzmann and Lam 2008). Section 5.2 provides a summary.

The upcoming section gives an overview of our work. Section 3 reviews the abstract CHR semantics. Section 5 discusses prior work on execution schemes for CHR and production rule systems which are related rewriting mechanisms. Section 6 concludes.

2 Overview

We first motivate concurrent execution of CHR rules via a few examples. Then, we review existing deterministic CHR execution schemes which are the basis for our concurrent goal-based CHR semantics.

2.1 Constraint Handling Rules and concurrency

Figures 1 and 2 contain several examples of CHR rules and derivations. We adopt the convention that lower-case symbols refer to variables and upper-case symbols refer to constraints. The notation *rulename*@ assigns distinct labels to CHR rules.

The first example simulates a simple communication channel. The $Get(x)$ constraint represents the action of writing a value from the communication channel into the variable x, while the $Put(y)$ constraint represents the action of putting the value y into the channel. The interaction between both constraints is specified via the CHR rule *get* which specifies the replacement of constraints matching $Get(x)$ and $Put(y)$ by $x = y$. The point to note is that in contrast to Prolog, we use matching and not unification to trigger rules.

For example, the constraint store $\{Get(m), Put(1)\}$ matches the left-hand side of the *get* rule by instantiating x by m and y by 1. Hence, $\{Get(m), Put(1)\}$ rewrites to the answer $\{m = 1\}$. We write $\{Get(m), Put(1)\} \rightarrowtail_{get} \{m = 1\}$ to denote this

https://doi.org/10.1017/S147106841000044X Published online by Cambridge University Press

Communication channel:

$$get@Get(x), Put(y) \Longleftrightarrow x = y$$

$$\frac{\{Get(m), Put(1)\} \rightarrowtail_{get} \{m = 1\} \parallel \{Get(n), Put(8)\} \rightarrowtail_{get} \{n = 8\}}{\{Get(m), Put(1), Get(n), Put(8)\} \rightarrowtail^* \{m = 1, n = 8\}}$$

Greatest common divisor:

$$gcd_1@Gcd(0) \Longleftrightarrow True$$
$$gcd_2@Gcd(n)\backslash Gcd(m) \Longleftrightarrow m \geq n \&\& n > 0 \mid Gcd(m - n)$$

$$\frac{\begin{array}{c}\{Gcd(3), Gcd(9)\} \rightarrowtail_{gcd2} \{Gcd(3), Gcd(6)\} \\ \parallel \\ \{Gcd(3), Gcd(3)\} \rightarrowtail_{gcd2} \{Gcd(3), Gcd(0)\}\end{array}}{\begin{array}{lll}\{Gcd(3), Gcd(3), Gcd(9)\} & \rightarrowtail_{gcd2,gcd2} & \{Gcd(3), Gcd(0), Gcd(6)\} \\ & \rightarrowtail^* & \{Gcd(3)\}\end{array}}$$

$$\{Gcd(3), Gcd(3), Gcd(9)\} \rightarrowtail^* \{Gcd(3)\}$$

Fig. 1. Communication channel and greatest common divisor.

derivation step. Similarly, we find that $\{Get(n), Put(8)\} \rightarrowtail_{get} \{n = 8\}$. Rules can be applied concurrently as long as they do not interfere. In our case, the two derivations above can be concurrently executed, indicated by the symbol $\parallel$, and we can straightforwardly combine both derivations which leads to the final answer $\{m = 1, n = 8\}$. We write $\rightarrowtail^*$ to denote exhaustive rule application.

The answer $\{m = 8, n = 1\}$ is also possible but the CHR rewrite semantics is committed-choice. We can guarantee a unique answer if the CHR rules are confluent which means that rewritings applicable on overlapping constraint sets are always joinable. In general, (non)confluence is of no concern to us here and is left to the programmer (if desired). We follow here the abstract CHR semantics (Frühwirth 1998) (formally defined in Section 3) which is inherently indeterministic. Rewrite rules can be applied in any order and thus CHR enjoy a high degree of concurrency.

The key to concurrency in CHR is monotonicity which guarantees that CHR executions remain valid if we include a larger context (i.e. store). The following result has been formally verified in Abdennadher *et al.* (1999),

Theorem 1 (*Monotonicity of CHR*)

For any sets of CHR constraints *A*,*B* and *S*, if $A \rightarrowtail^* B$ then $A \uplus S \rightarrowtail^* B \uplus S$

An immediate consequence of monotonicity is that concurrent CHR executions are sound in the sense that their effect can be reproduced using an appropriate

https://doi.org/10.1017/S147106841000044X Published online by Cambridge University Press

$$merge1@Leq(x,a)\backslash Leq(x,b) \iff a < b \mid Leq(a,b)$$
$$merge2@Merge(n,a), Merge(n,b) \iff a < b \mid Leq(a,b), Merge(n+1,a)$$

Shorthands: $L = Leq$ and $M = Merge$

$$\begin{array}{ll} & M(1,a), M(1,c), M(1,e), M(1,g) \\ \rightarrowtail_{merge2} & M(2,a), M(1,c), M(1,e), L(a,g) \\ \rightarrowtail_{merge2} & M(2,a), M(2,c), L(a,g), L(c,e) \\ \rightarrowtail_{merge2} & M(3,a), L(a,g), L(c,e), L(a,c) \\ \rightarrowtail_{merge1} & M(3,a), L(a,c), L(c,g), L(c,e) \\ \rightarrowtail_{merge1} & M(3,a), L(a,c), L(c,e), L(e,g) \end{array}$$

$$\Vert$$

$$\begin{array}{ll} & M(1,b), M(1,d), M(1,f), M(1,h) \\ \rightarrowtail^* & M(3,b), L(b,d), L(d,f), L(f,h) \end{array}$$

$$\begin{array}{ll} & M(3,a), L(a,c), L(c,e), L(e,g), M(3,b), L(b,d), L(d,f), L(f,h) \\ \rightarrowtail_{merge2} & M(4,a), L(a,c), L(a,b), L(c,e), L(e,g), L(b,d), L(d,f), L(f,h) \\ \rightarrowtail_{merge1} & M(4,a), L(a,b), L(b,c), L(c,e), L(e,g), L(b,d), L(d,f), L(f,h) \\ \rightarrowtail_{merge1} & M(4,a), L(a,b), L(b,c), L(c,d), L(c,e), L(e,g), L(d,f), L(f,h) \\ \rightarrowtail_{merge1} & M(4,a), L(a,b), L(b,c), L(c,d), L(d,e), L(e,g), L(d,f), L(f,h) \\ \rightarrowtail_{merge1} & M(4,a), L(a,b), L(b,c), L(c,d), L(d,e), L(e,f), L(e,g), L(f,h) \\ \rightarrowtail_{merge1} & M(4,a), L(a,b), L(b,c), L(c,d), L(d,e), L(e,f), L(f,g), L(f,h) \\ \rightarrowtail_{merge1} & M(4,a), L(a,b), L(b,c), L(c,d), L(d,e), L(e,f), L(f,g), L(g,h) \end{array}$$

$$\begin{array}{ll} & M(1,a), M(1,c), M(1,e), M(1,g), M(1,b), M(1,d), M(1,f), M(1,h) \\ \rightarrowtail^* & M(4,a), L(a,b), L(b,c), L(c,d), L(d,e), L(e,f), L(f,g), L(g,h) \end{array}$$

Fig. 2. Merge sort.

sequential sequence of execution steps. Thus, we can derive the following rule:

$$\textbf{(Concurrency)} \quad \frac{S \uplus S_1 \rightarrowtail^* S \uplus S_2 \quad S \uplus S_3 \rightarrowtail^* S \uplus S_4}{S \uplus S_1 \uplus S_3 \rightarrowtail^* S \uplus S_2 \uplus S_4}$$

In Frühwirth (2005), the above is referred to as "Strong Parallelism of CHR". However, we prefer to use the term "concurrency" instead of "parallelism". In the CHR context, concurrency means to run a CHR program (i.e. a set of CHR rules) by using concurrent execution threads.

Let us consider the second CHR example from Figure 1 which computes the greatest common divisor among a set of numbers by applying Euclid's algorithm. The left-hand side of rule $gcd2$ is interesting because it uses a mix of simplified and propagated constraint patterns. We replace (simplify) $Gcd(m)$ by $Gcd(m-n)$ but keep (propagate) $Gcd(n)$ if the guard $m >= n \&\& n > 0$ holds. For example, we find that $\{Gcd(3), Gcd(9)\} \rightarrowtail_{gcd2} \{Gcd(3), Gcd(6)\}$ and $\{Gcd(3), Gcd(3)\} \rightarrowtail_{gcd2}$

https://doi.org/10.1017/S147106841000044X Published online by Cambridge University Press

$\{Gcd(3), Gcd(0)\}$. The point to note is the above rule applications only overlap on the propagated part. Hence, we can execute both rewrite derivations simultaneously

$$\{Gcd(3), Gcd(3)Gcd(9)\} \rightarrowtail_{2\times gcd2} \{Gcd(3), Gcd(0), Gcd(6)\}$$

Our last example in Figure 2 is a CHR encoding of the well-known merge sort algorithm. To sort a sequence of (distinct) elements $e_1, \ldots, e_m$ where m is a power of 2, we apply the rules to the initial constraint store

$$Merge(1, e_1), \ldots, Merge(1, e_m)$$

Constraint $Merge(n, e)$ refers to a sorted sequence of numbers at level n whose smallest element is e. Constraint $Leq(a, b)$ denotes that a is less than b. Rule $merge2$ initiates the merging of two sorted lists and creates a new sorted list at the next level. The actual merging is performed by rule $merge1$. Sorting of sublists belonging to different mergers can be performed simultaneously. See the example derivation in Figure 2 where we simultaneously sort the characters a, c, e, g and b, d, f, h.

2.2 Goal-based CHR execution

Existing CHR implementation employ a more systematic CHR execution model where rules are triggered based on a set of available goals. The idea behind a goal-based CHR execution model is to separate the constraint store into two components: a set of goal constraints (constraints yet to be executed) and the actual constraint store (constraints that were executed). Previously, in the abstract semantics transitions $\rightarrowtail_{\mathscr{A}}$ are among states *Store* whereas in the goal-based semantics we find now transitions $\rightarrowtail_{\mathscr{g}}$ among states of the form $\langle Goals \mid Store \rangle$. Only goal constraints can trigger rules by searching for store constraint to build a complete match for a rule head, thus allowing for execution of the rule.

Below, we give a goal-based execution of the earlier communication buffer example.

$$get\ @Get(x), Put(y) \Longleftrightarrow x = y$$

$$\begin{array}{lll}
 & & \langle \{Get(x_1), Get(x_2), Put(1), Put(2)\} \mid \{\} \rangle \\
\text{(D1 Activate)} & \rightarrowtail_{\mathscr{g}} & \langle \{Get(x_1)\#1, Get(x_2), Put(1), Put(2)\} \mid \{Get(x_1)\#1\} \rangle \\
\text{(D2 Drop)} & \rightarrowtail_{\mathscr{g}} & \langle \{Get(x_2), Put(1), Put(2)\} \mid \{Get(x_1)\#1\} \rangle \\
\text{(D3 Activate)} & \rightarrowtail_{\mathscr{g}} & \langle \{Get(x_2)\#2, Put(1), Put(2)\} \mid \{Get(x_1)\#1, Get(x_2)\#2\} \rangle \\
\text{(D4 Drop)} & \rightarrowtail_{\mathscr{g}} & \langle \{Put(1), Put(2)\} \mid \{Get(x_1)\#1, Get(x_2)\#2\} \rangle \\
\text{(D5 Activate)} & \rightarrowtail_{\mathscr{g}} & \langle \{Put(1)\#3, Put(2)\} \mid \{Get(x_1)\#1, Get(x_2)\#2, Put(1)\#3\} \rangle \\
\text{(D6 Fire } get\text{)} & \rightarrowtail_{\mathscr{g}} & \langle \{Put(2), x_1 = 1\} \mid \{Get(x_2)\#2\} \rangle \\
\text{(D7 Activate} & \rightarrowtail_{\mathscr{g}} & \langle \{Put(2)\#3, x_1 = 1\} \mid \{Get(x_2)\#2, Put(2)\#3\} \rangle \\
\text{(D8 Fire } get\text{)} & \rightarrowtail_{\mathscr{g}} & \langle \{x_1 = 1, x_2 = 2\} \mid \{\} \rangle \\
\text{(D9 Solve)} & \rightarrowtail_{\mathscr{g}} & \langle \{x_2 = 2\} \mid \{x_1 = 1\} \rangle \\
\text{(D10 Solve)} & \rightarrowtail_{\mathscr{g}} & \langle \{\} \mid \{x_1 = 1, x_2 = 2\} \rangle
\end{array}$$

We label the xth derivation step by a label Dx. Let us walk through each of the individual goal-based execution steps. Initially, all constraints are kept in

https://doi.org/10.1017/S147106841000044X Published online by Cambridge University Press

the set of goals. At this point, all of the goals are inactive. Execution of goals proceeds in two stages: (1) Activation and (2a) rule execution, or (2b) dropping of goals. In the first stage, we activate a goal. In general, the order in which goals are activated is arbitrary. For concreteness, we assume a left-to-right activation order.

Hence, we first activate $Get(x_1)$ in derivation step (D1). Active goals carry a unique identifier, a distinct integer number. Besides assigning numbers to active goals, we also put them into the store. For instance, after activating $Get(x_1)$, we have $Get(x_1)\#1$ in both the goals and the store[1].

Active goals like $Get(x_1)\#1$ are executed by trying to build a complete match for a rule head with matching partner constraints in the store. Since there are no other constraints in the store, we cannot match $Get(x_1)\#1$ with the *get* rule. Therefore we drop $Get(x_1)\#1$ in step (D2). Dropping of a goal means the goal is removed from the set of goals but of course the (now inactive) goal is still present in the store. Step (D3) and (D4) are similar but executed on goal $Get(x_2)$. Then, we activate $Get(x_2)$ and find that $Get(x_2)\#2$ cannot build a complete match of the *get* rule, thus it is dropped too.

Next, we activate $Put(1)$ (Step D5). Constraint $Put(1)\#3$ can match with either $Get(x_1)\#1$ or $Get(x_2)\#2$ to form a complete instance of rule head of *get*. We pick $Get(x_1)\#1$ and fire the rule *get*, see step (D6). Step (D7) and (D8) perform similar execution steps on $Put(2)$ and the remaining stored constraint $Get(x_2)\#2$. Finally, we add the equations $x_1 = 1$ and $x_2 = 2$ into the store in steps (D9) and (D10). Exhaustive application of this goal-based execution strategy then leads to a state with no goals and a final store.

What we have described so far is essentially the execution scheme in which all major CHR implementations are based on. The semantics of these implementations assume a deterministic activation policy. For example, goals are kept in a stack (Duck *et al*. 2004) or priority queue (De Koninck *et al*. 2008). This of course implies a strictly sequential execution scheme.

To obtain a systematic, yet concurrent, CHR execution scheme we adapt the goal-based CHR semantics as follows. Several active goal constraints can simultaneously seek for partner constraints in the store to fire a rule instance. In the extreme case, all goal constraints could be activated at once. However, we generally assume that the number of active goals are bounded by n where n corresponds to the the number of actual threads available to the run-time system (e.g., processor cores).

Figure 3 shows a sample concurrent goal-based CHR derivation. We assume two concurrent threads, referred to as a and b, each thread executes the standard goal-based derivation steps. The novelty is that each goal-based derivation step $\overset{\delta}{\mapsto}_{\mathcal{G}}$ now records its effect on the store. The effect δ represents the sets of constraints in the store which were propagated or simplified. Goal-based derivation steps can be

[1] Numbered constraints also disambiguate multiple copies in the store but this is rather a side-effect. The main purpose of numbering constraints is to indicate activation and retain the link between active goal constraints and their stored copy.

https://doi.org/10.1017/S147106841000044X Published online by Cambridge University Press



Fig. 3. Example of concurrent goal-based CHR derivation.

executed concurrently if their effects are not in conflict.

(Goal-Concurrency)

$$\frac{\begin{array}{c}\langle G_1 \mid H_{S1} \cup H_{S2} \cup S\rangle \overset{\delta_1}{\rightarrowtail_{||\mathcal{G}}} \langle G_1' \mid H_{S2} \cup S\rangle \\ \langle G_2 \mid H_{S1} \cup H_{S2} \cup S\rangle \overset{\delta_2}{\rightarrowtail_{||\mathcal{G}}} \langle G_2' \mid H_{S1} \cup S\rangle \\ \delta_1 = H_{P1} \backslash H_{S1} \quad \delta_2 = H_{P2} \backslash H_{S2} \\ H_{P1} \subseteq S \quad H_{P2} \subseteq S \quad \delta = H_{P1} \cup H_{P2} \backslash H_{S1} \cup H_{S2}\end{array}}{\begin{array}{l}\langle G_1 \uplus G_2 \uplus G \mid H_{S1} \cup H_{S2} \cup S\rangle \\ \overset{\delta}{\rightarrowtail_{||\mathcal{G}}} \quad \langle G_1' \uplus G_2' \uplus G \mid S\rangle\end{array}}$$

The (Goal-Concurrency) rule, abbreviated ($\| \mathcal{G}$), states that two goal-derivations are not in conflict if their simplification effects are disjoint and the propagated effects are present in the joint store. We will provide more explanations later. Let us continue with our example.

https://doi.org/10.1017/S147106841000044X Published online by Cambridge University Press

Each thread activates one of the two *Get* goals (Steps D1a and D1b). Since both steps involve no rule application, side-effects are empty ($\{\}\backslash\{\}$). Both steps are executed concurrently denoted by the concurrent derivation step (D1a||D2a) $\overset{\{\}\backslash\{\}}{\rightarrowtail_{||\mathscr{G}}}$. Concurrent goal-based execution threads operate on a shared store and their effects will be immediately made visible to other threads. This is important to guarantee exhaustive rule firings.

In the second step (D2a||D2b), both active goals are dropped because there is no complete match for any rule head yet. Next, steps (D3a) and (D3b) activate the last two goal constraints, $Put(1)$ and $Put(2)$. Each active constraint can match with either of the two *Get* constraints in the store. We assume that active constraint $Put(1)\#3$ in step (D4a) matches with $Get(x_1)\#1$, while $Put(2)\#4$ in step (D4b) matches with $Get(x_2)\#2$, corresponding to the side-effects δ_1 and δ_2. This guarantees that steps (D4a) and (D4b) operates on different (nonconflicting) parts of the store. Thus, we can execute them concurrently which yields step (D4a||D4b). Their side-effects are combined as δ. Finally, in step (D5a||D5b) we concurrently solve the two remaining equations by adding them into the store and we are done.

The correctness of our concurrent goal-based semantics is established by showing that all concurrent derivations can be replicated by sequential goal-based executions. We also prove that there is a correspondence between our goal-based CHR semantics with the abstract CHR semantics. This proof generalizes from (Duck 2005) which shows a correspondence between the refined CHR operational semantics and abstract semantics. There are a number of subtle points we came across when developing the concurrent variant of the goal-based semantics. We will postpone a discussion of these issues, as well as a complete formalization of the concurrent goal-based semantics until Section 4. Next, we formally introduce the details of the abstract CHR semantics.

3 Constraint Handling Rules

Figure 4 reviews the essentials of the abstract CHR semantics (Frühwirth 1998). The general form of CHR rules contains propagated heads H_P and simplified heads H_S as well as a guard t_g

$$r \mathbin{@} H_P \backslash H_S \iff t_g \mid B$$

In CHR terminology, a rule with simplified heads only (H_P is empty) is referred to as a *simplification* rule, a rule with propagated heads only (H_S is empty) is referred to as a *propagation* rule. The general form is referred to as a *simpagation* rule.

CHR rules manipulate a global constraint store which is a multiset of constraints. We execute CHRs by exhaustive rewriting of constraints in the store with respect to the given rule system (a finite set of CHR rules), via the derivations $\rightarrowtail$. To avoid ambiguities, we annotate derivations of the abstract semantics with $\mathscr{A}$.

Rule (Rewrite) describes application of a CHR rule r at some instance ϕ. We simply (remove from the store) the matching copies of $\phi(H_S)$ and propagate (keep in the store) the matching copies of $\phi(H_P)$. But this only happens if the instantiated guard $\phi(t_g)$ is entailed by the equations present in the store S, written $Eqs(S) \models \phi(t_g)$.

https://doi.org/10.1017/S147106841000044X Published online by Cambridge University Press

Notations:

$\uplus$	Multi-set union
$\models$	Theoretic entailment
ϕ	Substitution
$\overline{a}$	Set/List of a's

CHR Syntax:

Functions	$f ::= + \mid >\mid \&\& \mid \cdots$
Constants	$v ::= 1 \mid true \mid \cdots$
Terms	$t ::= x \mid f\,\overline{t}$
Predicates	$p ::= Get \mid Put \mid \cdots$
Equations	$e ::= t = t$
CHR constraints	$c ::= p(\overline{t})$
Constraints	$b ::= e \mid c$
CHR Guards	$t_g ::= t$
CHR Heads	$H ::= \overline{c}$
CHR Body	$B ::= \overline{b}$
CHR Rule	$R ::= r @ H \setminus H \iff t_g \mid B$
CHR Store	$S ::= \overline{b}$
CHR Program	$\mathcal{P} ::= \overline{R}$

Abstract Semantics Rules: $\boxed{Store \rightarrowtail_{\mathcal{A}} Store}$

$$\textbf{(Rewrite)} \qquad \frac{(r @ H_P \setminus H_S \iff t_g \mid B) \in \mathcal{P} \text{ such that } \exists \phi \quad Eqs(S) \models \phi \wedge t_g \quad \phi(H_P \uplus H_S) = H'_P \uplus H'_S}{H'_P \uplus H'_S \uplus S \rightarrowtail_{\mathcal{A}} H'_P \uplus \phi(B) \uplus S}$$

$$\textbf{(Concurrency)} \qquad \frac{S \uplus S_1 \rightarrowtail^*_{\mathcal{A}} S \uplus S_2 \quad S \uplus S_3 \rightarrowtail^*_{\mathcal{A}} S \uplus S_4}{S \uplus S_1 \uplus S_3 \rightarrowtail^*_{\mathcal{A}} S \uplus S_2 \uplus S_4}$$

$$\textbf{(Closure)} \qquad \frac{S \rightarrowtail_{\mathcal{A}} S'}{S \rightarrowtail^*_{\mathcal{A}} S'} \qquad \frac{S \rightarrowtail_{\mathcal{A}} S' \quad S' \rightarrowtail^*_{\mathcal{A}} S''}{S \rightarrowtail^*_{\mathcal{A}} S''}$$

where $Eqs(S) = \{e \mid e \in S, e \text{ is an equation}\}$

Fig. 4. Abstract CHR semantics.

In case of a propagation rule we need to avoid infinite re-propagation. We refer to (Abdennadher 1997; Duck 2005) for details. Rule (Concurrency), introduced in (Frühwirth 2005), states that rules can be applied concurrently as long as they simplify on nonoverlapping parts of the store.

Definition 3.1 (*Nonoverlapping Rule Application*)
Two applications of the rule instances $r \,@\, H_P \setminus H_S \iff t_g \mid B$ and $r' \,@\, H'_P \setminus H'_S \iff t'_g \mid B'$ in store S are said to be nonoverlapping if and only if they simplify unique parts of S (i.e. $H_S, H'_S \subseteq S$ and $H_S \cap H'_S = \emptyset$).

https://doi.org/10.1017/S147106841000044X Published online by Cambridge University Press

Notations:

$\uplus$	Multi-set union
$\cup$	Set union
$\models$	Theoretic entailment
ϕ	Substitution
$\overline{a}$	Set/List of a's

CHR Syntax:

Functions	$f ::= + \mid >\mid \mid \&\& \mid \cdots$
Constants	$v ::= 1 \mid true \mid \cdots$
Terms	$t ::= x \mid f\,\overline{t}$
Predicates	$p ::= Get \mid Put \mid \cdots$
Equations	$e ::= t = t$
CHR Constraints	$c ::= p(\overline{t})$
Constraints	$b ::= e \mid c$
CHR Guards	$t_g ::= t$
CHR Heads	$H ::= \overline{c}$
CHR Body	$B ::= \overline{b}$
CHR Rule	$R ::= r @ H \setminus H \iff t_g \mid B$
CHR Program	$\mathcal{P} ::= \overline{R}$
Num Constraint	$nc ::= c\#i$
Goal Constraint	$g ::= c \mid e \mid nc$
Stored Constraint	$sc ::= nc \mid e$
CHR Num Store	$Sn ::= \overline{sc}$
CHR Goals	$G ::= \overline{g}$
CHR State	$\sigma ::= \langle G, Sn \rangle$
Side Effects	$\delta ::= Sn \setminus Sn$

Fig. 5. CHR goal-based syntax.

The two last (Closure) rules simply specify the transitive application of CHR rules.

4 Concurrent goal-based Constraint Handling Rules operational semantics

We present the formal details of the concurrent goal-based CHR semantics. Figure 5 describes the necessary syntactic extensions. Because constraints in the store now have unique identifiers, we treat the store as a set (as opposed to a multiset) and use set union $\cup$. Goals are still treated as multisets because they can contain multiple copies of (unnumbered) CHR constraints. The actual semantics is given in two parts. Figure 6 describes the single-step execution part whereas Figure 7 introduces the concurrent execution part. The first part is a generalization of an earlier goal-based description (Duck 2005) whereas the second (concurrent) part is novel.

We first discuss the single-step derivation steps in Figure 6. A derivation step $\sigma \overset{\delta}{\rightarrowtail_g} \sigma'$ maps the CHR state σ to σ' with some side-effect δ. δ represents the constraints that where propagated or simplified during rule application. Hence derivation steps that do not involve rule application ((Activate) and (Drop)) contain no side-effects (i.e. $\{\} \setminus \{\}$). We will omit side-effects δ as and when it is not relevant to our discussions. We ignore the (Solve) step for the moment. In (Activate), we activate a goal CHR constraint by assigning it a fresh unique identifier and adding

https://doi.org/10.1017/S147106841000044X Published online by Cambridge University Press

$$\boxed{\langle Goal \mid Store\rangle \overset{\delta}{\rightarrowtail_{\mathcal{G}}} \langle Goal \mid Store\rangle}$$

(Solve)
$$\frac{W = WakeUp(e, Sn)}{\langle \{e\} \uplus G \mid Sn\rangle \overset{W\setminus\{\}}{\rightarrowtail_{\mathcal{G}}} \langle W \uplus G \mid \{e\} \cup Sn\rangle}$$

(Activate)
$$\frac{i \text{ is a fresh identifier}}{\langle \{c\} \uplus G \mid Sn\rangle \overset{\{\}\setminus\{\}}{\rightarrowtail_{\mathcal{G}}} \langle \{c\#i\} \uplus G \mid \{c\#i\} \cup Sn\rangle}$$

(Simplify)
$$\frac{\begin{array}{c}(r@H'_P\backslash H'_S \iff t_g \mid B') \in \mathcal{P} \text{ such that}\\ \exists\phi \quad Eqs(Sn) \models \phi \wedge t_g \quad \phi(H'_P) = DropIds(H_P)\\ \phi(H'_S) = \phi(\{c\} \uplus DropIds(H_S))\\ \delta = H_P\backslash\{c\#j\} \cup H_S\end{array}}{\begin{array}{c}\langle \{c\#j\} \uplus G \mid \{c\#j\} \cup H_P \cup H_S \cup Sn\rangle\\ \overset{\delta}{\rightarrowtail_{\mathcal{G}}} \quad \langle \phi(B') \uplus G \mid H_S \cup Sn\rangle\end{array}}$$

(Propagate)
$$\frac{\begin{array}{c}(r@H'_P\backslash H'_S \iff t_g \mid B') \in \mathcal{P} \text{ such that}\\ \exists\phi \quad Eqs(Sn) \models \phi \wedge t_g \quad \phi(H'_S) = DropIds(H_S)\\ \phi(H'_P) = \phi(\{c\} \uplus DropIds(H_P))\\ \delta = \{c\#j\} \cup H_P\backslash H_S\end{array}}{\begin{array}{c}\langle \{c\#j\} \uplus G \mid \{c\#j\} \cup H_P \cup H_S \cup Sn\rangle\\ \overset{\delta}{\rightarrowtail_{\mathcal{G}}} \quad \langle \phi(B') \uplus \{c\#j\} \uplus G \mid \{c\#j\} \cup H_P \cup Sn\rangle\end{array}}$$

(Drop)
$$\frac{\text{(Simplify) and (Propagate) does not apply on } c\#j \text{ in } Sn}{\langle \{c\#j\} \uplus G \mid Sn\rangle \overset{\{\}\setminus\{\}}{\rightarrowtail_{\mathcal{G}}} \langle G \mid Sn\rangle}$$

where
$$\begin{array}{lcl} Eqs(S) & = & \{e \mid e \in S, e \text{ is an equation}\}\\ DropIds(Sn) & = & \{c \mid c\#i \in Sn\} \uplus \{e \mid e \in Sn, e \text{ is an equation}\}\\ WakeUp(e, Sn) & = & \{c\#i \mid c\#i \in Sn \wedge \phi \text{ m.g.u. of } Eqs(Sn) \wedge\\ & & \theta \text{ m.g.u. of } Eqs(Sn \cup \{e\}) \wedge \phi(c) \neq \theta(c)\}\end{array}$$

Fig. 6. Goal-based CHR semantics (single-step execution).

it to the store. Rewrite rules are executed in steps (Simplify) and (Propagate). We distinguish whether the rewrite rule is executed on a simplified or propagated active (goal) constraint *c#i*. For both cases, we seek for the missing partner constraints in the store for some matching substitution ϕ. The auxiliary function *DropIds* ignores the unique identifiers of numbered constraints. They do not matter when finding a rule head match. The guard t_g must be entailed by the primitive (here equational) store constraints under the substitution ϕ.

https://doi.org/10.1017/S147106841000044X Published online by Cambridge University Press

In case of a simplified goal, step (Simplify), we apply the rule instance of r by deleting all simplified matching constraints H_S and adding the rule body instance $\phi(B)$ into the goals. Since $c\#i$ is simplified, we drop $c\#i$ from the goals as it does not exist in the store any more. In case of a propagated goal, step (Propagate), $c\#i$ remains in the goal set as well in the store and thus can possibly fire further rules instances. For both (Simplify) and (Propagate) derivation step, say $\sigma \overset{H_P \backslash H_S}{\mapsto_{\mathscr{G}}} \sigma'$, we record as side-effect the numbered constraints in the store that were propagated (H_P) or simplified (H_S) during the derivation step. We will elaborate on the purpose of side-effects when we introduce the concurrent part of the semantics.

In step (Drop), we remove an active constraint from the set of goals, if the constraint failed to trigger any CHR rule.

Rule (Solve) moves an equation goal e into the store and *wakes up* (reactivates) any numbered constraint in the store which can possibly trigger further CHR rules due to the presence of e. Here is a simple example to show why reactivation is necessary.

$$r1 \ @ \ A(x), B(x) \Longleftrightarrow C(x)$$

$$\begin{array}{lll} & & \langle \{a = 2\} \mid \{A(a)\#1, B(2)\#2\} \rangle \\ \text{(Solve)} & \overset{\{A(2)\#1\} \backslash \{\}}{\mapsto_{\mathscr{G}}} & \langle \{A(2)\#1\} \mid \{A(2)\#1, B(2)\#2, a = 2\} \rangle \\ \text{(Simp } r1\text{)} & \overset{\{\} \backslash \{A(2)\#1, B(2)\#2\}}{\mapsto_{\mathscr{G}}} & \langle \{C(2)\} \mid \{a = 2\} \rangle \\ \ldots & & \end{array}$$

For clarity, we normalize all constraints in the store once an equation is added. Prior to addition of $a = 2$, $A(a)\#1, B(2)\#2$ cannot fire rule $r1$. After adding $a = 2$ however, we can normalize $A(a)\#1$ to $A(2)\#2$, which can now fire $r1$ with $B(2)\#2$. To guarantee exhaustive rule firings, we reactivate $A(2)\#2$ by adding it back to the set of goals. $WakeUp(e, Sn)$ represents a conservative approximation of the to be reactivated constraints (Duck 2005). Note that we treat reactivated constraints as propagated constraints in the side-effects.

Figure 7 presents the concurrent part of the goal-based operational semantics. In the (Lift) step, we turn a sequential goal-based derivation into a concurrent derivation. Note that side-effects are retained. Step (Goal Concurrency) joins together two concurrent derivations operating on a shared store, if their rewriting side-effects δ_1 and δ_2 are nonoverlapping as defined below.

Definition 4.1 (*Nonoverlapping Rewriting Side-Effects*)
Two rewriting side-effects $\delta_1 = H_{P1} \backslash H_{S1}$ and $\delta_2 = H_{P2} \backslash H_{S2}$ are said to be nonoverlapping, if and only if $H_{S1} \cap (H_{P2} \cup H_{S2}) = \{\}$ and $H_{S2} \cap (H_{P1} \cup H_{S1}) = \{\}$

Concurrent derivations with nonoverlapping side-effects essentially simplify distinct constraints in the store, as well as propagate constraints which are not simplified by one another. The (Goal Concurrency) step expresses nonoverlapping side-effects by structurally enforcing that simplified constraints H_{S1} and H_{S2} match distinct parts of the store, while propagated constraints H_{P1} and H_{P2} are found in the shared part of the store S not modified by both concurrent derivations. In the resulting concurrent derivation, the side-effects δ_1 and δ_2 are composed by the union of the propagate and simplify components respectively, forming δ.

https://doi.org/10.1017/S147106841000044X Published online by Cambridge University Press

$$\boxed{\langle Goal \mid Store\rangle \overset{\delta}{\rightarrowtail}_{||\mathcal{G}} \langle Goal \mid Store\rangle}$$

(Lift)
$$\frac{\langle G \mid Sn\rangle \overset{\delta}{\rightarrowtail}_{\mathcal{G}} \langle G' \mid Sn'\rangle}{\langle G \mid Sn\rangle \overset{\delta}{\rightarrowtail}_{||\mathcal{G}} \langle G' \mid Sn'\rangle}$$

(Goal Concurrency)
$$\frac{\begin{array}{c}\langle G_1 \mid H_{S1} \cup H_{S2} \cup S\rangle \overset{\delta_1}{\rightarrowtail}_{||\mathcal{G}} \langle G_1' \mid H_{S2} \cup S\rangle \\ \langle G_2 \mid H_{S1} \cup H_{S2} \cup S\rangle \overset{\delta_2}{\rightarrowtail}_{||\mathcal{G}} \langle G_2' \mid H_{S1} \cup S\rangle \\ \delta_1 = H_{P1} \backslash H_{S1} \quad \delta_2 = H_{P2} \backslash H_{S2} \\ H_{P1} \subseteq S \quad H_{P2} \subseteq S \quad \delta = H_{P1} \cup H_{P2} \backslash H_{S1} \cup H_{S2}\end{array}}{\begin{array}{l}\langle G_1 \uplus G_2 \uplus G \mid H_{S1} \cup H_{S2} \cup S\rangle \\ \overset{\delta}{\rightarrowtail}_{||\mathcal{G}} \quad \langle G_1' \uplus G_2' \uplus G \mid S\rangle\end{array}}$$

(Closure)
$$\frac{\sigma \overset{\delta}{\rightarrowtail}_{||\mathcal{G}} \sigma'}{\sigma \rightarrowtail^{*}_{||\mathcal{G}} \sigma'} \qquad \frac{\sigma \overset{\delta}{\rightarrowtail}_{||\mathcal{G}} \sigma' \quad \sigma' \rightarrowtail^{*}_{||\mathcal{G}} \sigma''}{\sigma \rightarrowtail^{*}_{||\mathcal{G}} \sigma''}$$

Fig. 7. Goal-based CHR semantics (concurrent part).

An immediate consequence is that we can execute k derivations concurrently by stacking them together as long as all side-effects are mutually nonoverlapping. The following lemma summarizes this observation.

Lemma 1 (*k-Concurrency*)
For any finite k of mutually nonoverlapping concurrent derivations,

$$\begin{array}{c}\langle G_1 \mid H_{S1} \cup \cdots \cup H_{Si} \cup \cdots \cup H_{Sk} \cup S\rangle \overset{H_{P1} \backslash H_{S1}}{\rightarrowtail}_{||\mathcal{G}} \langle G_1' \mid \{\} \cup \cdots \cup H_{Si} \cup \cdots \cup H_{Sk} \cup S\rangle \\ \cdots \\ \langle G_i \mid H_{S1} \cup \cdots \cup H_{Si} \cup \cdots \cup H_{Sk} \cup S\rangle \overset{H_{Pi} \backslash H_{Si}}{\rightarrowtail}_{||\mathcal{G}} \langle G_i' \mid H_{S1} \cup \cdots \cup \{\} \cup \cdots \cup H_{Sk} \cup S\rangle \\ \cdots \\ \langle G_k \mid H_{S1} \cup \cdots \cup H_{Si} \cup \cdots \cup H_{Sk} \cup S\rangle \overset{H_{Pk} \backslash H_{Sk}}{\rightarrowtail}_{||\mathcal{G}} \langle G_k' \mid H_{S1} \cup \cdots \cup H_{Si} \cup \cdots \cup \{\} \cup S\rangle \\ H_{P1} \subseteq S \cdots H_{Pi} \subseteq S \cdots H_{Pk} \subseteq S \\ \delta = H_{P1} \cup \cdots \cup H_{Pi} \cup \cdots \cup H_{Pk} \backslash H_{S1} \cup \cdots \cup H_{Si} \cup \cdots \cup H_{Sk}\end{array}$$

$$\begin{array}{l}\langle G_1 \uplus \cdots \uplus G_i \uplus \cdots \uplus G_k \uplus G \mid H_{S1} \cup \cdots \cup H_{Si} \cup \cdots \cup H_{Sk} \cup S\rangle \\ \overset{\delta}{\rightarrowtail}_{||\mathcal{G}} \quad \langle G_1' \uplus \cdots \uplus G_i' \uplus \cdots \uplus G_k' \uplus G \mid S\rangle\end{array}$$

we can decompose this into $k-1$ applications of the (pair-wise) (Goal Concurrency) derivation step.

The (Closure) step defines transitive application of the concurrent goal-based derivation. Because side-effect labels are only necessary for the (Goal Concurrency) step, we drop the side-effects in transitive derivations.

https://doi.org/10.1017/S147106841000044X Published online by Cambridge University Press

Any concurrent goal-based derivation can be reproduced in the abstract CHR semantics. This correspondence result is important to make use of the concurrent goal-based semantics as a more systematic execution scheme for CHR. We will formally verify this as well as other results in the up-coming Section 4.2. First, we give an in-depth discussion of the more subtle aspects of the concurrent goal-based semantics.

4.1 Discussion

Most of the issues we encounter are related to the problem of exhaustive rule firings. For brevity, we omit side-effects in derivation steps in the following examples as they do not matter.

Goal storage, shared store, and single-step Execution: Each of these issues affect (exhaustive) rule firings. We first consider goal storage. Suppose we would only store goals after execution (rule head matching). That is, we do not add the goals into the store during (Activate) step, but only during the (Drop) step.

$$\text{(Activate')}\quad \frac{i \text{ is a fresh identifier}}{\langle \{c\} \uplus G \mid Sn\rangle \rightarrowtail_{\mathscr{g}} \langle \{c\#i\} \uplus G \mid Sn\rangle}$$

$$\text{(Drop')}\quad \frac{\text{(Simplify) and (Propagate) does not apply on } c\#i \text{ in } Sn}{\langle \{c\#i\} \uplus G \mid Sn\rangle \rightarrowtail_{\mathscr{g}} \langle G \mid \{c\#i\} \cup Sn\rangle}$$

Then, for the CHR program

$$r1 \mathbin{@} A(x), B(y) \Longleftrightarrow C(x, y)$$

we obtain the following derivation

$$\langle \{A(1), B(2)\} \mid \{\}\rangle$$

$$\frac{\begin{array}{ll}\text{(Activate')} & \langle \{A(1)\} \mid \{\}\rangle \rightarrowtail_{\|\mathscr{g}} \langle \{A(1)\#1\} \mid \{\}\rangle \\ & \| \\ \text{(Activate')} & \langle \{B(2)\} \mid \{\}\rangle \rightarrowtail_{\|\mathscr{g}} \langle \{B(2)\#2\} \mid \{\}\rangle\end{array}}{\langle \{A(1), B(2)\} \mid \{\}\rangle \rightarrowtail_{\|\mathscr{g}} \langle \{A(1)\#1, B(2)\#2\} \mid \{\}\rangle}$$

$$\frac{\begin{array}{ll}\text{(Drop')} & \langle \{A(1)\#1\} \mid \{\}\rangle \rightarrowtail_{\|\mathscr{g}} \langle \{\} \mid \{A(1)\#1\}\rangle \\ & \| \\ \text{(Drop')} & \langle \{B(2)\#2\} \mid \{\}\rangle \rightarrowtail_{\|\mathscr{g}} \langle \{\} \mid \{B(2)\#2\}\rangle\end{array}}{\langle \{A(1)\#1, B(2)\#2\} \mid \{\}\rangle \rightarrowtail_{\|\mathscr{g}} \langle \{\} \mid \{A(1)\#1, B(2)\#2\}\rangle}$$

https://doi.org/10.1017/S147106841000044X Published online by Cambridge University Press

Initially, both goals $A(1)$ and $B(2)$ are concurrently activated. Since (Activate') does not store goals immediately, both active goals are not visible to each other in the store. Hence, we wrongfully apply the (Drop') step for both goals. However, there is clearly a complete rule head match $A(1)\#1, B(2)\#2$.

Next, we investigate the shared store issue. Suppose we allow for concurrent executions on (nonshared) split stores. Then, the following derivation is possible.

$$r1 \mathbin{@} A, B \iff C \qquad r2 \mathbin{@} D, E \iff F$$

$$\begin{array}{ll} \text{(Drop)} & \langle \{A\#3\} \mid \{A\#3, E\#2\}\rangle \rightarrowtail_{||_{\mathcal{G}}} \langle \{\} \mid \{A\#3, E\#2\}\rangle \\ \text{(Drop)} & \langle \{D\#4\} \mid \{B\#1, D\#4\}\rangle \rightarrowtail_{||_{\mathcal{G}}} \langle \{\} \mid \{B\#1, D\#4\}\rangle \end{array}$$

$$\langle \{A\#3, D\#4\} \mid \{A\#3, B\#1, D\#4, E\#2\}\rangle \rightarrowtail_{||_{\mathcal{G}}} \langle \{\} \mid \{A\#3, B\#1, D\#4, E\#2\}\rangle$$

The resulting store is a final store, there are no more goals left. However, if we consider the entire store $\{A\#3, E\#2, B\#1, D\#4\}$, it is clearly that goal $A\#3$ can execute rule $r1$ and goal $D\#4$ can execute rule $r2$. We conclude that splitting of the store leads to "stuck" states. We fail to exhaustively fire CHR rules.

For similar reasons, we demand that when joining concurrent executions, each individual execution can only make a single step. Otherwise, we encounter again a stuck state.

$$r1 \mathbin{@} A, B \iff C$$

$$\begin{array}{ll} (P1) & \langle \{A\} \mid \{\}\rangle \rightarrowtail_{||_{\mathcal{G}}} \langle \{A\#2\} \mid \{A\#2\}\rangle \rightarrowtail_{||_{\mathcal{G}}} \langle \{\} \mid \{A\#2\}\rangle \\ (P2) & \langle \{B\} \mid \{\}\rangle \rightarrowtail_{||_{\mathcal{G}}} \langle \{B\#3\} \mid \{B\#3\}\rangle \rightarrowtail_{||_{\mathcal{G}}} \langle \{\} \mid \{B\#3\}\rangle \end{array}$$

$$\langle \{A, B\} \mid \{\}\rangle \rightarrowtail^{*}_{||_{\mathcal{G}}} \langle \{\} \mid \{A\#2, B\#3\}\rangle$$

The sequence of derivation steps (P1) first activates A which is then dropped. Similarly, (P2) activates B which is then dropped as well which then leads to the stuck state $\langle \{\} \mid \{A\#2, B\#3\}\rangle$. We clearly missed to fire rule $r1$. This shows that single-step concurrent execution are essential to guarantee that newly added constraints are visible to all concurrent active goals, hence we have exhaustive rule firings in the goal-based semantics.

The underlying reason for nonexhaustive firing of rules is that the goal-based semantics is not monotonic in its store argument. However, execution is monotonic in the goal argument which leads us to the next issue.

Lazy matching and asynchronous goal execution: When executing goals, we lazily compute only matches that contain the specific goal and immediately apply such matches without concerning any further matches. For instance, consider the following CHR program and goal-based derivation:

$$r0 \mathbin{@} A(x), B(y) \iff D(x, y)$$

$$\begin{array}{ll} & \langle \{A(1)\#4\} \uplus \{A(2), A(3)\} \mid \{B(2)\#1, B(3)\#2, B(4)\#3, A(1)\#4\}\rangle \\ \rightarrowtail_{\mathcal{G}} & \langle \{D(1,2)\} \uplus \{A(2), A(3)\} \mid \{B(3)\#2, B(4)\#3\}\rangle \end{array}$$

https://doi.org/10.1017/S147106841000044X Published online by Cambridge University Press

We have applied the rule instance $A(1)\#4, B(2)\#1$ independently of the existence of the other goals (i.e. $\{A(2), A(3)\}$). In the literature, such a matching scheme is known as a *lazy* matching scheme, and often implemented by variants of the LEAPS algorithm (Miranker *et al.* 1990).

Lazy matching in the goal-based semantics is possible only because the goal-based semantics is monotonic with respect to the set of goals. The following illustrates this monotonicity property of goals:

$$\begin{array}{rl} & \langle \{A(1)\#4\} \mid \{B(2)\#1, B(3)\#2, B(4)\#3, A(1)\#4\} \rangle \\ \rightarrowtail_{\mathcal{G}} & \langle \{D(1,2)\} \mid \{B(3)\#2, B(4)\#3\} \rangle \end{array}$$

$$\begin{array}{rl} & \langle \{A(1)\#4\} \uplus \{A(2), A(3)\} \mid \{B(2)\#1, B(3)\#2, B(4)\#3, A(1)\#4\} \rangle \\ \rightarrowtail_{\mathcal{G}} & \langle \{D(1,2)\} \uplus \{A(2), A(3)\} \mid \{B(3)\#2, B(4)\#3\} \rangle \end{array}$$

$$\frac{\langle G \mid Sn \rangle \rightarrowtail_{\mathcal{G}} \langle G' \mid Sn' \rangle}{\langle G \uplus G'' \mid Sn \rangle \rightarrowtail_{\mathcal{G}} \langle G' \uplus G'' \mid Sn' \rangle}$$

The above property essentially states that we can execute goals G without prior knowledge of goals G''. Because of monotonicity, we are guaranteed that future executions of G'' will not invalidate them.

Monotonicity of the goals also allows us to execute goals asynchronously. For instance, consider the following:

$$r1 \mathbin{@} A(x), B(y) \Longleftrightarrow C(x, y)$$

$$\begin{array}{rl} & \langle \{A(1)\#1\} \mid \{A(1)\#1, B(2)\#2\} \cup \{A(3)\#3, B(4)\#4\} \rangle \\ \overset{\delta_1}{\rightarrowtail}_{\|\mathcal{G}} & \langle \{C(1,2)\} \mid \{\} \cup \{A(3)\#3, B(4)\#4\} \rangle \\ & \langle \{A(3)\#3\} \mid \{A(1)\#1, B(2)\#2\} \cup \{A(3)\#3, B(4)\#4\} \rangle \\ \overset{\delta_2}{\rightarrowtail}_{\|\mathcal{G}} & \langle \{C(3,4)\} \mid \{A(1)\#1, B(2)\#2\} \cup \{\} \rangle \\ \multicolumn{2}{c}{\delta_1 = \{\} \backslash \{A(1)\#1, B(2)\#2\} \quad \delta_2 = \{\} \backslash \{A(3)\#3, B(4)\#4\}} \\ \multicolumn{2}{c}{\delta = \{\} \backslash \{A(1)\#1, B(2)\#2, A(3)\#3, B(4)\#4\}} \end{array}$$

$$\begin{array}{rl} & \langle \{A(1)\#1\} \uplus \{A(3)\#3\} \mid \{A(1)\#1, B(2)\#2\} \cup \{A(3)\#3, B(4)\#4\} \rangle \\ \overset{\delta}{\rightarrowtail}_{\|\mathcal{G}} & \langle \{C(1,2)\} \uplus \{C(3,4)\} \mid \{\} \cup \{\} \rangle \end{array}$$

The above describes the concurrent execution of goals $A(1)\#1$ and $A(3)\#3$. Notice that in the derivations of the premise, we can ignore all goals which are not relevant to the derivation. For instance, execution of $A(1)\#1$ does not need goal $A(3)\#3$ to be visible, hence the goals effectively executes asynchronously. Goals do however, implicitly "synchronize" via the shared store. Namely, concurrent derivations must be chosen such that rewrite side-effects involve distinct parts of the store.

https://doi.org/10.1017/S147106841000044X Published online by Cambridge University Press

4.2 Correspondence results

We formally verify that the concurrent goal-based semantics is in exact correspondence to the abstract CHR semantics when it comes to termination and exhaustive rule firings. Detailed proofs are given in the Appendix. In the main text, we provide key lemmas and proof sketches. We first introduce some elementary definitions before stating the formal results.

Definitions: The first two definitions concern the abstract CHR semantics. A store is final if no further rules are applicable.

Definition 4.2 (*Final Store*)
A store S is known as a final store, denoted $Final_{\mathscr{A}}(S)$ if and only if no more CHR rules applies on it (i.e. $\neg\exists S'$ such that $S \rightarrowtail_{\mathscr{A}} S'$).

A CHR program terminates if all derivations lead to a final store in a finite number of states.

Definition 4.3 (*Terminating CHR Programs*)
A CHR program $\mathscr{P}$ is said to be terminating, if and only if for any CHR store S, all derivations starting from S are finite.

Next, we introduce some definitions in terms of the goal-based semantics. In an initial state, all constraints are goals and the store is empty. Final states are states which no longer have any goals. We will prove the exhaustiveness of the goal-based semantics by proving a correspondence between final stores in the abstract semantics and final states of the goal-based semantics

Definition 4.4 (*Initial and Final CHR States*)
An initial CHR state is a CHR state of the form $\langle G \mid \{\}\rangle$ where G contains no numbered constraints ($c\#n$), while a final CHR state is of the form $\langle\{\} \mid Sn\rangle$

A state is reachable if there exists a (sequential) goal-based sequence of derivations to this state. We write $\rightarrowtail^*_{\mathscr{g}}$ to denote the transitive closure of $\rightarrowtail_{\mathscr{g}}$.

Definition 4.5 (*Sequentially Reachable CHR states*)
For any CHR program $\mathscr{P}$, a CHR state $\langle G' \mid Sn'\rangle$ is said to be sequentially reachable by $\mathscr{P}$ if and only if there exists some initial CHR state $\langle G \mid \{\}\rangle$ such that $\langle G \mid \{\}\rangle \rightarrowtail^*_{\mathscr{g}} \langle G' \mid Sn'\rangle$.

4.2.1 Correspondence of derivations

We build a correspondence between the abstract semantics and the concurrent goal-based semantics. We begin with Theorem 2, which states the correspondence of the (sequential) goal-based semantics.

Theorem 2 (*Correspondence of Sequential Derivations*)
For any reachable CHR state $\langle G \mid Sn\rangle$, CHR state $\langle G' \mid Sn'\rangle$ and CHR program $\mathscr{P}$,

$$\begin{array}{ll} \text{if} & \langle G \mid Sn\rangle \rightarrowtail^*_{\mathscr{g}} \langle G' \mid Sn'\rangle \\ \text{then} & (NoIds(G) \uplus DropIds(Sn)) = (NoIds(G') \uplus DropIds(Sn')) \quad \vee \\ & (NoIds(G) \uplus DropIds(Sn)) \rightarrowtail^*_{\mathscr{A}} (NoIds(G') \uplus DropIds(Sn')) \end{array}$$

where $NoIds = \{c \mid c \in G, c \text{ is a CHR constraint}\} \uplus \{e \mid e \in G, e \text{ is an equation}\}$

https://doi.org/10.1017/S147106841000044X Published online by Cambridge University Press

The above result guarantees that any sequence of sequential goal-based derivations starting from a reachable CHR state either yields equivalent CHR abstract stores (due to goal-based behavior not captured by the abstract semantics, namely (Solve) (Activate), (Drop)) or corresponds to a derivation in the abstract semantics (due to rule application). A goal-based semantics state $\langle G \mid Sn \rangle$ is related to an abstract semantics store by removing all numbered constraints in G and unioning it with constraints in Sn without their identifiers. The theorem and its proof is a generalization of an earlier result given in Duck (2005).

We formalize the observation that the goal context can be extended without interfering with previous goal executions.

Lemma 2 (*Monotonicity of Goals in Goal-based Semantics*)
For any goals G,G' and G'' and CHR store Sn and Sn', If $\langle G \mid Sn \rangle \rightarrowtail^*_{\mathscr{G}} \langle G' \mid Sn' \rangle$ then $\langle G \uplus G'' \mid Sn \rangle \rightarrowtail^*_{\mathscr{G}} \langle G' \uplus G'' \mid Sn' \rangle$.

Next, we state that given any goal-based derivation with side-effects δ, we can safely ignore any constraints (represented by S_2) in the store which is not part of δ.

Lemma 3 (*Isolation of Goal-based Derivations*)
If $\langle G \mid H_P \cup H_S \cup S_1 \cup S_2 \rangle \overset{H_P \setminus H_S}{\rightarrowtail_{\mathscr{G}}} \langle G' \mid H_P \cup S_1' \cup S_2 \rangle$
then $\langle G \mid H_P \cup H_S \cup S_1 \rangle \overset{H_P \setminus H_S}{\rightarrowtail_{\mathscr{G}}} \langle G' \mid H_P \cup S_1' \rangle$

Lemma 3 can be straightforwardly extended to multiple derivation steps. This is stated in Lemma 4.

Lemma 4 (*Isolation of Transitive Goal-based Derivations*)
If $\langle G \mid H_P \cup H_S \cup S_1 \cup S_2 \rangle \rightarrowtail^*_{\mathscr{G}} \langle G' \mid H_P \cup S_1' \cup S_2 \rangle$with side-effects $\delta = H_P \setminus H_S$ then $\langle G \mid H_P \cup H_S \cup S_1 \rangle \rightarrowtail^*_{\mathscr{G}} \langle G' \mid H_P \cup S_1' \rangle$

The next states that any concurrent derivation starting from a reachable CHR state can be replicated by a sequence of sequential goal-based derivations. Lemma 5 is the first step to prove the correspondence of concurrent goal-based derivations.

Lemma 5 (*Sequential Reachability of Concurrent Derivation Steps*)
For any sequentially reachable CHR state σ, CHR state σ' and rewriting side-effects δ if $\sigma \overset{\delta}{\rightarrowtail_{||\mathscr{G}}} \sigma'$ then σ' is sequentially reachable, $\sigma \rightarrowtail^*_{\mathscr{G}} \sigma'$ with side-effects δ.

Proof
(Sketch) Via Lemma 1, we can always reduce k mutually nonoverlapping concurrent derivations into several applications of the (Goal Concurrency) step. Hence we can prove Lemma 5 by structural induction over the concurrent goal-based derivation steps (Lift) and (Goal Concurrency) where we use Lemmas 2 and 4 to show that concurrent derivations can always be replicated by a sequence of sequential goal-based derivations. □

Theorem 3 (*Sequential Reachability of Concurrent Derivations*)
For any initial CHR state σ, CHR state σ' and CHR Program $\mathscr{P}$, if $\sigma \rightarrowtail^*_{||\mathscr{G}} \sigma'$ then $\sigma \rightarrowtail^*_{\mathscr{G}} \sigma'$.

https://doi.org/10.1017/S147106841000044X Published online by Cambridge University Press

The above follows directly from Lemma 5 by converting each single step concurrent derivation into a sequence of sequential derivations, and showing their composibility.

From Theorem 2 and 3, we have the following corollary, which states the correspondence between concurrent goal-based CHR derivations and abstract CHR derivations.

Corollary 1 (*Correspondence of Concurrent Derivations*)
For any reachable CHR state $\langle G \mid Sn\rangle$, CHR state $\langle G' \mid Sn'\rangle$ and CHR program $\mathscr{P}$,

$$\begin{array}{ll} \text{if} & \langle G \mid Sn\rangle \rightarrowtail^*_{||\mathscr{G}} \langle G' \mid Sn'\rangle \\ \text{then} & (NoIds(G) \uplus DropIds(Sn)) = (NoIds(G') \uplus DropIds(Sn')) \quad \vee \\ & (NoIds(G) \uplus DropIds(Sn)) \rightarrowtail^*_{\mathscr{A}} (NoIds(G') \uplus DropIds(Sn')) \end{array}$$

where $NoIds = \{c \mid c \in G, c \text{ is a CHR constraint}\} \uplus \{e \mid e \in G, e \text{ is an equation}\}$

4.2.2 *Correspondence of termination*

We show that all derivations from an initial state to final states in the concurrent goal-based semantics correspond to some derivation from a store to a final store in the abstract semantics. We first define rule head instances:

Definition 4.6 (*Rule head instances*)
For any CHR state $\sigma = \langle G, Sn\rangle$ and CHR program $\mathscr{P}$, any $(H_P \cup H_S) \subseteq Sn$ is known as a rule head instance of σ, if and only if $\exists(r \mathbin{@} H'_P \backslash H'_P \iff t_g \mid B) \in \mathscr{P}, \exists \phi$ $Eqs(Sn) \models \phi \wedge t_g$ and $\phi(H'_P \uplus H'_S) = DropIds(H_P \cup H_S)$.

Definition 4.7 (*Active rule head instances*)
For any CHR state $\sigma = \langle G, Sn\rangle$ and CHR program $\mathscr{P}$, a rule head instance H of σ is said to be *active* if and only if there exists at least one $c\#i \in G$ such that $c\#i \in H$.

Rule head instances (Definition 4.6) are basically minimal subsets of the store which matches a rule head. Active rule head instance (Definition 4.7) additional have at least one of it is numbered constraint $c\#i$ in the goals as well. Therefore, by the definition of the goal-based semantics, active rule head instances will eventually be triggered by either the (Simplify) or (Propagate) derivation steps.

Lemma 6 (*Rule instances in reachable states are always active*)
For any reachable CHR state $\langle G \mid Sn\rangle$, any rule head instance $H \subseteq Sn$ must be active. i.e. $\exists c\#i \in H$ such that $c\#i \in G$.

Lemma 6 shows that all rule head instances in reachable states are always active. This means that by applying the semantics steps in any way, we must eventually apply the rule head instances as long as all it is constraints remain in the store.

Theorem 4 states that termination of a concurrent goal-based derivation corresponds to termination in the abstract semantics. This is of course, provided that the CHR program is terminating.

https://doi.org/10.1017/S147106841000044X Published online by Cambridge University Press

Theorem 4 (*Correspondence of Termination*)
For any initial CHR state $\langle G, \{\}\rangle$, final CHR state $\langle\{\}, Sn\rangle$ and terminating CHR program $\mathscr{P}$,

$$\text{if } \langle G \mid \{\}\rangle \rightarrowtail^*_{\|\mathscr{G}} \langle\{\} \mid Sn\rangle$$
$$\text{then } G \rightarrowtail^*_{\mathscr{A}} \mathit{DropIds}(Sn) \text{ and } \mathit{Final}_{\mathscr{A}}(\mathit{DropIds}(Sn))$$

We prove this theorem by first using Theorem 3 which guarantees that a concurrent goal-based derivation from an initial state to a final state corresponds to some abstract semantics derivation. We next show that final states correspond to final stores in the abstract semantics. This is done by contradiction, showing that assuming otherwise contradicts with Lemma 6.

4.3 Concurrent Constraint Handling Rules optimizations

In the sequential setting, there exist a wealth of optimizations (Duck 2005; Schrijvers 2005; Sneyers *et al*. 2005) to speed up the execution of CHR. Fortunately, many of these methods are still largely applicable to our concurrent goal-based variant as we discuss in the following. For the remainder, we assume that each goal (thread) tries the CHR rules from top-to-bottom to match the rule execution order assumed in Duck (2005), Schrijvers (2005), and Sneyers *et al.* (2005).

Basic constraint indexing like lookups via hashtables are still applicable with minor adaptations. For instance, the underlying hashtable implementation must be thread safe. Consider the following example:

$$r0@A(x,y), B(x), C(y) \Longleftrightarrow x > y \mid D(x,y)$$

Suppose we have the active constraint $A(1,2)\#n$. To search for a partner constraint of the form $B(1)\#m$ and $C(2)\#p$, standard CHR compilation techniques would optimize with indexing (hashtables) which allows constant time lookup for these constraints. The use of such indexing techniques is clearly applicable in a concurrent goal execution setting as long as concurrent access of the indexing data structures are handled properly. For example, we can possibly have a concurrent active constraint $A(1,3)\#q$ which will compete with $A(1,2)\#n$ for a matching partner $B(1)\#m$. As such, hashtable implementations that facilitate such indexing must be able to be accessed and modified concurrently.

Guard optimizations/simplifications aim at simplifying guard constraints by replacing guard conditions with equivalent but simplified forms. Since guards are purely declarative, they are not influenced by concurrently executing goal threads (i.e. CHR rules). Hence, all existing guard optimizations carry over to the concurrent setting.

The join order of a CHR rule determines the order in which partner constraints are searched to execute a rule. The standard CHR optimization known as *optimal join ordering* and *early guard scheduling* (Duck 2005) aims at executing goals with the most optimal order of partner constraints lookup and guard testing. By optimal, we refer to maximizing the use of constant time index lookup. Considering the same CHR rule ($r0$) above, given the active constraint $B(x)$, an optimal join ordering is to lookup for $A(x,y)$, schedule guard $x > y$, then lookup for $C(y)$. Since our

https://doi.org/10.1017/S147106841000044X Published online by Cambridge University Press

concurrent semantics does not restrict the order in which partner constraints are matched, optimal join ordering and early guard scheduling are still applicable.

Another set of optimizations tries to minimize the search for partner constraints by skipping definitely failing searches. Consider the following example:

$$r1@A \iff \cdots$$
$$r2@A, B \iff \cdots$$

If the active goal A cannot fire rule (r1) then we cannot fire rule (r2) either. Hence, after failing to fire rule (r1) we can drop goal A. Thus, we optimize away some definitely failing search. This statement is immediately true in the sequential setting where no other thread affects the constraint store. The situation is different in a concurrent setting where some other thread may have added in between the missing constraint A. Then, even after failing to fire (r1) we could fire rule (r2). However, we can argue that the optimization is still valid for this example. We will not violate the important condition to execute CHR rules exhaustively because the newly added constraint A will eventually be executed by a goal thread which then fires rule (r1). Hence, the only concern is here that the optimization leads to indeterminism in the execution order of CHR rules which is anyway unavoidable in a concurrent setting.

Yet there are existing optimizations which are not applicable in the concurrent setting. For example, *continuation optimizations* (Duck 2005; Schrijvers 2005) are not entirely applicable. Consider the following CHR rule:

$$r4@A(x), A(y) \iff x == y \mid \cdots$$

Given an active constraint $A(1)\#n$, fail continuation optimization will infer that if we fail to fire the rule with $A(1)\#n$ matching $A(x)$, there is no point trying to match it with $A(y)$ because it will most certainly fail as well, assuming that the store never changes. In a concurrent goal execution setting, we cannot assume that the store never changes (while trying to execute a CHR). For instance, after failing to trigger the rule by matching $A(1)\#n$ with $A(x)$, suppose that a new active goal $A(1)\#m$ is added to the store concurrently. Now when we match $A(1)\#n$ to $A(y)$ we can find match the partner $A(1)\#m$ with $A(x)$, hence breaking the assumptions of the fail continuation optimization.

Late (also known as delayed) storage optimization (Duck 2005) aims at delaying the storage of a goal g, until the latest point of its execution where g is possibly a partner constraint of another active constraint. Consider the following example:

$$r1@P_1 \implies Q$$
$$r2@P_2, T_1 \iff R$$
$$r3@P_3, R_1 \iff True$$
$$r4@P_4 \implies S$$
$$r5@P_5, S_1 \iff True$$

Note to distinguish the rule heads, we annotate each rule head with a subscript integer (e.g., P_x). With late storage analysis techniques described in Duck (2005), we can delay storage of an active constraint P until just before the execution of

https://doi.org/10.1017/S147106841000044X Published online by Cambridge University Press

the body of $r4$. This is because the execution of goal S (obtained from firing of $r4$) can possibly trigger $r5$. While this is safe in the sequential goal execution scheme, it is possible that rule matches are missing in the concurrent goal execution setting. Consider the case where we have some simultaneously active goals $P\#n$ and $T\#m$. Since $P\#n$ is only stored when its execution has reached $r4$, the match $r2$ can be missed entirely by both active parallel goals $P\#n$ and $T\#m$. Specifically, this happens if goal $T\#m$ is activated only after $P\#n$ has tried matching with P_2 (of $r2$), but completes goal execution (by trying T_1 of $r2$, and failing to match) before goal $P\#n$ is stored. Hence, we conclude that we cannot safely implement late storage in the concurrent setting.

5 Related work

We review prior work on execution schemes for CHR and production rule systems.

5.1 Constraint Handling Rules execution schemes

There exists a wealth of prior work on the semantics of CHR. We refer to Sneyers *et al.* (2010) for a comprehensive summary. Our focus here is on the *operational* CHR semantics and we briefly review the most relevant works.

The theoretical (a.k.a. high-level) operational semantics (Frühwirth 1998) is derived from the abstract semantics and inherits its high degree of indeterminism. The theoretical semantics has been mainly used for the study of high-level properties such as confluence (Abdennadher 1997; Abdennadher *et al*. 1999). Confluence analysis has been exploited to study the degree of concurrency in CHR programs (Frühwirth 2005; Schrijvers and Sulzmann 2008). None of these works however provide direct glues how to systematically execute concurrent programs.

In Duck *et al.* (2004) and De Koninck *et al.* (2008) some systematic, highly deterministic semantics have been developed to achieve efficient implementation schemes. However, these semantics are inherently single-threaded. Our motivation is to obtain systematic yet concurrent semantics which led us to develop the goal-based concurrent semantics presented in this paper. In the special case of a single goal thread, our semantics is equivalent to the refined operational semantics given in Duck *et al.* (2004) and De Koninck *et al.* (2008).

There are only few works which explore different semantics, other than the theoretical or abstract semantics, to address concurrency. The work of Sarna-Starosta and Ramakrishnan (2007) adopts a set-based semantics and supports tabled, possible concurrent, rule execution. This execution scheme is not applicable to CHR programs in general which usually assume a multiset based semantics. The recent work of Betz *et al.* (2009) takes a new stab at concurrency by introducing the notion of persistent constraints. The idea is to split the store into linear (multiset-like) and persistent (set-like) constraints. We are not aware of any evidence which shows that this approach supports effective concurrency in practice. Our approach leads to an efficient parallel implementation as we explain in the next section.

https://doi.org/10.1017/S147106841000044X Published online by Cambridge University Press

5.2 From concurrent to parallel Constraint Handling Rules execution

In our earlier works (Lam and Sulzmann 2007; Sulzmann and Lam 2008) we have developed a parallel CHR implementation scheme based on an informally described concurrent goal-based execution scheme (see Section 3 in Sulzmann and Lam 2008). The present works provides a concise formal treatment of the implemented concurrent goal-based execution scheme. In our implementation, multiple threads, each executing a unique CHR goal, are executed in parallel on multiple processor cores. Parallel goal executions are largely asynchronous, only implicitly synchronizing via the shared constraint store. Atomic CHR execution is guaranteed via advanced synchronization primitives such as Software Transactional Memory. We refer to Sulzmann and Lam (2008) for a thorough description of the more subtle implementation details. Our experimental results reported in Sulzmann and Lam (2008) show that we achieve good scalability when the number of processor cores increases. The overhead of the parallel implementation is fairly minor compared to a single-threaded implementations thanks to the use of lock-free algorithms. Optimization methods applicable in the concurrent/parallel setting are discussed in the earlier Section 4.3.

5.3 Parallel production rule systems

Parallel execution models of forward chaining production rule based languages (e.g., OPS5 Forgy and McDermott 1977) have been widely studied in the context of production rule systems. A production rule system is defined by a set of multiheaded production rules (analogous to CHR rules) and a set of assertions (analogous to the CHR store). Production rule systems are richer than the CHR language, consisting of user definable execution strategies and negated rule heads. This makes parallelizing production rule execution extremely difficult, because rule application is not monotonic (rules may not be applied in a larger context). As such, many previous works in parallel production rule systems focuses on efficient means of maintaining correctness of parallel rule execution (e.g., data dependency analysis Ishida 1991, sequential to parallel program transformation Gamble 1990), with respect to such user specified execution strategies. These works can be classified under two approaches, namely synchronous and asynchronous parallel production systems.

For synchronous parallel production systems (e.g., UMPOPS Gupta *et al.* 1988), multiple processors/threads run in parallel. They are synchronized by execution cycles of the production systems. Figure 8 illustrates the production cycle of a typical production rule system, consisting of three execution phases. In the (Match) phase, all rule matches are computed. Conflict resolution (CR) involves filtering out matches that do not conform to the user specified rule execution strategy, while (Act) applies the rule matches that remains (known as the eligible set) after the (CR) phase. By synchronizing parallel rule execution in production cycles, a larger class of user specified execution strategies can be supported since execution is staged.

https://doi.org/10.1017/S147106841000044X Published online by Cambridge University Press

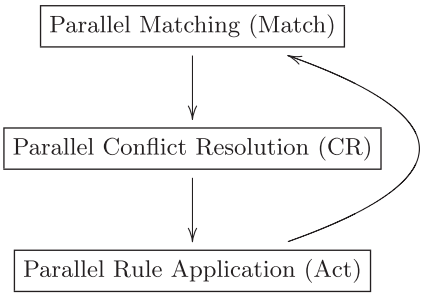
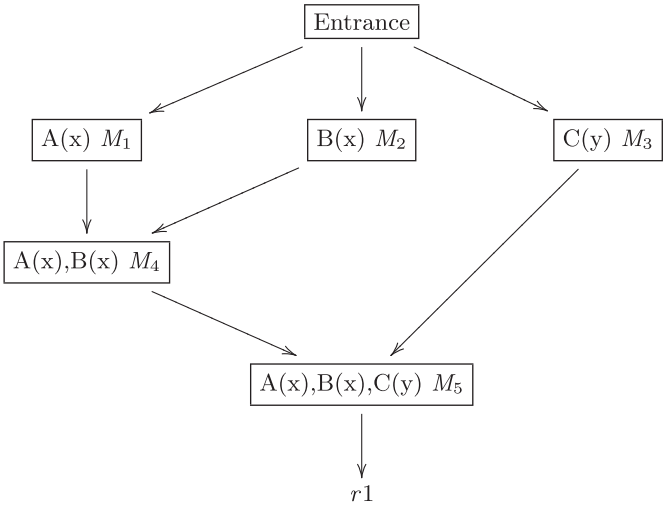


Fig. 8. Parallel production rule execution cycles.

$$r1@A(x)\backslash B(x), C(y) \Longleftrightarrow D(x,y) \qquad \{A(1), A(2), B(1), B(2), C(3)\}$$



$M_1 = \{A(1), A(2)\} \quad M_2 = \{B(1), B(2)\} \quad M_3 = \{C(3)\}$
$M_4 = \{\{A(1), B(1)\}, \{A(2), B(2)\}\}$
$M_5 = \{\{A(1), B(1), C(3)\}, \{A(2), B(2), C(3)\}\}$

Fig. 9. Example of a RETE network, in CHR context.

Matching in synchronous production rule systems often use some variant of the RETE network (Forgy 1982). RETE is an incremental matching algorithm where matching is done eagerly (data driven) in that each newly added assertion (constraint in CHR context) triggers computation of all it is possible matches to rule heads. Figure 9 illustrates a RETE network (acyclic graph), described in CHR context. Root node is the entrance where new constraints are added. Intermediate nodes with single output edges are known as alpha nodes. Intermediate nodes with two output edges are beta nodes, representing joins between alpha nodes. Each alpha node is associated with a set of constraint matching its pattern, while a beta node is associated with a set of partial/complete matches. Parallel implementation of RETE (Mahajan and Kumar 1990) allows distinct parts of the network to be computed in parallel. The most distinct characteristic of RETE is that partial

https://doi.org/10.1017/S147106841000044X Published online by Cambridge University Press

matches are computed and stored. This and the eager nature of RETE matching is suitable for production rule systems as assertions (constraints) are propagated (not deleted) by default. Hence computing all matches rarely results in redundancy. Traditional CHR systems do not advocate this eager matching scheme because doing so results to many redundancies, due to overlapping simplified matching heads. Eager matching algorithms is also proved in Miranker and Gadbois (1990) to have a larger asymptotic worst-case space complexity than lazy matching algorithms.

In Miranker (1990), the matching algorithm TREAT is proposed. TREAT is similar to RETE, except it does not store partial matches. TREAT performs better than RETE if the overhead of maintaining and storing partial matches outweighs that of recomputing partial matches.

Asynchronous parallel production rule systems (e.g., Swarm (Gamble 1990), CREL (Miranker *et al*. 1989)) introduce parallel rule execution via asynchronously running processors/threads. In such systems, rules can fire asynchronously (not synchronized by production cycles), hence enforcing execution strategies is more difficult and limited. Similar to implementations of goal based CHR semantics rule matching is such systems often use a variant of the LEAPS (Miranker *et al*. 1990) lazy matching algorithm.

5.3.1 Observations

Staging executions in synchronous parallel production rule systems allows for flexibility in imposing execution strategies, but at a cost. In Neiman (1991), synchronous execution of UMPOPS production rule system is shown to be less efficient than asynchronous execution. Hence it is clear that synchronous systems will only be necessary if we wish to impose some form of execution strategies on top of the abstract CHR semantics (e.g., rule-priority, refined operational semantics). We are interested in concurrent CHR semantics on the abstract CHR semantics. Its nondeterminism and monotonicity property provides us with the flexibility to avoid executing threads in strict staging cycles. Thus our approach is very similar to asynchronous parallel production rule systems.

Lazy matching in single-threaded CHR execution is the best choice, since we only ever have one thread of execution and wish to avoid computing overlapping (redundant) rule head matches. No doubt that in a parallel setting, eager matching (like RETE, TREAT) may be more optimal if the executed CHR program consist of rules with more propagated heads. This is because we compute more matches in parallel with brute force (find all match) parallelism and we can get away with less redundancy. Yet to cater for the general case (more simplified heads), we again choose lazy matching.

We therefore conclude that the goal-based execution model of CHR is still the ideal choice for a parallel implementation of the abstract CHR semantics. For CHR with rule priorities or refined CHR operational semantics, a variant of the synchronous parallel production rule execution is a possible choice. We leave this topic for future work.

https://doi.org/10.1017/S147106841000044X Published online by Cambridge University Press

Sequential Goal-based Semantics k-closure

$$\text{(k-Step)} \qquad \sigma \rightarrowtail^{0}_{\mathcal{G}} \sigma \qquad \frac{\sigma \rightarrowtail^{\delta}_{\mathcal{G}} \sigma' \quad \sigma' \rightarrowtail^{k}_{\mathcal{G}} \sigma''}{\sigma \rightarrowtail^{k+1}_{\mathcal{G}} \sigma''}$$

Concurrent Goal-based Semantics k-closure

$$\text{(k-Step)} \qquad \sigma \rightarrowtail^{0}_{||\mathcal{G}} \sigma \qquad \frac{\sigma \rightarrowtail^{\delta}_{||\mathcal{G}} \sigma' \quad \sigma' \rightarrowtail^{k}_{||\mathcal{G}} \sigma''}{\sigma \rightarrowtail^{k+1}_{||\mathcal{G}} \sigma''}$$

Fig. 10. k-closure derivation steps.

6 Conclusion

We have introduced a novel concurrent goal-based CHR semantics which is inspired by traditional single-threaded (sequential) goal-based CHR execution models. Existing CHR semantics aim at introducing specific execution strategies (e.g., ordered goal execution, rule priorities) on top of the CHR abstract semantics, hence adding more determinism. In contrast, the concurrent goal-based CHR semantics exploits the inherent nondeterministic and concurrent abstract CHR semantics, while introducing a systematic goal-based execution strategy. We have shown that all concurrent derivations can be replicated in the sequential goal-based semantics and that there is a correspondence between the sequential goal-based semantics and the abstract CHR semantics. Thus, establishing correctness of our concurrent goal-based CHR semantics. Our semantics provides the basis for an efficient parallel CHR implementation. The details are studied elsewhere (Sulzmann and Lam 2008).

An interesting question is how our concurrent semantics can help to parallelize an existing single-threaded semantics such as (Duck *et al*. 2004). We leave the study of this issue for future work.

Acknowledgements

We thank the reviewers for their helpful comments on a previous version of this paper.

Appendix A Proofs

In this section, we provide the proofs of the Lemmas and Theorems discussed in this paper. Because many of our proofs rely on inductive steps on the derivations, we define k-step derivations to facilitate the proof mechanisms. Figure 10 shows k-step derivations of the sequential goal-based derivations $\overset{\delta}{\rightarrowtail}_{\mathcal{G}}$ and the concurrent goal-based derivations $\overset{\delta}{\rightarrowtail}_{||\mathcal{G}}$.

https://doi.org/10.1017/S147106841000044X Published online by Cambridge University Press

A.1 Proof of correspondence of derivations

Theorem 2 (Correspondence of Sequential Derivations) For any reachable CHR state $\langle G \mid Sn\rangle$, CHR state $\langle G' \mid Sn'\rangle$ and CHR Program $\mathscr{P}$,

$$\begin{array}{ll} \text{if} & \langle G \mid Sn\rangle \rightarrowtail^{*}_{\mathscr{G}} \langle G' \mid Sn'\rangle \\ \text{then} & (NoIds(G) \uplus DropIds(Sn)) = (NoIds(G') \uplus DropIds(Sn')) \quad \vee \\ & (NoIds(G) \uplus DropIds(Sn)) \rightarrowtail^{*}_{\mathscr{A}} (NoIds(G') \uplus DropIds(Sn')) \end{array}$$

where $NoIds = \{c \mid c \in G, c \text{ is a CHR constraint}\} \uplus \{e \mid e \in G, e \text{ is an equation}\}$

Proof

We prove that for all finite n and reachable states $\langle G \mid Sn\rangle, \langle G' \mid Sn'\rangle$, $\langle G \mid Sn\rangle \rightarrowtail^{n}_{\mathscr{G}} \langle G' \mid Sn'\rangle$ either yields equivalent abstract stores or corresponds to some abstract semantics derivation. We prove by induction on the derivation steps n. Showing that goal-based derivation of any finite n steps satisfying one of the following conditions:

- **(C1)** $(NoIds(G) \uplus DropIds(Sn)) = (NoIds(G') \uplus DropIds(Sn'))$
- **(C2)** $(NoIds(G) \uplus DropIds(Sn)) \rightarrowtail^{*}_{\mathscr{A}} (NoIds(G') \uplus DropIds(Sn'))$

We have the following axioms, by definition of the functions *NoIds* and *DropIds*, for any goals G or store Sn:

- **(a1)** For any equation e, $NoIds(\{e\} \uplus G) = \{e\} \uplus NoIds(G)$
- **(a2)** For any equation e, $DropIds(\{e\} \cup Sn) = \{e\} \uplus DropIds(Sn)$
- **(a3)** For any numbered constraint $c\#i$, $NoIds(\{c\#i\} \uplus G) = NoIds(G)$
- **(a4)** For any numbered constraint $c\#i$, $DropIds(\{c\#i\} \cup Sn) = \{c\} \uplus DropIds(Sn)$
- **(a5)** For any CHR constraint c, $NoIds(\{c\} \uplus G) = \{c\} \uplus NoIds(G)$
- **(a6)** For any store Sn', $DropIds(Sn \cup Sn') = DropIds(Sn) \uplus DropIds(Sn')$

($a1$) and ($a2$) are so because *NoIds* and *DropIds* have no effect on equations. ($a3$) is true because *NoIds* is defined to drop numbered constraints. ($a4$) is true because *DropIds* is defined to remove identifier components of numbered constraints. We have ($a5$) because *NoIds* has no effect on CHR constraints. By definition of *DropIds*, ($a6$) is true.

Base case: We consider $\langle G \mid Sn\rangle \rightarrowtail^{0}_{\mathscr{G}} \langle G' \mid Sn'\rangle$. By definition of $\rightarrowtail^{0}_{\mathscr{G}}$, we have $G = G'$ and $Sn = Sn'$. Hence $(NoIds(G) \uplus DropIds(Sn)) = (NoIds(G') \uplus DropIds(Sn'))$ and we are done.

Inductive case: We assume that the theorem is true for some finite $k > 0$, hence $\langle G \mid Sn\rangle \rightarrowtail^{k}_{\mathscr{G}} \langle G' \mid Sn'\rangle$ have some correspondence with the abstract semantics.

We now prove that by extending these k derivations with another step, we preserve correspondence, namely $\langle G \mid Sn\rangle \rightarrowtail^{k}_{\mathscr{G}} \langle G' \mid Sn'\rangle \overset{\delta}{\rightarrowtail}_{\mathscr{G}} \langle G'' \mid Sn''\rangle$ has a correspondence with the abstract semantics. We prove this by considering all possible form of derivation step, step $k+1$ can take:

- (Solve) $k+1$ step is of the form $\langle \{e\} \uplus G''' \mid Sn'\rangle \overset{\delta}{\rightarrowtail}_{\mathscr{G}} \langle W \uplus G''' \mid \{e\} \cup Sn'\rangle$ such that for some G''' and W

$$G' = \{e\} \uplus G''', G'' = W \uplus G''' \text{ and } Sn'' = \{e\} \cup Sn' \quad (\mathbf{a_{solve}})$$

https://doi.org/10.1017/S147106841000044X Published online by Cambridge University Press

where e is an equation, $W = WakeUp(e, Sn)$ contains only goals of the form $c\#i$. This is because (Solve) only wakes up stored numbered constraints. Hence,

$$\begin{array}{rcll} NoIds(G'') \uplus DropIds(Sn'') & = & NoIds(W \uplus G''') \uplus DropIds(\{e\} \cup Sn') & (\mathbf{a_{solve}}) \\ & = & NoIds(G''') \uplus DropIds(\{e\} \cup Sn') & (\mathbf{a3}) \\ & = & NoIds(G''') \uplus \{e\} \uplus DropIds(Sn') & (\mathbf{a2}) \\ & = & NoIds(\{e\} \uplus G''') \uplus DropIds(Sn') & (\mathbf{a1}) \\ & = & NoIds(G') \uplus DropIds(Sn') & (\mathbf{a_{solve}}) \end{array}$$

Hence we can conclude that the evaluated store of derivation step $k+1$ is equivalent to abstract store of evaluated store of step k, therefore satisfying condition (**C1**).

- (Activate) $k+1$ step is of the form $\langle \{c\} \uplus G''' \mid Sn' \rangle \overset{\delta}{\rightarrowtail}_{\mathscr{G}} \langle \{c\#i\} \uplus G''' \mid \{c\#i\} \cup Sn' \rangle$ such that for some G'''

$$G' = \{c\} \uplus G''', G'' = \{c\#i\} \uplus G''' \text{ and } Sn'' = \{c\#i\} \cup Sn' \quad (\mathbf{a_{act}})$$

Hence,

$$\begin{array}{rcll} & & NoIds(G'') \uplus DropIds(Sn'') & \\ & = & NoIds(\{c\#i\} \uplus G''') \uplus DropIds(\{c\#i\} \cup Sn') & (\mathbf{a_{act}}) \\ & = & NoIds(G''') \uplus DropIds(\{c\#i\} \cup Sn') & (\mathbf{a3}) \\ & = & NoIds(G''') \uplus \{c\} \uplus DropIds(Sn') & (\mathbf{a4}) \\ & = & NoIds(\{c\} \uplus G''') \uplus DropIds(Sn') & (\mathbf{a5}) \\ & = & NoIds(G') \uplus DropIds(Sn') & (\mathbf{a_{act}}) \end{array}$$

Hence we can conclude that evaluated store of derivation step $k+1$ is equivalent to abstract store of evaluated store of step k, therefore satisfying condition (**C1**).

- (Simplify) $k+1$ step is of the form $\langle \{c\#i\} \uplus G''' \mid H_P \cup \{c\#i\} \cup H_S \cup Sn''' \rangle \overset{\delta}{\rightarrowtail}_{\mathscr{G}}$ $\langle B \uplus G''' \mid H_P \cup Sn''' \rangle$ for some H_P,H_S and B such that for some G''' and Sn'''

$$\begin{array}{l} Sn' = H_P \cup \{c\#i\} \cup H_S \cup Sn''', Sn'' = H_P \cup Sn''', \\ G' = \{c\#i\} \uplus G''' \text{ and } G'' = B \uplus G''' \end{array} \quad (\mathbf{a1_{simp}})$$

and there exists a CHR rule $r \mathbin{@} H'_P \backslash H'_S \iff t_g \mid B'$ such that exists ϕ where

$$\begin{array}{l} DropIds(\{c\#i\} \cup H_S) = \phi(H'_S) \quad DropIds(H_P) = \phi(H'_P) \\ Eq(Sn''') \models \phi \wedge t_g \quad B = \phi(B') \end{array} \quad (\mathbf{a2_{simp}})$$

Hence,

$$\begin{array}{ll} & NoId(G') \uplus DropIds(Sn') \\ = & NoIds(\{c\#i\} \uplus G''') \uplus DropIds(H_P \cup \{c\#i\} \cup H_S \cup Sn''') \quad (\mathbf{a1_{simp}}) \\ = & NoIds(G''') \uplus DropIds(H_P \cup \{c\#i\} \cup H_S \cup Sn''') \quad (\mathbf{a_3}) \\ = & NoIds(G''') \uplus DropIds(H_P) \uplus DropIds(\{c\#i\} \cup H_S) \uplus DropIds(Sn''') \quad (\mathbf{a_6}) \\ = & NoIds(G''') \uplus \phi(H'_P) \uplus \phi(H'_S) \uplus DropIds(Sn''') \quad (\mathbf{a2_{simp}}) \end{array}$$

By definition of the abstract semantics and $a2_{simp}$, we know that we have the rule application $\phi(H'_P) \cup \phi(H'_S) \rightarrowtail_{\mathscr{A}} \phi(B')$ Therefore, by monotonicity of CHR

https://doi.org/10.1017/S147106841000044X Published online by Cambridge University Press

rewriting (Theorem 1)

$$\begin{array}{rll} & NoId(G') \uplus DropIds(Sn') & \\ = & NoIds(G''') \uplus \phi(H'_P) \uplus \phi(H'_S) \uplus DropIds(Sn''') & \\ \rightarrowtail_{\mathscr{A}} & NoIds(G''') \uplus \phi(B') \uplus DropIds(Sn''') & \textbf{(Theorem 1)} \\ = & NoIds(\phi(B') \uplus G''') \uplus DropIds(Sn''') & \textbf{(a1)}, \textbf{(a3)} \\ = & NoIds(G'') \uplus DropIds(Sn'') & \textbf{(a1}_{\textbf{simp}}\textbf{)} \end{array}$$

Therefore, we have $NoId(G) \uplus DropIds(Sn) \rightarrowtail^*_{\mathscr{A}} NoId(G') \uplus DropIds(Sn') \rightarrowtail_{\mathscr{A}} NoIds(G'') \uplus DropIds(Sn'')$, such that the $k+1$ goal-based derivation step satisfy condition (**C2**).

- (Propagate) $k+1$ step is of the form $\langle \{c\#i\} \uplus G''' \mid H_P \cup \{c\#i\} \cup H_S \cup Sn''' \rangle \overset{\delta}{\rightarrowtail}_{\mathscr{G}} \langle B \uplus \{c\#i\} \uplus G''' \mid H_P \cup \{c\#i\} \cup Sn''' \rangle$ for some H_P,H_S and B such that for some G''' and Sn'''

$$\begin{array}{l} Sn' = H_P \cup \{c\#i\} \cup H_S \cup Sn''', Sn'' = H_P \cup \{c\#i\} \cup Sn''', \\ G' = \{c\#i\} \uplus G''' \text{ and } G'' = B \uplus \{c\#i\} \uplus G''' \qquad \textbf{(a1}_{\textbf{prop}}\textbf{)} \end{array}$$

and there exists a CHR rule $r \mathbin{@} H'_P \backslash H'_S \iff t_g \mid B'$ such that exists ϕ where

$$\begin{array}{l} DropIds(H_S) = \phi(H'_S) \quad DropIds(\{c\#i\} \cup H_P) = \phi(H'_P) \\ Eq(Sn''') \models \phi \wedge t_g \quad B = \phi(B') \qquad \textbf{(a2}_{\textbf{prop}}\textbf{)} \end{array}$$

Hence,

$$\begin{array}{ll} NoId(G') \uplus DropIds(Sn') & \\ = NoIds(\{c\#i\} \uplus G''') \uplus DropIds(H_P \cup \{c\#i\} \cup H_S \cup Sn''') & \textbf{(a1}_{\textbf{prop}}\textbf{)} \\ = NoIds(G''') \uplus DropIds(H_P \cup \{c\#i\} \cup H_S \cup Sn''') & \textbf{(a}_{\textbf{3}}\textbf{)} \\ = NoIds(G''') \uplus DropIds(\{c\#i\} \cup H_P) \uplus DropIds(H_S) \uplus DropIds(Sn''') & \textbf{(a}_{\textbf{6}}\textbf{)} \\ = NoIds(G''') \uplus \phi(H'_P) \uplus \phi(H'_S) \uplus DropIds(Sn''') & \textbf{(a2}_{\textbf{prop}}\textbf{)} \end{array}$$

By definition of the abstract semantics and $a2_{simp}$, we know that we have the rule application $\phi(H'_P) \cup \phi(H'_S) \rightarrowtail_{\mathscr{A}} \phi(B')$ Therefore, by monotonicity of CHR rewriting (Theorem 1)

$$\begin{array}{rll} & NoId(G') \uplus DropIds(Sn') & \\ = & NoIds(G''') \uplus \phi(H'_P) \uplus \phi(H'_S) \uplus DropIds(Sn''') & \\ \rightarrowtail_{\mathscr{A}} & NoIds(G''') \uplus \phi(B') \uplus DropIds(Sn''') & \textbf{(Theorem 1)} \\ = & NoIds(\phi(B') \uplus G''') \uplus DropIds(Sn''') & \textbf{(a1)}, \textbf{(a5)} \\ = & NoIds(\phi(B') \uplus \{c\#i\} \uplus G''') \uplus DropIds(Sn''') & \textbf{(a3)} \\ = & NoIds(G'') \uplus DropIds(Sn'') & \textbf{(a1}_{\textbf{prop}}\textbf{)} \end{array}$$

Therefore, we have $NoId(G) \uplus DropIds(Sn) \rightarrowtail^*_{\mathscr{A}} NoId(G') \uplus DropIds(Sn') \rightarrowtail_{\mathscr{A}} NoIds(G'') \uplus DropIds(Sn'')$, such that the $k+1$ goal-based derivation step satisfy condition (**C2**).

- (Drop) $k+1$ step is of the form $\langle \{c\#i\} \uplus G'' \mid Sn' \rangle \overset{\delta}{\rightarrowtail}_{\mathscr{G}} \langle \{G'' \mid Sn' \rangle$ such that for some G'''

$$G'' = \{c\#i\} \uplus G' \text{ and } Sn' = Sn'' \quad \textbf{(a}_{\textbf{drop}}\textbf{)}$$

https://doi.org/10.1017/S147106841000044X Published online by Cambridge University Press

Hence,

$$\begin{array}{lcll} NoIds(G'') \uplus DropIds(Sn'') & = & NoIds(\{c\#i\} \uplus G') \uplus DropIds(Sn') & (\mathbf{a_{drop}}) \\ & = & NoIds(G') \uplus DropIds(Sn') & (\mathbf{a3}) \end{array}$$

Hence we can conclude that evaluated store of derivation step $k+1$ is equivalent to abstract store of evaluated store of step k, therefore satisfying condition (**C1**).

Considering all forms of $k+1$ derivation steps, (Solve), (Activate), and (Drop) satisfies condition $bf(C1)$, while (Simplify) and (Propagate) satisfy condition (**C2**). Hence we can conclude that Theorem 2 holds. □

Lemma 1 (k-Concurrency) For any finite k of mutually nonoverlapping concurrent derivations,

$$\begin{array}{c} \langle G_1 \mid H_{S1} \cup \cdot\cdot \cup H_{Si} \cup \cdot\cdot \cup H_{Sk} \cup S\rangle \overset{H_{P1}\backslash H_{S1}}{\rightarrowtail_{||\mathscr{G}}} \langle G_1' \mid \{\} \cup \cdot\cdot \cup H_{Si} \cup \cdot\cdot \cup H_{Sk} \cup S\rangle \\ \cdot\cdot \\ \langle G_i \mid H_{S1} \cup \cdot\cdot \cup H_{Si} \cup \cdot\cdot \cup H_{Sk} \cup S\rangle \overset{H_{Pi}\backslash H_{Si}}{\rightarrowtail_{||\mathscr{G}}} \langle G_i' \mid H_{S1} \cup \cdot\cdot \cup \{\} \cup \cdot\cdot \cup H_{Sk} \cup S\rangle \\ \cdot\cdot \\ \langle G_k \mid H_{S1} \cup \cdot\cdot \cup H_{Si} \cup \cdot\cdot \cup H_{Sk} \cup S\rangle \overset{H_{Pk}\backslash H_{Sk}}{\rightarrowtail_{||\mathscr{G}}} \langle G_k' \mid H_{S1} \cup \cdot\cdot \cup H_{Si} \cup \cdot\cdot \cup \{\} \cup S\rangle \\ H_{P1} \subseteq S \cdot\cdot H_{Pi} \subseteq S \cdot\cdot H_{Pk} \subseteq S \\ \delta = H_{P1} \cup \cdot\cdot \cup H_{Pi} \cup \cdot\cdot \cup H_{Pk} \backslash H_{S1} \cup \cdot\cdot \cup H_{Si} \cup \cdot\cdot \cup H_{Sk} \end{array}$$

$$\begin{array}{ll} & \langle G_1 \uplus \cdot\cdot \uplus G_i \uplus \cdot\cdot \uplus G_k \uplus G \mid H_{S1} \cup \cdot\cdot \cup H_{Si} \cup \cdot\cdot \cup H_{Sk} \cup S\rangle \\ \overset{\delta}{\rightarrowtail_{||\mathscr{G}}} & \langle G_1' \uplus \cdot\cdot \uplus G_i' \uplus \cdot\cdot \uplus G_k' \uplus G \mid S\rangle \end{array}$$

we can decompose this into $k-1$ applications of the (pair-wise) (Goal Concurrency) derivation step.

Proof

We prove the soundness of k-concurrency by showing that k mutually nonoverlapping concurrent derivation can be decomposed into $k-1$ applications of (Goal Concurrency) step. We prove by induction on the number of concurrent derivations k.

Base case: $k = 2$. 2-concurrency immediately corresponds to (Goal Concurrency) rule, hence it is true by definition.

Inductive case: We assume that for $j > 2$ and $j < k$, we can decompose j mutually nonoverlapping concurrent derivations. into $j-1$ applications of the (Goal Concurrency) step. We now consider $j+1$ mutually nonoverlapping concurrent derivations. Because all derivations are nonoverlapping, we can compose any two derivations amongst these $j+1$ into a single concurrent step via the (Goal Concurrency) rule. We pick any two concurrent derivations, say the jth and

https://doi.org/10.1017/S147106841000044X Published online by Cambridge University Press

$(j+1)$th (note that by symmetry, this choice is arbitrary):

$$\langle G_j \mid H_{S1} \cup \cdot\cdot \cup H_{Sj} \cup H_{Sj+1} \cup S\rangle \overset{H_{Pj}\setminus H_{Sj}}{\rightarrowtail_{||\mathcal{G}}} \langle G'_j \mid H_{S1} \cup \cdot\cdot \cup \{\} \cup H_{Sj+1} \cup S\rangle$$

$$\langle G_{j+1} \mid H_{S1} \cup \cdot\cdot \cup H_{Sj} \cup H_{Sj+1} \cup S\rangle \overset{H_{Pj+1}\setminus H_{Sj+1}}{\rightarrowtail_{||\mathcal{G}}} \langle G'_{j+1} \mid H_{S1} \cup \cdot\cdot \cup H_{Sj} \cup \{\} \cup S\rangle$$

$$H_{Pj} \subseteq S \quad H_{Pj+1} \subseteq S$$

By applying the above two nonoverlapping derivations with an instance of the (Goal Concurrency) rule, we have

$$\langle G_{j'} \mid H_{S1} \cup \cdot\cdot \cup H_{Sj'} \cup S\rangle \overset{H_{Pj'}\setminus H_{Sj'}}{\rightarrowtail_{||\mathcal{G}}} \langle G'_{j'} \mid H_{S1} \cup \cdot\cdot \cup \{\} \cup S\rangle$$
$$\text{where} \quad G_{j'} = G_j \uplus G_{j+1} \quad G'_{j'} = G'_j \uplus G'_{j+1}$$
$$H_{Sj'} = H_{Sj} \cup H_{Sj+1} \quad H_{Pj'} = H_{Pj} \cup H_{Pj+1}$$

Hence we have reduced $j+1$ nonoverlapping concurrent derivations into j nonoverlapping concurrent derivations by combining via the (Goal Concurrency) derivation step.

$$\langle G_1 \mid H_{S1} \cup \cdot\cdot \cup H_{Sj'} \cup S\rangle \overset{H_{P1}\setminus H_{S1}}{\rightarrowtail_{||\mathcal{G}}} \langle G'_1 \mid \{\} \cup \cdot\cdot \cup H_{Sj'} \cup S\rangle$$
$$\cdot\cdot$$
$$\langle G_{j'} \mid H_{S1} \cup \cdot\cdot \cup H_{Sj'} \cup S\rangle \overset{H_{Pj'}\setminus H_{Sj'}}{\rightarrowtail_{||\mathcal{G}}} \langle G'_{j'} \mid H_{S1} \cup \cdot\cdot \cup \{\} \cup S\rangle$$
$$H_{P1} \subseteq S \cdot\cdot H_{Pj'} \subseteq S$$
$$\delta = H_{P1} \cup \cdot\cdot \cup H_{Pj'} \setminus H_{S1} \cup \cdot\cdot \cup H_{Sj'}$$

$$\langle G_1 \uplus \cdot\cdot \uplus G_{j'} \uplus G \mid H_{S1} \cup \cdot\cdot \cup H_{Sj'} \cup S\rangle$$
$$\overset{\delta}{\rightarrowtail_{||\mathcal{G}}} \quad \langle G'_1 \uplus \cdot\cdot \uplus G'_{j'} \uplus G \mid S\rangle$$

Hence, by our original assumption, the above is decomposable into $j-1$ applications of the (Goal Concurrency) step. This implies that $j+1$ concurrent derivations are decomposable into j (Goal Concurrency) step. □

Lemma 2 (Monotonicity of Goals in Goal-based Semantics) For any goals G,G' and G'' and CHR store Sn and Sn', if $\langle G \mid Sn\rangle \rightarrowtail^*_{\mathcal{G}} \langle G' \mid Sn'\rangle$ then $\langle G \uplus G'' \mid Sn\rangle \rightarrowtail^*_{\mathcal{G}} \langle G' \uplus G'' \mid Sn'\rangle$

Proof

We need to prove that for any finite k, if $\langle G \mid Sn\rangle \rightarrowtail^k_{\mathcal{G}} \langle G' \mid Sn'\rangle$ we can always extend the goals with any G'' such that $\langle G \uplus G'' \mid Sn\rangle \rightarrowtail^k_{\mathcal{G}} \langle G' \uplus G'' \mid Sn'\rangle$.

We prove this by induction on the number of derivation steps k, showing that for any finite $i \leqslant k$, goals are monotonic.

Base case: We consider $\langle G \mid Sn\rangle \rightarrowtail^0_{\mathcal{G}} \langle G' \mid Sn'\rangle$. By definition of $\rightarrowtail^0_{\mathcal{G}}$, we have $G = G'$ and $Sn = Sn'$. Hence we immediately have $\langle G \uplus G'' \mid Sn\rangle \rightarrowtail^0_{\mathcal{G}} \langle G' \uplus G'' \mid Sn'\rangle$

Inductive case: We assume that the lemma is true for some finite $i > 0$, hence $\langle G \mid Sn\rangle \rightarrowtail^i_{\mathcal{G}} \langle G' \mid Sn'\rangle$ is monotonic with respect to the goals.

We now prove that by extending these i derivations with another step, we still preserve monotonicity of the goals. Namely, if $\langle G \mid Sn\rangle \rightarrowtail^i_{\mathcal{G}} \langle \{g\} \uplus G_i \mid Sn_i\rangle \overset{\delta}{\rightarrowtail_{\mathcal{G}}}$

https://doi.org/10.1017/S147106841000044X Published online by Cambridge University Press

$\langle G_{i+1} \mid Sn_{i+1}\rangle$ then $\langle G \uplus G'' \mid Sn\rangle \rightarrowtail^{i}_{\mathscr{g}} \langle G_i \uplus G'' \mid Sn_i\rangle \rightarrowtail^{\delta}_{\mathscr{g}} \langle G_{i+1} \uplus G'' \mid Sn_{i+1}\rangle$ We prove this by considering all possible form of derivation step, step $i+1$th can take:

- (Solve) Consider $i+1$th derivation step of the form $\langle \{e\} \uplus G_i \mid Sn_i\rangle \rightarrowtail_{\mathscr{g}} \langle W \uplus G \mid \{e\} \cup Sn_i\rangle$ for some equation e and $W = WakeUp(e, Sn_i)$.
  By definition, the (Solve) step only make reference to e and Sn_i, hence we can extend G_i with any G'' without affecting the derivation step, i.e.

  $$\langle \{e\} \uplus G_i \uplus G'' \mid Sn_i\rangle \rightarrowtail_{\mathscr{g}} \langle W \uplus G_i \uplus G'' \mid \{e\} \cup Sn_i\rangle$$

  Hence, given our assumption that the first i derivations are monotonic with respect to the goals, extending with a $i+1$th (Solve) step preserves monotonicity of the goals.
- (Activate) Consider $i+1$th derivation step of the form $\langle \{c\} \uplus G_i \mid Sn_i\rangle \rightarrowtail_{\mathscr{g}} \langle \{c\#j\} \uplus G_i \mid \{c\#j\} \cup Sn_i\rangle$ for some CHR constraint c, goals G_i and store Sn_i.
  By definition, the (Activate) step only make reference to goal c, hence we can extend G_i with any G'' without affecting the derivation step, i.e.

  $$\langle \{c\} \uplus G_i \uplus G'' \mid Sn_i\rangle \rightarrowtail_{\mathscr{g}} \langle \{c\#j\} \uplus G_i \uplus G'' \mid \{c\#j\} \cup Sn_i\rangle$$

  Hence, given our assumption that the first i derivations are monotonic with respect to the goals, extending with a $i+1$th (Activate) step preserves monotonicity of the goals.
- (Simplify) Consider $i+1$th derivation step of the form $\langle \{c\#j\} \uplus G_i \mid \{c\#j\} \uplus H_S \cup Sn_i\rangle \rightarrowtail_{\mathscr{g}} \langle B \uplus G_i \mid Sn_i\rangle$ for some CHR constraints H_S and body constraints B.
  By definition, the (Simplify) step only make reference to goal $c\#j$, and H_S of the store, hence we can extend G_i with any G'' without affecting the derivation step, i.e.

  $$\langle \{c\#j\} \uplus G_i \uplus G'' \mid \{c\#j\} \cup H_S \cup Sn_i\rangle \rightarrowtail_{\mathscr{g}} \langle B \uplus G_i \uplus G'' \mid Sn_i\rangle$$

  Hence, given our assumption that the first i derivations are monotonic with respect to the goals, extending with a $i+1$th (Simplify) step preserves monotonicity of the goals.
- (Propagate) Consider $i+1$th derivation step of the form $\langle \{c\#j\} \uplus G_i \mid H_S \cup Sn_i\rangle \rightarrowtail_{\mathscr{g}} \langle B \uplus \{c\#j\} \uplus G_i \mid Sn_i\rangle$ for some CHR constraints H_S and body constraints B.
  By definition, the (Propagate) step only make reference to goal $c\#j$, and H_S of the store, hence we can extend G_i with any G'' without affecting the derivation step, i.e.

  $$\langle \{c\#j\} \uplus G_i \uplus G'' \mid H_S \cup Sn_i\rangle \rightarrowtail_{\mathscr{g}} \langle B \uplus \{c\#j\} \uplus G_i \uplus G'' \mid Sn_i\rangle$$

  Hence, given our assumption that the first i derivations are monotonic with respect to the goals, extending with a $i+1^{th}$ (Propagate) step preserves monotonicity of the goals.
- (Drop) Consider $i+1$th derivation step of the form $\langle \{c\#j\} \uplus G_i \mid Sn_i\rangle \rightarrowtail_{\mathscr{g}} \langle G_i \mid Sn_i\rangle$ for some numbered constraint $c\#j$.

https://doi.org/10.1017/S147106841000044X Published online by Cambridge University Press

By definition, the (Drop) step only make reference to goal $c\#j$, while its premise depend on Sn_i, hence we can extend goals G_i with any G'' without affecting the derivation step, i.e.

$$\langle \{c\#j\} \uplus G_i \uplus G'' \mid Sn_i \rangle \rightarrowtail_{\mathscr{g}} \langle G_i \uplus G'' \mid Sn_i \rangle$$

Hence, given our assumption that the first i derivations are monotonic with respect to the goals, extending with a $i{+}1$th (Drop) step preserves monotonicity of the goals.

Hence, with our assumption of monotonicity of goals for i steps, the goals are still monotonic for $i+1$ steps regardless of the form of the $i+1$th derivation step. □

Lemma 3 (Isolation of Goal-based Derivations) If $\langle G \mid H_P \cup H_S \cup S_1 \cup S_2 \rangle \overset{H_P \setminus H_S}{\rightarrowtail_{\mathscr{g}}} \langle G' \mid H_P \cup S_1' \cup S_2 \rangle$ then $\langle G \mid H_P \cup H_S \cup S_1 \rangle \overset{H_P \setminus H_S}{\rightarrowtail_{\mathscr{g}}} \langle G' \mid H_P \cup S_1' \rangle$

Proof

We need to show that for any goal-based derivation, we can omit any constraint of the store which is not a side-effect of the derivation. To prove this, we consider all possible forms of goal-based derivations:

- (Solve) Consider derivation of the form

$$\langle \{e\} \uplus G \mid W \cup \{\} \cup S_1 \cup S_2 \rangle \overset{W \setminus \{\}}{\rightarrowtail_{\mathscr{g}}} \langle W \uplus G \mid W \cup \{\} \cup \{e\} \cup S_1 \cup S_2 \rangle$$

  Since wake up side-effect is captured in W, we can drop S_2 without affecting the derivation. Hence we also have:

$$\langle \{e\} \uplus G \mid W \cup \{\} \cup S_1 \rangle \overset{W \setminus \{\}}{\rightarrowtail_{\mathscr{g}}} \langle W \uplus G \mid W \cup \{\} \cup \{e\} \cup S_1 \rangle$$

- (Activate) Consider derivation of the form

$$\langle \{c\} \uplus G \mid \{\} \cup \{\} \cup S_1 \cup S_2 \rangle \overset{\{\} \setminus \{\}}{\rightarrowtail_{\mathscr{g}}} \langle \{c\#i\} \uplus G \mid \{\} \cup \{\} \cup \{c\#i\} \cup S_1 \cup S_2 \rangle$$

  Since (Activate) simply introduces a new constraint $c\#i$ into the store, we can drop S_2 without affecting the derivation. Hence we also have:

$$\langle \{c\} \uplus G \mid \{\} \cup \{\} \cup S_1 \rangle \overset{\{\} \setminus \{\}}{\rightarrowtail_{\mathscr{g}}} \langle \{c\#i\} \uplus G \mid \{\} \cup \{\} \cup \{c\#i\} \cup S_1 \rangle$$

- (Simplify) Consider derivation of the form

$$\langle \{c\#i\} \uplus G \mid H_P \cup H_S \cup S_1 \cup S_2 \rangle \overset{H_P \setminus H_S}{\rightarrowtail_{\mathscr{g}}} \langle B \uplus G \mid H_P \cup S_1 \cup S_2 \rangle$$

  Since S_2 is not part of the side-effects of this derivation, we can drop S_2 without affecting the derivation. Hence we also have:

$$\langle \{c\#i\} \uplus G \mid H_P \cup H_S \cup S_1 \rangle \overset{H_P \setminus H_S}{\rightarrowtail_{\mathscr{g}}} \langle B \uplus G \mid H_P \cup S_1 \rangle$$

- (Propagate) Consider derivation of the form

$$\langle \{c\#i\} \uplus G \mid H_P \cup H_S \cup S_1 \cup S_2 \rangle \overset{H_P \setminus H_S}{\rightarrowtail_{\mathscr{g}}} \langle B \uplus \{c\#i\} \uplus G \mid H_P \cup S_1 \cup S_2 \rangle$$

https://doi.org/10.1017/S147106841000044X Published online by Cambridge University Press

Since S_2 is not part of the side-effects of this derivation, we can drop S_2 without affecting the derivation. Hence we also have

$$\langle \{c\#i\} \uplus G \mid H_P \cup H_S \cup S_1 \rangle \overset{H_P \backslash H_S}{\mapsto_{\mathscr{G}}} \langle B \uplus \{c\#i\} \uplus G \mid H_P \cup S_1 \rangle$$

- (Drop) Consider derivation of the form

$$\langle \{c\#i\} \uplus G \mid \{\} \cup \{\} \cup S_1 \cup S_2 \rangle \overset{\{\} \backslash \{\}}{\mapsto_{\mathscr{G}}} \langle G \mid \{\} \cup \{\} \cup S_1 \cup S_2 \rangle$$

(Drop) simply removes the goal $c\#i$ when no instances of (Simplify) or (Propagate) can apply on it. Note that its premise references to the entire store, so removing S_2 may seems unsafe. But since removing constraints from the store will not cause $c\#i$ to be applicable to any instances of (Simplify) or (Propagate), hence we also have:

$$\langle \{c\} \uplus G \mid \{\} \cup \{\} \cup S_1 \rangle \overset{\{\} \backslash \{\}}{\mapsto_{\mathscr{G}}} \langle G \mid \{\} \cup \{\} \cup S_1 \rangle$$

□

Lemma 4 (Isolation of Transitive Goal-based Derivations) If $\langle G \mid H_P \cup H_S \cup S_1 \cup S_2 \rangle \mapsto^*_{\mathscr{G}} \langle G' \mid H_P \cup S'_1 \cup S_2 \rangle$ with side-effects $\delta = H_P \backslash H_S$, then $\langle G \mid H_P \cup H_S \cup S_1 \rangle \mapsto^*_{\mathscr{G}} \langle G' \mid H_P \cup S'_1 \rangle$

Proof

We need to prove that for all k, $\langle G \mid H_P \cup H_S \cup S_1 \cup S_2 \rangle \overset{k}{\mapsto_{\mathscr{G}}} \langle G' \mid H_P \cup S'_1 \cup S_2 \rangle$ with side-effects $\delta = H_P \backslash H_S$ we can always safely omit affected potions of the store from the derivation. We prove by induction on $i \leqslant k$.

Base case: $i = 1$. Consider, $\langle G \mid H_P \cup H_S \cup S_1 \cup S_2 \rangle \mapsto^1_{\mathscr{G}} \langle G' \mid H_P \cup S'_1 \cup S_2 \rangle$. This corresponds to the premise in Lemma 3, hence we can safely omit S_2 from the derivation.

Inductive case: $i > 1$. we assume that for any $\langle G \mid H_{Pi} \cup H_{Si} \cup S_{1i} \cup S_{2i} \rangle \mapsto^i_{\mathscr{G}} \langle G' \mid H_{Pi} \cup S'_{1i} \cup S_{2i} \rangle$ with side-effects $\delta_i = H_{Pi} \backslash H_{Si}$, we can safely omit S_{2i} from the derivation. Let us consider a $j = i + 1$ derivation step from here, which contains side-effects $\delta_j = H_{Pj} \backslash H_{Sj}$ nonoverlapping with δ_i. Hence H_{Pj} and H_{Sj} must be in S_{2i} (i.e. $S_{2i} = H_{Pj} \cup H_{Sj} \cup S_{1j} \cup S_{2j}$).

$$\begin{array}{ll} & \langle G \mid H_{Pi} \cup H_{Si} \cup S_{1i} \cup H_{Pj} \cup H_{Sj} \cup S_{1j} \cup S_{2j} \rangle \\ \mapsto^i_{\mathscr{G}} & \langle G' \mid H_P \cup S'_{1i} \cup H_{Pj} \cup H_{Sj} \cup S_{1j} \cup S_{2j} \rangle \\ \overset{\delta_j}{\mapsto_{\mathscr{G}}} & \langle G'' \mid H_P \cup S'_{1i} \cup H_{Pj} \cup S'_{1j} \cup S_{2j} \rangle \end{array}$$

Hence consider the following substitutions:

$$\begin{array}{ll} H_P = H_{Pi} \cup H_{Pj} & H_S = H_{Si} \cup H_{Sj} \\ S_1 = S_{1i} \cup S_{1j} & S'_1 = S'_{1i} \cup S'_{1j} \\ \delta = H_P \backslash H_S & \end{array}$$

we have $\langle G \mid H_P \cup H_S \cup S_1 \cup S_{2j} \rangle \mapsto^{i+1}_{\mathscr{G}} \langle G \mid H_P \cup S'_1 \cup S_{2j} \rangle$ with side-effects δ such that no constraints in S_{2j} is in δ. Hence we can safely omit S_{2j} from the derivation and we have isolation for $i + 1$ derivations as well. □

https://doi.org/10.1017/S147106841000044X Published online by Cambridge University Press

Lemma 5 (Sequential Reachability of Concurrent Derivation Steps) For any sequentially reachable CHR state σ, CHR state σ' and rewriting side-effects δ if $\sigma \overset{\delta}{\rightarrowtail}_{||\mathscr{g}} \sigma'$ then σ' is sequentially reachable, $\sigma \rightarrowtail^*_{\mathscr{g}} \sigma'$ with side-effects δ.

Proof

From the k-concurrency Lemma (Lemma 1) we showed that any finite k mutually nonoverlapping concurrent goal-based derivations can be replicated by nested application of the (Goal Concurrency) step. Hence, to prove sequential reachability of concurrent derivations, we only need to consider the derivation steps (Lift) and (Goal Concurrency) which sufficiently covers the concurrent behaviour of any k concurrent derivations.

We prove by structural induction of the concurrent goal-based semantics derivation steps (Lift) and (Goal Concurrency).

- (Lift) is the base case. Application of (Lift) simply lifts a goal-based derivation $\sigma \overset{\delta}{\rightarrowtail}_{\mathscr{g}} \sigma'$ into a concurrent goal-based derivation $\sigma \overset{\delta}{\rightarrowtail}_{||\mathscr{g}} \sigma'$. Thus states σ' derived from the (Lift) step is immediately sequentially reachable since $\sigma \overset{\delta}{\rightarrowtail}_{\mathscr{g}} \sigma'$ implies $\sigma \rightarrowtail^*_{\mathscr{g}} \sigma'$.
- (Goal Concurrency)

$$\begin{array}{c}
\textbf{(D1)} \quad \langle G_1 \mid H_{S1} \cup H_{S2} \cup S\rangle \overset{\delta_1}{\rightarrowtail}_{||\mathscr{g}} \langle G'_1 \mid \{\} \cup H_{S2} \cup S\rangle \\
\textbf{(D2)} \quad \langle G_2 \mid H_{S1} \cup H_{S2} \cup S\rangle \overset{\delta_2}{\rightarrowtail}_{||\mathscr{g}} \langle G'_2 \mid H_{S1} \cup \{\} \cup S\rangle \\
\delta_1 = H_{P1} \backslash H_{S1} \quad \delta_2 = H_{P2} \backslash H_{S2} \\
H_{P1} \subseteq S \quad H_{P2} \subseteq S \quad \delta = H_{P1} \cup H_{P2} \backslash H_{S1} \cup H_{S2} \\
\hline
\\
\textbf{(C)} \quad \overset{\delta}{\rightarrowtail}_{||\mathscr{g}} \quad \begin{array}{l} \langle G_1 \uplus G_2 \uplus G \mid H_{S1} \cup H_{S2} \cup S\rangle \\ \langle G'_1 \uplus G'_2 \uplus G \mid S\rangle \end{array}
\end{array}$$

we assume that **(D1)** and **(D2)** are sequentially reachable. This means that we have the following:

$$\begin{array}{l}
\langle G_1 \mid H_{S1} \cup H_{S2} \cup S\rangle \rightarrowtail^*_{\mathscr{g}} \langle G'_1 \mid \{\} \cup H_{S2} \cup S\rangle \\
\quad \text{with side-effects } \delta_1 = H_{P1} \backslash H_{S1} \text{ such that } H_{P1} \subseteq S \quad (\mathbf{a_{D1}})
\end{array}$$

$$\begin{array}{l}
\langle G_2 \mid H_{S1} \cup H_{S2} \cup S\rangle \rightarrowtail^*_{\mathscr{g}} \langle G'_2 \mid H_{S1} \cup \{\} \cup S\rangle \\
\quad \text{with side-effects } \delta_2 = H_{P2} \backslash H_{S2} \text{ such that } H_{P2} \subseteq S \quad (\mathbf{a_{D2}})
\end{array}$$

Since both derivations are by definition nonoverlapping in side-effects, we can show that **(C)** is sequentially reachable, using monotonicity of goals (Lemma 2) and isolation of derivations (Lemma 3):

$$\begin{array}{lll}
 & \langle G_1 \uplus G_2 \uplus G \mid H_{S1} \cup H_{S2} \cup S\rangle & \\
\rightarrowtail^*_{\mathscr{g}} & \langle G'_1 \uplus G_2 \uplus G \mid H_{S2} \cup S\rangle & (\mathbf{Lemma2}, \mathbf{a_{D1}}) \\
\rightarrowtail^*_{\mathscr{g}} & \langle G'_1 \uplus G'_2 \uplus G \mid S\rangle & (\mathbf{Lemma2}, \mathbf{Lemma4}, \mathbf{a_{D2}})
\end{array}$$

Hence, the above sequential goal-based derivation shows that (Goal Concurrency) derivation step is sequentially reachable with side-effect δ.

□

https://doi.org/10.1017/S147106841000044X Published online by Cambridge University Press

Theorem 3 (Sequential Reachability of Concurrent Derivations) For any initial CHR state σ, CHR state σ' and CHR Program $\mathscr{P}$, if $\sigma \rightarrowtail^*_{||\mathscr{G}} \sigma'$ then $\sigma \rightarrowtail^*_{\mathscr{G}} \sigma'$.

Proof
We prove that for all finite k number of concurrent derivation steps $\sigma \rightarrowtail^k_{||\mathscr{G}} \sigma'$, we can find a corresponding sequential derivation sequence $\sigma \rightarrowtail^*_{\mathscr{G}} \sigma'$.

Base case: $k = 1$. We consider $\sigma \rightarrowtail^1_{||\mathscr{G}} \sigma'$. From Lemma 5, we can conclude that we have $\sigma \rightarrowtail^*_{\mathscr{G}} \sigma'$ as well.

Inductive case: $k > 1$. We consider $\sigma \rightarrowtail^k_{||\mathscr{G}} \sigma'$, assuming that it is sequentially reachable, hence we also have $\sigma \rightarrowtail^*_{\mathscr{G}} \sigma'$. We consider extending this derivation with the $k+1$th step $\sigma' \rightarrowtail_{||\mathscr{G}} \sigma''$. By Lemma 5, we can conclude that the $k+1$th concurrent derivation is sequential reachable, hence $\sigma' \rightarrowtail^*_{\mathscr{G}} \sigma''$. Hence we have $\sigma \rightarrowtail^*_{\mathscr{G}} \sigma' \rightarrowtail^*_{\mathscr{G}} \sigma''$ showing that $\sigma \rightarrowtail^{k+1}_{||\mathscr{G}} \sigma''$ is sequentially reachable. □

A.2 Correspondence of termination

Lemma 6 (Rule instances in reachable states are always active) For any reachable CHR state $\langle G \mid Sn\rangle$, any rule head instance $H \subseteq Sn$ must be active. i.e. $\exists c\#i \in H$ such that $c\#i \in G$.

Proof
We will prove this for the sequential goal-based semantics. Since Theorem 3 states all concurrent derivation is sequentially reachable, this Lemma immediately applies to the concurrent goal-based semantics as well.

We prove that for all finite k derivations from any initial CHR state $\langle G \mid \{\}\rangle$, i.e. $\langle G \mid \{\}\rangle \rightarrowtail^k_{\mathscr{G}} \langle G' \mid Sn'\rangle$, all rule head instances $H \subseteq Sn'$ has at least one $c\#i \in H$ such that $c\#i \in G$. We prove by induction on $i < k$ that states reachable by i derivations from an initial stage have the above property.

Base case: $i = 0$. Hence $\langle G \mid \{\}\rangle \rightarrowtail^0_{\mathscr{G}} \langle G' \mid Sn'\rangle$. By definition, $G = G'$ and $Sn' = \{\}$. Since Sn' is empty, the base case immediately satisfies the Lemma.

Inductive case: $i > 0$. We assume that for any $\langle G \mid \{\}\rangle \rightarrowtail^i_{\mathscr{G}} \langle G' \mid Sn'\rangle$, all rule head instances $H \subseteq Sn'$ is active, hence have at least one $c\#i \in H$ such that $c\#i \in G'$. We extend this derivation with an $i{+}1^{th}$ step, hence $\langle G \mid \{\}\rangle \rightarrowtail^i_{\mathscr{G}} \langle G' \mid Sn'\rangle \overset{\delta}{\rightarrowtail}_{\mathscr{G}} \langle G'' \mid Sn''\rangle$. We now prove that all rule head instances in Sn'' are active. We consider all possible forms of this $i+1$th derivation step. We omit side-effects.

- (Solve) $i+1$ derivation step is of the form $\langle \{e\} \uplus G''' \mid Sn'\rangle \rightarrowtail_{\mathscr{G}} \langle W \uplus G''' \mid \{e\} \cup Sn'\rangle$ for some goals G''' and $W = WakeUp(e, Sn')$. Our assumption provides that all rule head instances in Sn' are active. Introducing e into the store will possibly introduce new rule head instances. This is because for some CHR rule $(r \mathbin{@} H_P \backslash H_S \iff t_g \mid B) \in \mathscr{P}$ since we may have a new ϕ such that $Eqs(\{e\} \cup Sn') \models \phi \wedge t_g$ and $\phi(H_P \cup H_S) \in Sn'$. This means that there is at least one $c\#i$ in $\phi(H_P \cup H_S)$ which is further grounded by e. Thankfully, by definition of $W = WakeUp(e, Sn')$, we have $c\#i \in W$. Hence new rule head instances will become active because of introduction of W to the goals.
- (Activate) $i+1$ derivation step is of the form $\langle \{c\} \uplus G''' \mid Sn'\rangle \rightarrowtail_{\mathscr{G}} \langle \{c\#i\} \uplus G''' \mid \{c\#i\} \cup Sn'\rangle$. Our assumption provides that all rule head instances in Sn' are

https://doi.org/10.1017/S147106841000044X Published online by Cambridge University Press

active. By adding $c\#i$ to the store, we can possibly introduce new rule head instances $\{c\#i\} \cup H$ such that $H \in Sn'$. Since $c\#i$ is also retained as a goal, such new rule head instances are active as well.

- (Simplify) $i+1$ derivation step is of the form $\langle \{c\#i\} \uplus G''' \mid \{c\#i\} \cup H_S \cup Sn' \rangle \rightarrowtail_{\mathscr{G}} \langle B \uplus G''' \mid Sn' \rangle$. Our assumption provides that all rule head instances in Sn' are active. $c\#i$ has applied a rule instance, removing $c\#i$ and some H_S from the store. Since $c\#i$ is no longer in the store, we can safely remove $c\#i$ from the goals. Removing H_S from the store will only (possibly) remove other rule head instance from the store. Hence rule head instances in Sn' still remain active.
- (Propagate) $i+1$ derivation step is of the form $\langle \{c\#i\} \uplus G''' \mid \{c\#i\} \cup H_S \cup Sn' \rangle \rightarrowtail_{\mathscr{G}} \langle B \uplus \{c\#i\} \uplus G''' \mid \{c\#i\} \cup Sn' \rangle$. Our assumption provides that all rule head instances in Sn' are active. $c\#i$ has applied a rule instance, removing some H_S from the store. Since $c\#i$ is still in the store, we cannot safely remove $c\#i$ from the goals, thus it is retained. Removing H_S from the store will only (possibly) remove other rule head instance from the store. Hence rule head instances in Sn', including those that contains $c\#i$, still remain active.
- (Drop) $i+1$ derivation step is of the form $\langle \{c\#i\} \uplus G''' \mid Sn' \rangle \rightarrowtail_{\mathscr{G}} \langle G''' \mid Sn' \rangle$. Our assumption provides that all rule head instances in Sn' are active. Premise of the (Drop) step demands that no (Simplify) and (Propagate) steps apply on $c\#i$. This means that $c\#i$ is not part of any rule head instances in Sn'. Hence we can safely remove $c\#i$ from the goals without risking to deactivate any rule instances.

Hence (Solve) and (Activate) guarantees that new rule head instances become active, (Drop) safely removes a goal without deactivating any rule head instances and (Simplify) and (Propagate) only removes constraint from the store. In all cases, existing rule head instances remain active while new rule head instances become active, thus we have proved the lemma. □

Theorem 4 (Correspondence of Termination) For any initial CHR state $\langle G, \{\} \rangle$, final CHR state $\langle \{\}, Sn \rangle$ and terminating CHR program $\mathscr{P}$,

$$\begin{aligned} &\text{if } \langle G \mid \{\} \rangle \rightarrowtail^*_{\|\mathscr{G}} \langle \{\} \mid Sn \rangle \\ &\text{then } G \rightarrowtail^*_{\mathscr{A}} \mathit{DropIds}(Sn) \text{ and } \mathit{Final}_{\mathscr{A}}(\mathit{DropIds}(Sn)) \end{aligned}$$

Proof

We prove that for any concurrent derivation $\langle G \mid \{\} \rangle \rightarrowtail^*_{\|\mathscr{G}} \langle \{\} \mid Sn \rangle$, we have a corresponding abstract derivation $G \rightarrowtail^*_{\mathscr{A}} \mathit{DropIds}(Sn)$. Theorem 3 states that we can replicate the above concurrent derivation, with a sequential derivation. Hence we have $\langle G \mid \{\} \rangle \rightarrowtail^*_{\mathscr{G}} \langle \{\} \mid Sn \rangle$. By instantiating Theorem 2, we immediately have $G \rightarrowtail^*_{\mathscr{A}} \mathit{DropIds}(Sn)$ from this sequential goal-based derivation.

Next we show that $\mathit{DropIds}(Sn)$ is a final store ($\mathit{Final}_{\mathscr{A}}(\mathit{DropIds}(Sn))$) with respect to some CHR program $\mathscr{P}$. We prove by contradiction: Suppose $\mathit{DropIds}(Sn)$ is not a final store, hence $\langle \{\} \mid Sn \rangle$ has at least one rule head instance H of $\mathscr{P}$ in Sn which is not active, since the goals are empty. However, this contradicts with Lemma 6, which states that all reachable states have only active rule head instances. Since $\langle \{\} \mid Sn \rangle$

https://doi.org/10.1017/S147106841000044X Published online by Cambridge University Press

is sequentially reachable, it must be the case that Sn has no rule head instances of $\mathscr{P}$. Therefore $DropIds(Sn)$ must be a final store. □

References

Abdennadher, S. 1997. Operational semantics and confluence of constraint propagation rules. In *Proceedings of CP'97*, G. Smolka, Ed. Lecture Notes in Computer Science, vol. 1330. Springer-Verlag, 252–266.

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

Betz, H., Raiser, F. and Frühwirth, T. 2009. Persistent constraints in Constraint Handling Rules. In *WLP '09: Proceedings of the 23rd Workshop on (Constraint) Logic Programming* (in press).

De Koninck, L., Stuckey, P. and Duck, G. 2008. Optimizing compilation of CHR with rule priorities. In *Proceedings of FLOPS'08*. Lecture Notes in Computer Science, vol. 4989. Springer-Verlag, 32–47.

Miranker, D. P., Brant, D., Lofaso, B. and Gadbois, D. 1990. On the performance of lazy matching in production systems. In *In Proceedings of International Conference on Artificial Intelligence AAAI*. 685–692.

Duck, G. J. 2005. *Compilation of Constraint Handling Rules*. PhD thesis, The University of Melbourne.

Duck, G. J., Stuckey, P. J., de la Banda, M. J. G. and Holzbaur, C. 2004. The refined operational semantics of Constraint Handling Rules. In *Proceedings of ICLP'04*. Lecture Notes in Computer Science, vol. 3132. Springer-Verlag, 90–104.

Forgy, C. 1982. Rete: A fast algorithm for the many patterns/many objects match problem. *Artificial Intelligence 19,* 1, 17–37.

Forgy, C. and McDermott, J. P. 1977. Ops, a domain-independent production system language. In *IJCAI*. 933–939.

Frühwirth, T. 1998. Theory and practice of constraint handling rules. *Journal of Logic Programming, Special Issue on Constraint Logic Programming 37,* 1–3, 95–138.

Frühwirth, T. 2005. Parallelizing union-find in Constraint Handling Rules using confluence analysis. In *Proceedings of ICLP'05*. Lecture Notes in Computer Science, vol. 3668. Springer-Verlag, 113–127.

Frühwirth, T. 2006. Constraint handling rules: the story so far. In *Proceedings of PPDP '06*. ACM Press, 13–14.

Gamble, R. F. 1990. Transforming rule-based programs: from the sequential to the parallel. In *IEA/AIE '90: Proceedings of the 3rd international conference on Industrial and engineering applications of artificial intelligence and expert systems*. ACM, New York, NY, 854–863.

Gupta, A., Forgy, C., Kalp, D., Newell, A., and Tambe, M. 1988. Parallel ops5 on the encore multimax. In *ICPP (1)*. 71–280.

Ishida, T. 1991. Parallel rule firing in production systems. *IEEE Transactions on Knowledge and Data Engineering 3,* 1, 11–17.

Lam, E. S. L. and Sulzmann, M. 2007. A concurrent Constraint Handling Rules implementation in Haskell with software transactional memory. In *Proc. of ACM SIGPLAN Workshop on Declarative Aspects of Multicore Programming (DAMP'07)*. 19–24.

Mahajan, M. and Kumar, V. K. P. 1990. Efficient parallel implementation of rete pattern matching. *Computer Systems: Science & Engineering 5,* 3, 187–192.

Miranker, D. P. 1990. *TREAT: A New and Efficient Match Algorithm for AI Production Systems*. Morgan Kaufmann Publishers Inc., San Francisco, CA.

https://doi.org/10.1017/S147106841000044X Published online by Cambridge University Press

Miranker, D. P., Kuo, C. and Browne, J. C. 1989. *Parallelizing Transformations for a Concurrent Rule Execution Language*. Tech. Rep.: CS-TR-89-30, University of Texas, Austin, TX.

Neiman, D. E. 1991. Control issues in parallel rule-firing production systems. In *Proceedings of National Conference on Artificial Intelligence*. AAAI Press, 310–316.

Sarna-Starosta, B. and Ramakrishnan, C. R. 2007. Compiling constraint handling rules for efficient tabled evaluation. In *Proc. of PADL'07*. Lecture Notes in Computer Science, vol. 4354. Springer, 170–184.

Schrijvers, T. 2005. Analyses, optimizations and extensions of Constraint Handling Rules: PhD summary. In *Proc. of ICLP'05*. Lecture Notes in Computer Science, vol. 3668. Springer-Verlag, 435–436.

Schrijvers, T. and Sulzmann, M. 2008. Transactions in constraint handling rules. In *Proc. of ICLP'08*. Lecture Notes in Computer Science, vol. 5366. Springer, 516–530.

Sneyers, J., Schrijvers, T. and Demoen, B. 2005. Guard and continuation optimization for occurrence representations of CHR. In *Proceedings of ICLP'05*. Lecture Notes in Computer Science, vol. 3668. Springer-Verlag, 83–97.

Sneyers, J., Van Weert, P., Schrijvers, T. and De Koninck, L. 2010. As time goes by: Constraint handling rules - a survey of CHR research from 1998 to 2007. *Theory and Practice of Logic Programming 10* (1), 1–47.

Sulzmann, M. and Lam, E. S. L. 2008. Parallel execution of multi-set constraint rewrite rules. In *Proc. of PPDP'08*. ACM Press, 20–31.

https://doi.org/10.1017/S147106841000044X Published online by Cambridge University Press