Establishing a Correspondence between Transactional Events and Constraint Handling Rules

Martin Sulzmann Kai Stadtmüller Edmund Lam
Karlsruhe University of Applied Sciences,Karlsruhe, Germany CMU Qatar
martin.sulzmann@hs-karlsruhe.de kai.stadtmueller@live.de sllam@qatar.cmu.edu

Transactional events and Constraint Handling Rules (CHR) are high-level concurrency abstractions. Both concepts have so far been studied independently. We establish a formal correspondence between the two by showing that the semantics of transactional events can be explained in terms of CHR. This result opens up the opportunity for a fruitful exchange of ideas. For example, applying CHR optimization and analysis methods to the setting of transactional events.

1 Introduction

Message-based concurrency is an attractive programming model which generally avoids the common pitfalls of low-level concurrency abstractions based on shared memory and locks. Languages such as Concurrent ML [14] provide high-level event abstractions based on a 2-way rendezvous, synchronous message exchange among two parties and non-deterministic choice via which we can select among multiple events. Thus, programmers can efficiently build applications which are scalable on modern multi-core architectures [13] The work in [2] extends this programming model by including a form of software transaction. Unlike Software Transactional Memory (STM) which enforces strict isolation [18], transactional events support the cooperation among transactions while still following the 'all-or-nothing' semantics of STM.

Our interest is to establish a formal correspondence between transactional events [2] and Constraint Handling Rules (CHR) [7]. CHR were originally invented for the rule-based formulation of high-level constraint solvers. In recent years, CHR saw a multitude of further applications ranging from temporal reasoning, testing/verification, type system specifications etc. We refer to [19] for an overview. We argue that CHR are also a good match to specify the semantics of high-level concurrency features such as transactional events.

CHR is a naturally concurrent, rule-based formalism with a simple semantics and efficient compilation scheme. Briefly, CHR operate on a (multi-set) constraint store where each CHR rule specifies the atomic rewriting of some constraints by some other constraints. In our setting, constraints describe synchronous message exchange via channels. By expressing transactional events via CHR, we can take advantage of the many works on CHR optimization and analysis, e.g. see [3, 17, 4, 15]. This hopefully enables a fruitful exchange of ideas among two concepts which so far have been studied independently.

In this work, we take a first step in establishing a formal correspondence between CHR and transactional events. Specifically, our contributions are:

- We introduce a trace semantics for transactional events via which we can precisely track 2-way rendezvous synchronization steps (Section 3).
- The trace semantics provides the formal basis to establish a correspondence between transactional events and CHR. Fully synchronized transactional event reductions can be expressed in terms of CHR rewriting steps (Section 4).

Submitted to:
EXPRESS/SOS 2015

© M. Sulzmann, K. Stadtmüller & E. Lam
This work is licensed under the
Creative Commons Attribution License.

- For special cases such as 2-way rendezvous and non-deterministic choice, as for example found in Concurrent ML, we can give a complete description of the semantics in terms of a generic set of CHR rules (Section 4.3).

The up-coming section reviews the basics of transactional events [2]. Section 5 discusses related work and concludes.

2 Transactional Events

The syntax of transactional events we will work with is as follows.

Definition 1 (Syntax)

$$\begin{array}{rcll} E & ::= & !x \mid ?x & \textit{Send/Receive} \\ & \mid & E+E & \textit{Alternatives} \\ & \mid & E;E & \textit{Sequence} \end{array}$$

where we use letters $x, y, z, \ldots$ to denote channel names.

The key ingredients are send/receive primitives which follow the 2-way rendezvous principle found in CSP [8]. Via $+$ we can non-deterministically select among events. Furthermore, we support the execution of events in sequence as proposed in [2] which follow the all-or-nothing semantics of transactions. For brevity, we ignore values transmitted over channels. We also omit combinators to process (un)successful events as found in CML [14].

In examples, we assume that ; binds tighter than $+$. Hence, to avoid clutter we write $!x;!y+?x$ as a short-hand for $(!x;!y)+?x$.

The meaning of transactional events is explained in terms of a small-step rewrite semantics in style of [12]. The small-step rewrite semantics operates on a set of concurrent (transactional events). Formally, we make use of configurations to represent this set.

Definition 2 (Configurations) *We say $X_1 \| \ldots \| X_n$ is a* configuration *where each component X_i is of the form*

$$\begin{array}{rcl} E' & ::= & \bullet \mid E \\ X & ::= & \mathsf{sync}\, E \mid E' \end{array}$$

We say that $\mathsf{sync}\, E_1 \| \ldots \| \mathsf{sync}\, E_n$ is an initial *configuration. We say that $\bullet \| \ldots \| \bullet \| \mathsf{sync}\, E_i \| \ldots \| \mathsf{sync}\, E_n$ is a* final *configuration. To all other configurations we refer to as* intermediate *configurations.*

The idea is that $\bullet$ represents an event that could be successfully executed (synchronized). The form $\mathsf{sync}\, E$ denotes a pending event. That is, an event which has not been executed yet. The form E denotes an event in execution where some parts have already been executed. We write E' to denote intermediate forms which represent either a successfully executed or a partially executed event.

For example, $\mathsf{sync}\ !x+?z \| \mathsf{sync}\ !y+?x$ denotes an initial configuration which consists of two concurrent events. The first event can be executed if we find matching partners for either $!x$ or $?z$. The second event seeks for matching partners for either $!y$ or $?x$. It is easy to see that both events can be synchronized via channel x.

Next, we define the semantic rules which rewrite an initial configuration into some final configuration (if possible at all).

Definition 3 (Semantics)

$$\textit{(Rendezvous)} \quad !x||?x \longrightarrow \bullet||\bullet \quad !x;E_1||?x;E_2 \longrightarrow E_1||E_2 \quad !x||?x;E_2 \longrightarrow \bullet||E_2 \quad !x;E_1||?x \longrightarrow E_1||\bullet$$

$$\textit{(Alt)} \quad E_1+E_2||E_3 \longrightarrow E_1||E_3 \quad E_1+E_2||E_3 \longrightarrow E_2||E_3$$

$$(E_1+E_2);E_3 \longrightarrow E_1;E_3 \quad (E_1+E_2);E_3 \longrightarrow E_2;E_3$$

$$\textit{(Seq)} \quad (E_1;E_2);E_3 \longrightarrow E_1;(E_2;E_3)$$

$$\textit{(Sync)} \quad \mathsf{sync}\ E_1||X_2||...||X_n \longrightarrow E_1||X_2||...||X_n$$

$$\textit{(Context)} \quad \frac{E_1||E_2 \longrightarrow E_1'||E_2'}{E_1||E_2||X_3||...||X_n \longrightarrow E_1'||E_2'||X_3||...||X_n}$$

$$\textit{(Reorder)} \quad X_1||...||X_n \longrightarrow X_{\pi(1)}||...||X_{\pi(n)}$$

where π is a permutation on $\{1,....,n\}$.

The first rule set (Rendezvous) takes care of the 2-way rendezvous principle and therefore only involves configurations consisting of two components. The first rule consider the case that we find a matching pair of sender and receiver. Via the (Reorder) we can swap components. Hence, it suffices to assume that the sender is in the first and the receiver in the second component. The remaining rules are necessary to cover cases where sender/receiver appear in a sequence.

Rules (Alt) deal with non-deterministic choice. Rule (Seq) ensures that sequences of events are in right-associative form. Thus, we can guarantee that send/receive primitives appear in leading position and therefore the four (Rendezvous) rules are sufficient. Via rule (Sync) we start the execution of some event. With rule (Context) we apply the rules involving two components in a larger context. Recall that 'primed' events represent either a successfully executed or a partially executed event. See Definition 2. Thanks to the reordering rule (Reorder) it suffices to consider only the leading components in rule (Context).

Definition 4 (Synchronization) *We say that a set of concurrent (transactional) events $E_1,...,E_n$ can be* synchronized *if we find a derivation*

$$\mathsf{sync}\ E_1||...||\mathsf{sync}\ E_n \longrightarrow^* \bullet||...||\bullet||\mathsf{sync}\ E_i||...||\mathsf{sync}\ E_n$$

where $\longrightarrow^*$ denotes the reflexive, transitive closure of the reduction relation $\longrightarrow$.

We consider a few examples. We find that $\mathsf{sync}\ !x+?y||\mathsf{sync}\ !y+?x$ can be synchronized because

$$\mathsf{sync}\ !x+?y||\mathsf{sync}\ !y+?x \longrightarrow^* \bullet||\bullet$$

The individual derivation steps are

$$\begin{array}{ll}
 & \mathsf{sync}\ !x+?y||\mathsf{sync}\ !y+?x \\
\rightarrowtail_{\mathit{Sync}} & !x+?y||\mathsf{sync}\ !y+?x \\
\rightarrowtail_{\mathit{Reorder}} & \mathsf{sync}\ !y+?x||!x+?y \\
\rightarrowtail_{\mathit{Sync}} & !y+?x||!x+?y \\
\rightarrowtail_{\mathit{Alt}} & ?x||!x+?y \\
\rightarrowtail_{\mathit{Reorder}} & !x+?y||?x \\
\rightarrowtail_{\mathit{Alt}} & !x||?x \\
\rightarrowtail_{\mathit{Rendezvous}} & \bullet||\bullet
\end{array}$$

where for clarity we annotate each derivation step with the rule (set) involved.

It's clear that the semantics is non-deterministic as we find another derivation

$$\begin{array}{ll} & \mathsf{sync}\ !x{+}?y \| \mathsf{sync}\ !y{+}?x \\ \rightarrowtail^{*} & ?y \| !y \\ \rightarrowtail_{\mathit{Rendezvous}} & \bullet \| \bullet \end{array}$$

where we synchronize via channel y instead of channel x.

Next, we consider some examples involving sequences of events. For brevity, we omit the (Reorder) step and assume that (Rendezvous) steps can be applied to any two components (not only the first two) in a configuration. We find that

$$\begin{array}{ll} & \mathsf{sync}\ !x;!y \| \mathsf{sync}\ ?x \| \mathsf{sync}\ ?y \\ \rightarrowtail^{*}_{\mathit{Sync}} & !x;!y \| ?x \| ?y \\ \rightarrowtail_{\mathit{Rendezvous}} & !y \| \bullet \| ?y \\ \rightarrowtail_{\mathit{Rendezvous}} & \bullet \| \bullet \| \bullet \end{array}$$

On the other hand, for $\mathsf{sync}\ !x;!y \| \mathsf{sync}\ ?y;?x$ no final configuration exists because the leading events in the sequence do not match up.

As our last example, we consider

$$\begin{array}{ll} & \mathsf{sync}\ !x;!y \| \mathsf{sync}\ ?x;?z \| \mathsf{sync}\ !z;?y \\ \longrightarrow^{*}_{\mathit{Sync}} & !x;!y \| ?x;?z \| !z;?y \\ \longrightarrow_{\mathit{Rendezvous}} & !y \| ?z \| !z;?y \\ \longrightarrow_{\mathit{Rendezvous}} & !y \| \bullet \| ?y \\ \longrightarrow_{\mathit{Rendezvous}} & \bullet \| \bullet \| \bullet \end{array}$$

Our next goal is to establish a refined semantics via which we can record a *trace* of the individual rendezvous steps. Thanks to the information recorded in the trace, we can replay the semantic behavior of transactional events in terms of CHR rewriting. The up-coming section presents the details of the trace-based semantics. Section 4 covers the connection to CHR rewriting.

3 Trace Semantics

For convenience, we assume that events are normalized by assuming that an event consists of alternatives of sequences of primitives where each sequence is in right-associative form.

Each event can be transformed into such a normal form by applying the following equivalences.

Definition 5 (Equivalences) *We consider two events equivalent if they can be transformed into each other via the following identities.*

$$(E_1;E_2);E_3 \equiv E_1;(E_2;E_3)$$

$$E_1 + E_2 \equiv E_2 + E_1 \quad (E_1+E_2);E_3 \equiv E_1;E_3 + E_2;E_3$$

The semantics of the thus transformed events remains unchanged as stated by the following result. Besides symbols E, we also use symbols F to denote events.

Proposition 1 (Semantic Preservation for Normalized Events) *Let $E_1,...,E_n,F_1,...,F_n$ be events such that $E_i \equiv F_i$ for $i = 1,...,n$. Then,*

$$\mathsf{sync}\ E_1 \| ... \| \mathsf{sync}\ E_n \longrightarrow^* \bullet \| ... \| \bullet \| \mathsf{sync}\ E_{\pi(i)} \| ... \| \mathsf{sync}\ E_{\pi(n)}$$

iff

$$\mathsf{sync}\ F_1 \| ... \| \mathsf{sync}\ F_n \longrightarrow^* \bullet \| ... \| \bullet \| \mathsf{sync}\ F_{\pi(i)} \| ... \| \mathsf{sync}\ F_{\pi(n)}$$

where π is a permutation on $1,...,n$.

Proof Sketch. By induction over the derivation. □

To precisely keep track of the rendezvous steps of normalized events we assume that events are annotated with branch and position information.

Definition 6 (Annotated Normalized Events)

$$\begin{array}{lcl} P & ::= & !x_{(b,p)} \mid ?x_{(b,p)} \\ S & ::= & P \mid P;S \\ A & ::= & S \mid A + A \end{array}$$

where $b, p \in \mathbb{N}^+$ and b represents the branch and p the position.

We say that a normalized event A is valid *if each branch has a distinct branch number and the positions in each sequence are in increasing order. We assume that branch numbers and position start with 1. This condition is expressed formally in terms of relations $\vdash^b_{alt} A$, $\vdash^{(b,p)}_{seq} S$ and $\vdash^{(b,p)}_{prim} P$. For a top-level expression A, we verify that $\vdash^1_{alt} A$ holds. The rules are as follows:*

$$\text{(Alt-Valid)} \quad \frac{\vdash^b_{alt} A_1 \quad \vdash^{b+1}_{alt} A_2}{\vdash^b_{alt} A_1 + A_2} \qquad \frac{\vdash^{(b,1)}_{seq} S}{\vdash^b_{alt} S}$$

$$\text{(Seq-Valid)} \quad \frac{\vdash^{(b,p)}_{prim} P}{\vdash^{(b,p)}_{seq} P} \qquad \frac{\vdash^{(b,p)}_{prim} P \quad \vdash^{(b,p+1)}_{seq} S}{\vdash^{(b,p)}_{seq} P;S}$$

$$\text{(Prim-Valid)} \quad \frac{b = b' \ \ p = p'}{\vdash^{(b',p')}_{prim} !x_{(b,p)}} \qquad \frac{b = b' \ \ p = p'}{\vdash^{(b',p')}_{prim} ?x_{(b,p)}}$$

For example, $!x_{(1,1)}; !y_{(1,2)} + ?x_{(2,1)}$ is valid. On the other hand, $!x_{(1,1)}; !y_{(2,2)} + ?x_{(2,1)}$ is invalid because of the ill-formed branch in $!y_{(2,2)}$.

Definition 7 (Annotated Configurations)

$$\begin{array}{lcl} A' & ::= & \bullet \mid A \\ Y & ::= & \mathsf{sync}\ A \mid A' \end{array}$$

We attach a thread number to each component in a configuration. We say a configuration $t_1 : Y_1 \| ... \| t_n : Y_n$ is valid *iff thread numbers t_i are distinct and components Y_i are valid.*

Let us recall the earlier (but now annotated) valid initial configuration

$$1 : \mathsf{sync}\ !x_{(1,1)} + ?y_{(2,1)} \| 2 : \mathsf{sync}\ !y_{(1,1)} + ?x_{(2,1)}$$

Next, we introduce the details of the trace semantics whose purpose is to record rendezvous steps.

Definition 8 (Trace Semantics)

$$\begin{array}{lcll} L & ::= & (t,b,p) & \textit{Label} \\ K & ::= & !?x(t_1,b_1,p_1) \leftrightarrow (t_2,b_2,p_2) & \textit{Labeled Rendezvous Pair} \\ R & ::= & \emptyset \mid K \mid R,R & \textit{Trace} \end{array}$$

For traces the following identities apply:

$$\emptyset,R = R \quad R,\emptyset = R \quad R_1,\emptyset,R_2 = R_1,R_2$$

$$\textit{(Rendezvous)} \quad \frac{R = !?x(t_1,b_1,p_1) \leftrightarrow (t_2,b_2,p_2)}{t_1 : !x_{(b_1,p_1)} \| t_2 : ?x_{(b_2,p_2)} \xrightarrow{R} \bullet \| \bullet} \quad \frac{R = !?x(t_1,b_1,p_1) \leftrightarrow (t_2,b_2,p_2)}{t_1 : !x_{(b_1,p_1)};S_1 \| t_2 : ?x_{(b_2,p_2)};S_2 \xrightarrow{R} S_1 \| S_2}$$

$$\frac{R = !?x(t_1,b_1,p_1) \leftrightarrow (t_2,b_2,p_2)}{t_1 : !x_{(b_1,p_1)} \| t_2 : ?x_{(b_2,p_2)};S_2 \xrightarrow{R} \bullet \| S_2} \quad \frac{R = !?x(t_1,b_1,p_1) \leftrightarrow (t_2,b_2,p_2)}{t_1 : !x_{(b_1,p_1)};S_1 \| t_2 : ?x_{(b_2,p_2)} \xrightarrow{R} S_1 \| \bullet}$$

$$\textit{(Alt)} \quad A_1 + A_2 \xrightarrow{\emptyset} A_1 \quad A_1 + A_2 \xrightarrow{\emptyset} A_2$$

$$\textit{(Sync)} \quad t_1 : \mathsf{sync}\, A_1 \| t_2 : Y_2 \| ... \| t_n : Y_n \xrightarrow{\emptyset} t_1 : A_1 \| t_2 : Y_2 \| ... \| t_n : Y_n$$

$$\textit{(Context)} \quad \frac{t_1 : A_1 \| t_2 : A_2 \xrightarrow{R} t_1 : A_1' \| t_2 : A_2'}{t_1 : A_1 \| t_2 : A_2 \| t_3 : Y_3 \| ... \| t_n : Y_n \xrightarrow{R} t_1 : A_1' \| t_2 : A_2' \| t_3 : Y_3 \| ... \| t_n : Y_n}$$

$$\textit{(Reorder)} \quad t_1 : X_1 \| ... \| t_n : X_n \xrightarrow{\emptyset} t_{\pi(1)} : X_{\pi(1)} \| ... \| t_{\pi(n)} : X_{\pi(n)}$$

$$\textit{(Trans)} \quad \frac{t_1 : X_1 \| ... \| t_n : X_n \xrightarrow{R_1} t_1' : X_1' \| ... \| t_n' : X_n' \quad t_1' : X_1' \| ... \| t_n' : X_n' \xrightarrow{R_2} t_1'' : X_1'' \| ... \| t_n'' : X_n''}{t_1 : X_1 \| ... \| t_n : X_n \xrightarrow{R_1,R_2} t_1'' : X_1'' \| ... \| t_n'' : X_n''}$$

where π is a permutation on $\{1,....,n\}$.

A trace records the two endpoints of a rendezvous step. The semantic rules are largely the ones from Definition 3 but customized to the case of normalized events. Derivation steps include now traces. The symbol $\emptyset$ denotes the neutral element among traces and can be removed by applying the above identities. For example, we find that

$$1 : \mathsf{sync}\ !x_{(1,1)} + ?y_{(2,1)} \| 2 : \mathsf{sync}\ !y_{(1,1)} + ?x_{(2,1)} \xrightarrow{R}{}^{*} \bullet \| \bullet$$

where either $R = !?x(1,1,1) \leftrightarrow (2,2,1)$ or $R = !?y(1,2,1) \leftrightarrow (2,1,1)$. As observed earlier, there are two possible choices for synchronization. Either via channel x or channel y and each trace precisely captures this information.

Ignoring traces the refined (trace) semantics is equivalent to the earlier semantics (for normalized events).

Proposition 2 (Semantic Preservation for Trace Semantics) *Let* $t_1 : \mathsf{sync}\, A_1 \| ... \| t_n : \mathsf{sync}\, A_n$ *be a valid initial configuration. Then,*

$$\mathsf{sync}\, A_1 \| ... \| \mathsf{sync}\, A_n \longrightarrow^* \bullet \| ... \| \bullet \| \mathsf{sync}\, A_{\pi(i)} \| ... \| \mathsf{sync}\, A_{\pi(n)}$$

iff

$$t_1 : \mathsf{sync}\, A_1 \| ... \| t_n : \mathsf{sync}\, A_n \overset{R}{\longrightarrow^*} t_{\pi(1)} : \bullet \| ... \| t_{\pi(i-1)} : \bullet \| t_{\pi(i)} : \mathsf{sync}\, A_{\pi(i)} \| ... \| t_{\pi(n)} : \mathsf{sync}\, A_{\pi(n)}$$

for some R where π *is a permutation on* $\{1, ..., n\}$.

Proof Sketch. By induction over the derivation. □

The recorded traces precisely capture among which threads and among which branches and at which position a rendezvous step takes place. We characterize this information in terms of the following condition.

Definition 9 (Valid Trace Rendezvous Condition) *We say that R is* valid *iff the following conditions hold:*

1. *For each* $!?x(t_1, b_1, p_1) \leftrightarrow (t_2, b_2, p_2)$ *in R we have that* $t_1 \neq t_2$.
2. *For any two* $!?x(t_1, b_1, p_1) \leftrightarrow (t_2, b_2, p_2), !?y(t_3, b_3, p_3) \leftrightarrow (t_4, b_4, p_4)$ *in R we have that*

$$\begin{array}{l}(t_1 = t_3 \implies b_1 = b_3) \wedge (t_1 = t_4 \implies b_1 = b_4) \wedge \\ (t_2 = t_3 \implies b_2 = b_3) \wedge (t_2 = t_4 \implies b_2 = b_3)\end{array}$$

3. *For any two* $!?x(t_1, b_1, p_1) \leftrightarrow (t_2, b_2, p_2), !?y(t_3, b_3, p_3) \leftrightarrow (t_4, b_4, p_4)$ *in R where*

$$R = ..., !?x(t_1, b_1, p_1) \leftrightarrow (t_2, b_2, p_2), ..., !?y(t_3, b_3, p_3) \leftrightarrow (t_4, b_4, p_4), ...$$

we have that

$$\begin{array}{l}(t_1 = t_3 \implies p_1 < p_3) \wedge (t_1 = t_4 \implies p_1 < p_4) \wedge \\ (t_2 = t_3 \implies p_2 < p_3) \wedge (t_2 = t_4 \implies p_2 < p_4)\end{array}$$

The first condition ensures that a rendezvous always takes place among distinct threads. The second condition guarantees that all rendezvous steps for a specific thread are taken from the same branch The last (third) condition ensures that rendezvous steps for a specific thread are sorted according to their textual (position) order.

Proposition 3 (Final Configurations yield Valid Traces) *Let* $t_1 : \mathsf{sync}\, A_1 \| ... \| t_n : \mathsf{sync}\, A_n$ *be a valid initial configuration such that*

$$t_1 : \mathsf{sync}\, A_1 \| ... \| t_n : \mathsf{sync}\, A_n \overset{R}{\longrightarrow^*} t_{\pi(1)} : \bullet \| ... \| t_{\pi(i-1)} : \bullet \| t_{\pi(i)} : \mathsf{sync}\, A_{\pi(i)} \| ... \| t_{\pi(n)} : \mathsf{sync}\, A_{\pi(n)}$$

for some R where π *is a permutation on* $\{1, ..., n\}$. *Then, R is valid.*

Proof Sketch. First condition in Definition 9 follows immediately because thread numbers are distinct.

Let us consider the second condition. The semantic rules always pick a certain sequence for synchronization. See rules (Alt) in Definition 8. Furthermore, branch information must be valid for annotated events. See Definition 6. Hence, it is a simple combinatorial observation, considering the combinations of trace-based semantics rules (Rendezvous) and (Reorder), that for $!?x(t_1,b_1,p_1) \leftrightarrow (t_2,b_2,p_2), !?y(t_3,b_3,p_3) \leftrightarrow (t_4,b_4,p_4)$ in R we must have that

$$\begin{array}{l}(t_1 = t_3 \implies b_1 = b_3) \wedge (t_1 = t_4 \implies b_1 = b_4) \wedge \\ (t_2 = t_3 \implies b_2 = b_3) \wedge (t_2 = t_4 \implies b_2 = b_3)\end{array}$$

A similar observation applies to the third condition. □

The trace semantics as well as the valid trace rendezvous condition are our technical instruments to verify that the semantic behavior of transactional events can be captured in terms of CHR rewriting. This is what we will discuss next.

4 Correspondence to CHR Rewriting

We first introduce the basics of CHR rewriting [7]. Then, we give a rather elementary CHR encoding of transactional events. We show that for any initial configuration, all possible final configurations can be obtained via CHR rewriting. Lastly, we show that the encoding scheme can be significantly improved for some classes of transactional events.

4.1 CHR Basics

Constraints are first-order predicates. We distinguish among two kinds of constraints. CHR *user* constraints and *built-in* constraints. Built-in constraints are dealt with by a predefined constraint solver. The details are kept abstract and therefore we assume that validity of built-in constraints is checked via some entailment relation $\models$. The meaning of CHR constraints is defined in terms of CHR rules. For our purposes, it is sufficient to consider simplification rules.

Definition 10 (CHR Simplification Rules) *A* simplification *CHR is of the form*

$$r@H_1,...,H_n \Longleftrightarrow G \mid B_1,...,B_n$$

where we assume that H_i, B_j are CHR constraints and G is a built-in constraint. If the sequence $B_1,...,B_n$ is empty, we write True*.*

We commonly record CHR rules in a set P.

CHR operate on a constraint store. Commonly, the CHR store is treated like a multi-set semantics [7]. In our setting, all our CHR constraints are distinct. Hence, we treat the store as a *set*.

Definition 11 (CHR Rewriting) *Let S be a set of CHR constraints also referred to as a constraint* store*. Let $r@H_1,...,H_n \Longleftrightarrow G \mid B_1,...,B_n$ be a CHR simplification. Then, S can be rewritten into S' by application of r, written $S \rightarrowtail_r S'$ if the following holds:*

- *There exists $H'_1,...,H'_n \in S$ and a substitution ϕ such that $\phi(H_i) = H'_i$ for $i = 1,...,n$.*
- *$\models \phi(G)$.*
- *$S' = (S - \{H'_1,...,H'_n\}) \cup \{\phi(B_1),...,\phi(B_n)\}$.*

We write $\rightarrowtail^*$ to denote the application of several CHR rewriting steps.

In our setting, all CHR rules are terminating. That is, we always reach a final store S on which no further rules are applicable.

Next, we consider how to express transactional events in terms of CHR.

4.2 Elementary CHR Encoding of Transactional Events

We start by giving a mapping from events to constraints. For our purposes, we make use of CHR constraints of the form $!x(t,b,p)$, $?x(t,b,p)$, $\mathsf{sync}(t)$ and $\mathsf{last}(t,b,p)$. Constraints $!x(t,b,p)$, $?x(t,b,p)$ represent the respective end points of a rendezvous step. Constraint $\mathsf{sync}(t)$ indicates that a thread attempts to synchronize. Constraint $\mathsf{last}(t,b,p)$ represents the 'last' position in a sequence and ensures that all events in a sequence are synchronized.

Definition 12 (From Events to Constraints)

$$[\![t : !x_{(b,p)}]\!]_E = \begin{cases} \{!x(t,b,p), \mathsf{last}(t,b,p)\} & \textit{if } p \textit{ last position in } (t,b) \\ \{!x(t,b,p)\} & \textit{otherwise} \end{cases}$$

$$[\![t : ?x_{(b,p)}]\!]_E = \begin{cases} \{?x(t,b,p), \mathsf{last}(t,b,p)\} & \textit{if } p \textit{ last position in } (t,b) \\ \{?x(t,b,p)\} & \textit{otherwise} \end{cases}$$

$$[\![t : P; S]\!]_E = [\![t : P]\!]_E \cup [\![t : S]\!]_E \quad [\![t : A_1 + A_2]\!]_E = [\![t : A_1]\!]_E \cup [\![t : A_2]\!]_E$$

$$[\![t_1 : \mathsf{sync}\, A_1 \| \ldots \| t_n : \mathsf{sync}\, A_N]\!]_E = \{\mathsf{sync}(t_1)\} \cup [\![t_1 : A_1]\!]_E \cup \ldots \cup \{\mathsf{sync}(t_n)\} \cup [\![t_n : A_n]\!]_E$$

On the thus translated initial configuration, we apply some appropriate CHR rules which mimic the semantic rules of transactional events. The insight is that we can derive these rules from traces.

We first introduce a mapping $[\![\cdot]\!]_H$ for turning each trace into a rule head. Each rendezvous synchronization point is decomposed into the respective end points of the rendezvous step. For each thread we add a sync constraint and also record the last position of each sequence. As we might add several sync constraints for the same thread, we wish to remove such duplicates from rule heads. Hence, we use a set to accumulate constraints. Thus, duplicates are removed. This set is then flattened into a rule head. Hence, we find the auxiliary mappings $[\![\cdot]\!]_{H'}$ and $|\cdot|$.

Definition 13 (From Traces to Rule Heads)

$$\begin{array}{rcl} [\![!?x(t_1,b_1,p_1) \leftrightarrow (t_2,b_2,p_2)]\!]_{H'} & = & \{\mathsf{sync}(t_1,), \mathsf{sync}(t_2), !x(t_1,b_1,p_1), ?x(t_2,b_2,p_2)\} \\ & & \cup\{\mathsf{last}(t_1,b_1,p_1) \mid \textit{if } p_1 \textit{ last position in } (t_1,b_1)\} \\ & & \cup\{\mathsf{last}(t_2,b_2,p_2) \mid \textit{if } p_2 \textit{ last position in } (t_2,b_2)\} \end{array}$$

$$[\![R,R]\!]_{H'} = [\![R]\!]_{H'} \cup [\![R]\!]_{H'}$$

$$|\{H_1,\ldots,H_n\}| = H_1,\ldots.,H_n \ \ \textit{where } H_i \neq H_j \textit{ for } i \neq j$$

$$[\![R]\!]_H = |[\![R]\!]_{H'}|$$

Next, we introduce mapping $[\![R]\!]_{CHR}$ which translates each trace R into a CHR rule (R).

Definition 14 (From Traces to CHR)

$$[\![R]\!]_{CHR} = R@|[\![R]\!]_{H'}| \Longleftrightarrow R \textit{ is valid} \mid |\{\mathsf{cleanup}(t) \mid \mathsf{sync}(t) \in [\![R]\!]_{H'}\}|$$

$$\begin{array}{rcl} c1@\mathsf{cleanup}(t), !x(t',b,p) & \Longleftrightarrow & t' = t \mid \mathsf{cleanup}(t) \\ c2@\mathsf{cleanup}(t), ?x(t',b,p) & \Longleftrightarrow & t' = t \mid \mathsf{cleanup}(t) \\ c3@\mathsf{cleanup}(t), \mathsf{last}(t',b,p) & \Longleftrightarrow & t' = t \mid \mathsf{cleanup}(t) \\ c4@\mathsf{cleanup}(t) & \Longleftrightarrow & \mathsf{True} \end{array}$$

As we will see, to correctly capture the semantics of transactional events, we need to guarantee that the trace satisfies the validity conditions in Definition 9. Hence, the guard condition in the CHR rule R. Furthermore, we need some 'clean-up' rules to remove constraints associated to events which did not take part in a synchronization step.

In our formulation of the cleanup rules $c1-c4$, we play a well-known CHR trick. We assume a goal-based execution scheme [3, 10] where goals attempt to execute rules in textual order. This guarantees that rule $c4$ will only be tried if none of the rules $c1-3$ are applicable. Hence, this trick ensures that the cleanup is carried out exhaustively via rules $c1-3$ and the cleanup constraint is finally removed via $c4$. There would be no harm to drop rule $c4$ on the expense that some 'garbage' remains in the store. Thanks to rules $c1-4$ such 'garbage' can be removed which also makes the formulation of our results more straightforward.

Here is a simple example to illustrate the CHR encoding of transactional events. Let us recall the earlier example

$$1 : \mathsf{sync}\ !x_{(1,1)} + ?y_{(2,1)} \| 2 : \mathsf{sync}\ !y_{(1,1)} + ?x_{(2,1)} \xrightarrow{R} \bullet \| \bullet$$

where $R_1 = !?x(1,1,1) \leftrightarrow (2,2,1)$.

The above mapping yields

$$\begin{array}{c} [\![1 : \mathsf{sync}\ !x_{(1,1)} + ?y_{(2,1)} \| 2 : \mathsf{sync}\ !y_{(1,1)} + ?x_{(2,1)}]\!]_E \\ = \\ \left\{ \begin{array}{l} \mathsf{sync}(1), !x(1,1,1), \mathsf{last}(1,1,1), ?y(1,2,1), \mathsf{last}(1,2,1), \\ \mathsf{sync}(2), !y(2,1,1), \mathsf{last}(2,1,1), ?x(2,2,1), \mathsf{last}(2,2,1) \end{array} \right\} \end{array}$$

$$[\![R_1]\!]_{CHR} = R1@ \begin{array}{l} \mathsf{sync}(1), !x(1,1,1), \mathsf{last}(1,1,1), \\ \mathsf{sync}(2), ?x(2,2,1), \mathsf{last}(2,2,1) \end{array} \iff \mathsf{cleanup}(1), \mathsf{cleanup}(2)$$

Let P be the set of CHR consisting of rule R and the cleanup rules $c1-4$. Then, we find

$$\begin{array}{ll} & \left\{ \begin{array}{l} \underline{\mathsf{sync}(1)}, \underline{!x(1,1,1)}, \underline{\mathsf{last}(1,1,1)}, ?y(1,2,1), \mathsf{last}(1,2,1), \\ \underline{\mathsf{sync}(2)}, !y(2,1,1), \mathsf{last}(2,1,1), \underline{?x(2,2,1)}, \underline{\mathsf{last}(2,2,1)} \end{array} \right\} \\[2ex] \rightarrowtail_{R1} & \left\{ \begin{array}{l} \underline{\mathsf{cleanup}(1)}, \underline{?y(1,2,1)}, \mathsf{last}(1,2,1), \\ \mathsf{cleanup}(2), !y(2,1,1), \mathsf{last}(2,1,1) \end{array} \right\} \\[2ex] \rightarrowtail_{c1} & \left\{ \begin{array}{l} \underline{\mathsf{cleanup}(1)}, \underline{\mathsf{last}(1,2,1)}, \\ \mathsf{cleanup}(2), !y(2,1,1), \mathsf{last}(2,1,1) \end{array} \right\} \\[2ex] \rightarrowtail_{c3} & \{\underline{\mathsf{cleanup}(1)}, \mathsf{cleanup}(2), !y(2,1,1), \mathsf{last}(2,1,1)\} \\[2ex] \rightarrowtail_{c4} & \{\mathsf{cleanup}(2), !y(2,1,1), \mathsf{last}(2,1,1)\} \\[2ex] \rightarrowtail^{*} & \{\} \end{array}$$

Constraints involved in a rule application are underlined. For brevity, we abbreviate clean-up of thread 2 which could be executed concurrently with clean-up of thread 1.

More formally, we can state that CHR resulting from valid traces capture the semantics of transactional events.

Proposition 4 (Transactional Events via CHR - Soundness) *Let $t_1 : \mathsf{sync}\, A_1 || ... || t_n : \mathsf{sync}\, A_n$ be a valid initial configuration such that*

$$t_1 : \mathsf{sync}\, A_1 || ... || t_n : \mathsf{sync}\, A_n \overset{R}{\longrightarrow}{}^{*}\ t_{\pi(1)} : \bullet || ... || t_{\pi(i-1)} : \bullet || t_{\pi(i)} : \mathsf{sync}\, A_{\pi(i)} || ... || t_{\pi(n)} : \mathsf{sync}\, A_{\pi(n)}$$

for some R where π is a permutation on $\{1, ..., n\}$. Let P consist of the CHR rule R resulting from $[\![R]\!]_{CHR}$ and the clean-up rules $c1-4$. Then, $[\![t_1 : \mathsf{sync}\, A_1 || ... || t_n : \mathsf{sync}\, A_n]\!]_E \rightarrowtail_P^* S$ for some S where $S = [\![t_{\pi(i)} : \mathsf{sync}\, A_{\pi(i)} || ... || t_{\pi(n)} : \mathsf{sync}\, A_{\pi(n)}]\!]_E$.

Proof Sketch. The insight is that in the CHR encoding we perform a 'big' step to reach a final configuration whereas the transactional semantics performs 'small' steps. The intermediate configurations we reach in the transactional small-step semantics effectively correspond to CHR matching steps where we seek to find matching constraints for rules heads.

Hence, a rigorous proofs requires a further refined semantic description of CHR, e.g. in style of the goal-based semantics in [3]. Thus, we can carry out an inductive proof over the trace-based transactional events derivation which is then connected the the refined CHR semantic rules. For brevity, we omit the mostly tedious technical details. □

Furthermore, we can show that for a fixed initial configuration we find a finite set of CHR which captures all possible final configurations. We simply build brute-force an approximation of all valid traces and then apply the above result.

We find a finite approximation of all valid traces as follows. The maximal length m of any valid trace is bound by the maximal length of a sequence. Each component of a trace is of the form $!?x(t_1, b_1, p_1) \leftrightarrow (t_2, b_2, p_2)$ where channels x, threads t_1, t_2, branches b_1, b_2 and p_1, p_2 are constrained by the initial configuration. Hence, we simply consider all possible combinations of $!?x(t_1, b_1, p_1) \leftrightarrow (t_2, b_2, p_2)$ for traces up to length m. We denote this set by $\mathscr{R}$.

We then build the set of CHR $[\![\mathscr{R}]\!]_{CHR} = \{[\![R]\!]_{CHR} \mid R \in \mathscr{R}\}$. R may not be a valid trace. Hence, the guard condition in Definition 14 becomes important.

By application of Proposition 4 we obtain the following result.

Proposition 5 (Transactional Events via CHR - Soundness II) *Let $t_1 : \mathsf{sync}\ A_1 || ... || t_n : \mathsf{sync}\ A_n$ be a valid initial configuration such that*

$$t_1 : \mathsf{sync}\, A_1 || ... || t_n : \mathsf{sync}\, A_n \overset{R}{\longrightarrow}{}^{*}\ t_{\pi(1)} : \bullet || ... || t_{\pi(i-1)} : \bullet || t_{\pi(i)} : \mathsf{sync}\, A_{\pi(i)} || ... || t_{\pi(n)} : \mathsf{sync}\, A_{\pi(n)}$$

for some R where π is a permutation on $\{1, ..., n\}$. Let $\mathscr{R}$ denote the approximation of the set of all valid traces resulting from $t_1 : \mathsf{sync}\, A_1 || ... || t_n : \mathsf{sync}\, A_n$. Let P consist of the set of CHR rules $[\![\mathscr{R}]\!]_{CHR}$ and the clean-up rules $c1-4$. Then, there exists a constraint S such that

$$[\![t_1 : \mathsf{sync}\, A_1 || ... || t_n : \mathsf{sync}\, A_n]\!]_E \rightarrowtail_P^* S$$

where $S = [\![t_{\pi(i)} : \mathsf{sync}\, A_{\pi(i)} || ... || t_{\pi(n)} : \mathsf{sync}\, A_{\pi(n)}]\!]_E$.

In our next result, we state that any final store reached with CHR corresponds to a valid final configuration.

Proposition 6 (Transactional Events via CHR - Completeness) *Let $t_1 : \mathsf{sync}\ A_1 || ... || t_n : \mathsf{sync}\ A_n$ be a valid initial configuration Let $\mathscr{R}$ denote the approximation of the set of all valid traces resulting from*

$t_1 : \mathsf{sync}\, A_1 \| ... \| t_n : \mathsf{sync}\, A_n$. *Let P consist of the set of CHR rules* $[\![\mathscr{R}]\!]_{CHR}$ *and the clean-up rules* $c1-4$. *Then, for any final store S where*

$$[\![t_1 : \mathsf{sync}\, A_1 \| ... \| t_n : \mathsf{sync}\, A_n]\!]_E \rightarrowtail_P^* S$$

we find a trace R and a π *is a permutation on* $\{1,....,n\}$ *such that*

$$t_1 : \mathsf{sync}\, A_1 \| ... \| t_n : \mathsf{sync}\, A_n \overset{R}{\longrightarrow^*} t_{\pi(1)} : \bullet \| ... \| t_{\pi(i-1)} : \bullet \| t_{\pi(i)} : \mathsf{sync}\, A_{\pi(i)} \| ... \| t_{\pi(n)} : \mathsf{sync}\, A_{\pi(n)}$$

and $S = [\![t_{\pi(i)} : \mathsf{sync}\, A_{\pi(i)} \| ... \| t_{\pi(n)} : \mathsf{sync}\, A_{\pi(n)}]\!]_E$.

Proof Sketch. Rules are applied exhaustively which results in a final store S. This ensures that any 'garbage', i.e. constraints connected to synchronized threads, has been removed.

CHR rules in $[\![\mathscr{R}]\!]_{CHR}$ only fire for valid traces. This ensures that we only encounter valid 'big' step CHR rewriting steps which can be replayed in terms of 'small' transactional event reduction steps.

In this result, the presence of constraints of the form $\mathsf{last}(\cdot,\cdot,\cdot)$ becomes important. Constraint $\mathsf{last}(\cdot,\cdot,\cdot)$ ensures that all primitive events in a sequence are synchronized. The issue is that in $\mathscr{R}$ we brute-force generate traces. There might be (valid) traces which only cover parts of a sequence. While CHR rewriting guarantees the atomic removal of a bunch of constraints, we still must ensure that these constraints belong to a complete sequence. Hence, the use of $\mathsf{last}(\cdot,\cdot,\cdot)$.

Constraints of the form $\mathsf{sync}(\cdot)$ ensure that once a branch takes part in a synchronization step, all other branches belonging to the same thread can no longer be synchronized. We could actually do without this form assuming that cleanup rules are applied aggressively. However, it is convenient to have $\mathsf{sync}(\cdot)$ to explicitly track which threads are pending and which threads have been synchronized. □

4.3 Improvements to CHR Encoding

The approach laid out by Proposition 6 is to brute-force generate a large set of *propositional* CHR rules to capture the semantics for a *fixed* initial configuration. For specific classes of transactional events we can do much better. We will show that for some forms of transactional events it is possible to define a fixed *generic* set of CHR rules which captures the semantics for *any* initial configuration.

First, we consider the class of events as found in Concurrent ML [14].

Definition 15 (CML Events) *We refer to a* CML *event as any event E which only consist of send/receive primitives and alternatives but no sequences.*

For CML events, the forms E and A can be used interchangeably.

Traces for CML events can all be broken into simple forms.

Proposition 7 (CML Synchronizations are Simple) *Let* $t_1 : \mathsf{sync}\, E_1 \| ... \| t_n : \mathsf{sync}\, E_n$ *be a valid initial configuration where* E_i *are CML events such that*

$$t_1 : \mathsf{sync}\, E_1 \| ... \| t_n : \mathsf{sync}\, E_n \overset{R}{\longrightarrow^*} t_{\pi(1)} : \bullet \| ... \| t_{\pi(i-1)} : \bullet \| t_{\pi(i)} : \mathsf{sync}\, E_{\pi(i)} \| ... \| t_{\pi(n)} : \mathsf{sync}\, E_{\pi(n)}$$

for some R where π *is a permutation on* $\{1,....,n\}$. *Then, for any* $!?x(t_i,b_i,1) \leftrightarrow (t_j,b_j,1)$ *in R we find that* $t_i : \mathsf{sync}\, E_i \| t_j : \mathsf{sync}\, E_j \longrightarrow^* \bullet \| \bullet$.

Proof Sketch. By observing the structure of CML events and the semantic rules applicable on them. □

This suggests the following generic CHR encoding for CML events.

Definition 16 (CML CHR Encoding) *We define P_{CML} to be the following set of rules:*

$$c5@!x(t,b,p) \Longleftrightarrow \mathsf{comm(x,SND,t,b,p)}$$

$$c6@?x(t,b,p) \Longleftrightarrow \mathsf{comm(x,RCV,t,b,p)}$$

$$c7@ \begin{array}{l}\mathsf{sync}(t_1),\mathsf{comm(x,SND,t_1,b_1,1),last(t_1,b_1,1)},\\ \mathsf{sync}(t_2),\mathsf{comm(y,RCV,t_2,b_2,1),last(t_2,b_2,1)}\end{array} \Longleftrightarrow t_1 \neq t_2 \wedge x = y \mid \mathsf{cleanup}(t_1),\mathsf{cleanup}(t_2)$$

plus rules $c1-4$ from Definition 14.

We introduce an auxiliary predicate symbol comm such that we can define rules which operate on arbitrary channels. Rules $c5$ and $c6$ transform the specific predicate symbols $!x$ and $?x$ into the more general form. Technically, we require for each channel name x instances of rules $c5$ and $c6$. However, it should be clear that we could immediately apply this step in the transformation from events to constraints. Recall Definition 12. For CML events, all traces can be split into a simple form. Then, rule $c7$ is sufficient to capture the semantics of CML events. We assume that SND and RCV are constructors to represent send and receive.

It is straightforward to restate Propositions 5 and 6 for CML events where the set of CHR is in use equals P_{CML}. For brevity, we omit the details.

This approach extends to CML events which concurrently execute w.r.t. a single transaction.

Definition 17 (Single Transaction) *We say that $t_1 : \mathsf{sync}\, A_1 \| ... \| t_n : \mathsf{sync}\, A_n$ is a* single transaction *initial configuration if $A_{\pi(2)}$, ..., $A_{\pi(n)}$ are CML events for some permutation π on $1,...,n$.*

By assumption there is a single transaction whose maximum sequence length is k. Hence, we need to consider all combinations such that a sequence up to length $m <= k$ synchronizes with m other threads composed of CML events.

Definition 18 (Single Transaction CHR Encoding) *Let $t_1 : \mathsf{sync}\ A_1 \| ... \| t_n : \mathsf{sync}\ A_n$ be a valid, single transaction initial configuration where the maximal length of a sequence is k.*

For each $m \in \{1,...,k\}$, we define a CHR

$$c8^m @ \begin{array}{l}\mathsf{sync}(t'_0),\mathsf{comm(x_1,sr_1,t'_0,b_0,1),...,comm(x_n,sr_m,t'_0,b_0,m),last(t'_0,b_0,j)},\\ \mathsf{sync}(t'_1),\mathsf{comm(x'_1,sr'_1,t'_1,b_1,1),last(t'_1,b_1,1)},\\ \ldots \\ \mathsf{sync}(t'_m),\mathsf{comm(x'_n,sr'_m,t'_m,b_m,1),last(t'_m,b_m,1)} \\ \qquad\qquad\qquad\qquad \Longleftrightarrow \\ \quad \begin{array}{l} t'_i \neq t'_j \textit{ for } i \neq j \textit{ and} \\ x_j = x'_j, sr_j \neq sr'_j \textit{ for } j = 1,...,m \end{array} \mid \mathsf{cleanup}(t_0),...,\mathsf{cleanup}(t_m)\end{array}$$

We define P_{ST^k} to be the set of rules consisting of $c8^1,...,c8^k$ plus rules $c1-4$ from Definition 14 and $c5-6$ from Definition 16.

To avoid confusion with the thread numbers of the initial configuration t_1 : $\mathsf{sync}\ A_1\|...\|t_n$: $\mathsf{sync}\ A_n$ we use 'prime' for thread variables in rule $c8^m$. Guard "$t'_i \neq t'_j$ for $i \neq j$" ensures that we synchronize among distinct threads. Guard "$x_j = x'_j, sr_j \neq sr'_j$ for $j = 1,...,m$" guarantees that we only consider proper rendezvous pairs which match on the channel and perform complementary operations. For $m = 1$, rule $c8^m$ boils down to rule $c7$.

5 Related Work and Conclusion

Similar in spirit to our work, the work in [20] shows how to encode low-level lock-based constructs in terms of the join calculus [6]. There exists a close connection between CHR and join calculus [11]. However, the join calculus lacks guard constraints which are essential in our CHR encoding of transactional events.

Earlier works on transactional events [2] focus on fairness issues [1], programming idioms [9] and the interaction with mutable references [5]. To the best of our knowledge, we are the first to draw formal connections between transactional events and CHR. Our technical results in Section 4 have some important implications which we highlight in the following.

Proposition 4 establishes a close connection between semantic reductions of transactional events and CHR rewriting. This suggests that many of the CHR methods for execution and optimizations [3, 17, 4, 15] can be applied to the setting of transactional events. This is something we plan to investigate in detail in future work.

For specific cases, configurations consisting of either CML events or a single transaction only, we can generate a generic set of CHR rules. See Section 4.3. Hence, we can easily derive an executable semantics by means of CHR. There are highly-optimized CHR implementations [16] and it will be interesting to compare the CHR encoding against native implementations of transactional events. Yet another direction for future work is the application of CHR for program analysis purposes where we use CHR to abstract the run-time behavior of transactional events.

Our main focus has been how to express transactional events in terms of CHR. We believe that the other direction is also possible. That is, to encode CHR in terms of transactional events. Briefly, for propositional CHR where there are no guard conditions we could encode rule heads as sequence of receive primitives and matching constraints in the store as their send counterpart. Actual implementations of transactional events [2] support a form of a guarded receive. Hence, more expressive forms of CHR with guards seem feasible. The details need yet to be worked out.

References

[1] Edward Amsden & Matthew Fluet (2012): *Fairness for Transactional Events*. In: *Proc. of IFL'11*, Springer-Verlag, Berlin, Heidelberg, pp. 17–34.

[2] Kevin Donnelly & Matthew Fluet (2008): *Transactional events*. *J. Funct. Program.* 18(5-6), pp. 649–706. Available at `http://dx.doi.org/10.1017/S0956796808006916`.

[3] Gregory J. Duck, Peter J. Stuckey, Maria J. García de la Banda & Christian Holzbaur (2004): *The Refined Operational Semantics of Constraint Handling Rules*. In: *Proc. of ICLP'04*, LNCS, Springer, pp. 90–104.

[4] Gregory J. Duck, Peter J. Stuckey & Martin Sulzmann (2007): *Observable Confluence for Constraint Handling Rules*. In: *Proc. of ICLP'07*, *LNCS* 4670, Springer, pp. 224–239.

[5] Laura Effinger-Dean, Matthew Kehrt & Dan Grossman (2008): *Transactional Events for ML*. *SIGPLAN Not.* 43(9), pp. 103–114.

[6] Cédric Fournet & Georges Gonthier (1996): *The Reflexive CHAM and the Join-calculus*. In: Proc. of POPL'96, ACM, New York, NY, USA, pp. 372–385. Available at `http://doi.acm.org/10.1145/237721.237805`.

[7] Thom Frühwirth (1994): *Theory and Practice of Constraint Handling Rules*. The Journal of Logic Programming 37(1-3), pp. 95–138.

[8] C. A. R. Hoare (1978): *Communicating Sequential Processes*. Commun. ACM 21(8), pp. 666–677. Available at `http://doi.acm.org/10.1145/359576.359585`.

[9] Matthew Kehrt, Laura Effinger-Dean, Michael Schmitz & Dan Grossman (2009): *Programming Idioms for Transactional Events*. In: Proc. of PLACSES'09, EPTCS 17, pp. 43–48.

[10] Edmund S. L. Lam & Martin Sulzmann (2011): *Concurrent goal-based execution of Constraint Handling Rules*. TPLP 11(6), pp. 841–879. Available at `http://dx.doi.org/10.1017/S147106841000044X`.

[11] Edmund S.L. Lam & Martin Sulzmann (2008): *Finally, A Comparison Between Constraint Handling Rules and Join-Calculus*. In: Proc. of CHR'08, Fith Workshop on Constraint Handling Rules, RISC Report Series 08-10, University of Linz, Austria, pp. 51–66.

[12] Gordon D. Plotkin (2004): *A structural approach to operational semantics*. J. Log. Algebr. Program. 60-61, pp. 17–139.

[13] John Reppy, Claudio V. Russo & Yingqi Xiao (2009): *Parallel Concurrent ML*. SIGPLAN Not. 44(9), pp. 257–268. Available at `http://doi.acm.org/10.1145/1631687.1596588`.

[14] John H. Reppy (1999): *Concurrent Programming in ML*. Cambridge University Press, New York, NY, USA.

[15] Tom Schrijvers (2005): *Analyses, Optimizations and Extensions of Constraint Handling Rules: Ph.D. Summary*. In: Proc. of ICLP'05, LNCS 3668, Springer, pp. 435–436.

[16] Tom Schrijvers (2011): *The K.U.Leuven CHR system*. In Thom Fruehwirth & Frank Raiser, editors: Constraint handling rules : compilation, execution, and analysis, Books on Demand GmbH, pp. 71–88.

[17] Tom Schrijvers, Peter J. Stuckey & Gregory J. Duck (2005): *Abstract interpretation for constraint handling rules*. In: Proc. of PPDP'05, ACM, pp. 218–229.

[18] Nir Shavit & Dan Touitou (1995): *Software Transactional Memory*. In: Proc. of PODC'95, ACM, New York, NY, USA, pp. 204–213. Available at `http://doi.acm.org/10.1145/224964.224987`.

[19] Jon Sneyers, Peter Van Weert, Tom Schrijvers & Leslie De Koninck (2010): *As time goes by: Constraint Handling Rules*. TPLP 10(1), pp. 1–47. Available at `http://dx.doi.org/10.1017/S1471068409990123`.

[20] Aaron J. Turon & Claudio V. Russo (2011): *Scalable Join Patterns*. SIGPLAN Not. 46(10), pp. 575–594. Available at `http://doi.acm.org/10.1145/2076021.2048111`.