Relating Coloured Petri Nets to Constraint Handling Rules

Hariolf Betz

Faculty of Engineering and Computer Sciences, University of Ulm
hariolf.betz@uni-ulm.de

Abstract. Constraint Handling Rules (CHR) is a declarative rule-based concurrent committed-choice programming language. Petri nets are a well-known formalism for modeling and analysis of concurrent processes. We aim to develop a framework to exploit Petri nets as a tool for the modeling and analysis of CHR programs. In this paper, we show that place/transition nets can easily be embedded into CHR and we develop a translation of a significant segment of CHR into coloured Petri nets (CPN).

1 Introduction

Constraint Handling Rules (CHR)[4–7] is a concurrent committed-choice rule-based programming language developed in the 1990s by Frühwirth as a portable language extension for the implementation of constraint solvers. Over the last decade it has matured into a stand-alone general purpose declarative programming language.
One of the main features of CHR is its inherent concurrency. Though well-known, the feature has scarcely been exploited for parallel execution until the recent past: Frühwirth [8] has proposed a parallel execution model for CHR in 2005 and recent works by Meister [11, 12] document an enormous potential for speedup by parallelization of CHR programs.
Petri nets [13] are a well-known formalism for the modeling and analysis of concurrent processes that has been in use for more than four decades. They are both mathematically and graphically founded and there exists a wide variety of algorithms for their design and analysis.
Therefore, having a general framework for the application of these algorithms for the analysis and design of CHR programs would certainly be beneficial. Due to the concurrent nature of Petri nets, we are confident that such a framework would especially be helpful in the analysis of concurrency properties of CHR programs and thus in their parallelization. This paper presents first results in this promising direction.
Contributions and Overview. As a first step, we design a translation of place/transition nets (P/T nets) – a basic variant of Petri nets into CHR. We show that this standard variant of Petri nets translates to a considerably small subsegment of CHR. This result backs the assumption, that we have to choose a higher-level variant of Petri nets in order to represent a significant subset of CHR.

Our main contribution is the definition of and interesting and significant, yet easy-to-handle subsegment of CHR and a sound and complete translation from this segment of CHR to coloured Petri nets.

- We recall the basics of Constraint Handling Rules (CHR) in Sect. 2.
- We recall the basics of Petri Nets, especially P/T-nets [] and Coloured Petri Nets (CPN) [] in Sect. 3.
- We apply CHR as a simulator for P/T nets in Sect. 4.
- In Sect. 5 we present a translation of a significant segment of CHR into CPN, along with theorems concerning its soundness and completeness as well as an example.

2 Constraint Handling Rules

In this section, we will introduce the syntax and semantics of CHR as a self-contained language. An introductory example program is followed by the formal definition of its syntax and semantics, and we discuss why we chose to consider the abstract semantics of CHR in this paper.

2.1 CHR by Example

The following classic CHR program implements a constraint handler for the partial-order relation *leq* (for less-or-equal). The constraints $\doteq$ and $\top$ are so-called *built-in constraints* which we assume to be handled by a predefined constraint handler. $\doteq$ is syntactic equality and $\top$ is truth.

$$\begin{array}{ll} reflexivity@ & A\, leq\, B \Leftrightarrow A \doteq B \mid \top \\ antisymmetry@ & A\, leq\, B, B\, leq\, A \Leftrightarrow A \doteq B \\ transitivity@ & A\, leq\, B, B\, leq\, C \Rightarrow A\, leq\, C \end{array}$$

Rule *reflexivity* states, that a less-or-equal relation between two equal variables, i.e. a relation of the form $A \leq A$ is redundant. Rule *antisymmetry* states that if both $A{\leq}B$ and $B{\leq}A$ hold, then A and B must be equal. Rule *transitivity* states that if $A{\leq}B$ and $B{\leq}C$ hold, we may conclude that $A{\leq}C$ also holds.
A sample execution of the handler with an initial state $\langle A{\leq}B, B{\leq}C, C{\leq} A; \top\rangle$ could look as follows:

$$\begin{array}{rl} & \langle \underline{A{\leq}B}, \underline{B{\leq}C}, C{\leq}A; \top\rangle \\ \mapsto_{\mathbf{Propagate}(transitivity)} & \langle A{\leq}B, B{\leq}C, \underline{C{\leq}A}, \underline{A{\leq}C}; \top\rangle \\ \mapsto_{\mathbf{Simplify}(antisymmetry)} & \langle A{\leq}B, B{\leq}C, \underline{A{\doteq}C}; \top\rangle \\ \mapsto_{\mathbf{Solve}} & \langle \underline{A{\leq}B}, \underline{B{\leq}A}; A{\doteq}C\rangle \\ \mapsto_{\mathbf{Simplify}(antisymmetry)} & \langle \underline{A{\doteq}B}; A{\doteq}C\rangle \\ \mapsto_{\mathbf{Solve}} & \langle \top; A{\doteq}B \wedge A{\doteq}C\rangle \end{array}$$

Note that the *antisymmetry* rule operationally acts as a substitution, whereas application of the *transitivity* rule results in the *addition* of a constraint to the state, without removing anything.

2.2 The Syntax of CHR

Constraint handling rules have originally been designed to extend an existing programming language with a user-defined constraint handler. Although CHR can extend other that constraint programming languages and even run as a stand-alone programming language, its run-time environment has to provide at least basic constraint handling capabilities.
We therefore distinguish two disjoint sets of constraint symbols: *User-defined constraints* are those constraints that are handled by the user-defined handler, i.e. by the CHR program. By contrast, handling of the *built-in constraints* has to be done by a predefined constraint handler. The actual set of built-in constraints depends on the implementation. However, it contains at least the constraints $\top$, $\bot$ and $\doteq$, where the latter stands for syntactic equailty. A *goal* is multiset conjunction of bulit-in and user-defined constraints.
Programs in CHR consist of two sorts of guarded rules: Simplification rules express the conditional *substitution* of the rule head with the body in the current state. Propagation rules express the conditional *addition* of the body to the current state (without removing the head). The syntax of the simplification rule is $(E \Leftrightarrow C|G)$ and that of the propagation rule is $(E \Rightarrow C|G)$. The rule *head* E is a user-defined constraint, the *guard* C is a built-in constraint and the *body* G is a goal. If the guard equals $\top$, it can be omitted.
The syntax of the rules is mnemonic in that it reflects the declarative meaning of the rules. That is, one would apply a simplification rule if the rule body is logically *equivalent* to the rule head and one would apply a propagation rule is the body is merely *implied* by (but not equivalent to) the head.
At each given point in time, there is exactly one CHR state at each moment of its execution. A CHR state is of the form $\langle G; C \rangle$, where G is a goal and C is a built-in constraint. G is called *goal store* and C is called *constraint store*. Of the two, only the goal store is directly accessible by a CHR program. The constraint store is handled by a predefined constraint handler according to a consistent first-order constraint theory CT. A state of the form $\langle G; \top \rangle$ is called an *initial state*.
The complete syntax of CHR is summarized in the following table:

Built-in constraint:	C,D	$::= \top \mid \bot \mid c(\bar{t}) \mid C \wedge D$	
User-defined constraint:	E,F	$::= e(\bar{t}) \mid E \wedge F$	
Goal:	G,H	$::= C \mid E \mid G \wedge H$	
Simplification rule:	R	$::= (E \Leftrightarrow C \mid G)$	
Propagation rule:	R	$::= (E \Rightarrow C \mid G)$	
CHR program:	P	$::= R_1, \ldots, R_n$	$n \geq 0$
CHR state:	S	$::= \langle G; C \rangle$	
Initial state:	S_0	$::= \langle G; \top \rangle$	

2.3 The Operational Semantics of CHR

There are actually several variants of the operational semantics for CHR. The original operational semantics for CHR was published in [4] and is

known as the *abstract* semantics of CHR. It is the original and most general operational semantics in that every derivation possible in any of the other semantics is also true in the abstract semantics.
In [1], Abdennadher extended the abstract semantics with a token store for propagation rules in order to avoid trivial non-termination. This was extended to the so-called *refined semantics* of CHR [3], which is closest to the execution strategy used in most current implementations of CHR. Examples for other relevant operational semantics include especially the semantics of Probabilistic CHR [6], in which each applicable rule has a (weighted) chance of firing.
We chose to consider the abstract abstract semantics for two reasons: Secondly, it is the most general operational semantics for CHR, i.e. every derivation that is possible in the abstract operational semantics. Secondly, the refined semantics of CHR is inherently deterministic and non-concurrent, whereas Petri nets are in general non-deterministic and concurrent. The transition rules that constitute the abstract operational semantics of CHR are given in Fig. 1.
The **Simplify** transition performs a conditional substitution of a user-defined constraint in the goal store with a different one.
A CHR rule $(F \Leftrightarrow D \mid H)$ is applicable if the rule head F matches a user-defined constraint E in the goal store[1]. Furthermore, the constraint store C of the current program state must imply the rule guard D under the constraint theory CT.
If **Simplify** is applied, the constraint E in goal store is substituted by F, and the variable matching $(E \doteq F)$ as well as the guard D are added to the constraint store.
Propagation differs from simplification in that the matched user-defined constraints E are kept in the constraint store rather than being removed. In practical terms, this raises the question of how to avoid trivial non-termination, which has been addressed in [1] and [3]. Under the abstract semantics, however, a propagation rule $F \Rightarrow D \mid H$ can be faithfully reduced to a simplification rule $F \Leftrightarrow D \mid F \wedge H$. Hence, we will only consider simplification rules in the rest of this paper.
The **Solve** transition moves a built-in constraint from the goal store to the constraint store. We will, however, be able to ignore this transition in the segment of CHR we are using.
We denote by $\mapsto^*$ the reflexive transitive closure of the $\mapsto$ relation.

3 Petri Nets

Petri nets [13] are a well-known formalism for the modeling and analysis of concurrent processes. Of the many existing variants, this section shall introduce place/transition nets (P/T nets) and coloured Petri nets (CPNs) [10]. For the two variants, we will present both the graphical representation and the mathematical foundation.

[1] Note that CHR does not use unification like Prolog but one-sided matching, i.e. the variables in the rule head have to be matched with those in the store, *not* vice versa.

Simplify	
If	$(F \Leftrightarrow D \mid H)$ is a fresh variant of a rule in P with variables $\bar{x}$
and	$CT \models \forall (C \rightarrow \exists \bar{x}(F \doteq E \wedge D))$
then	$\langle E \wedge G; C\rangle \mapsto \langle H \wedge G; (F \doteq E) \wedge D \wedge C\rangle$
Propagate	
If	$(F \Rightarrow D \mid H)$ is a fresh variant of a rule in P with variables $\bar{x}$
and	$CT \models \forall (C \rightarrow \exists \bar{x}(F \doteq E \wedge D))$
then	$\langle E \wedge G; C\rangle \mapsto \langle E \wedge H \wedge G; (F \doteq E) \wedge D \wedge C\rangle$
Solve	
If	$CT \models (C \wedge D_1) \leftrightarrow D_2$
then	$\langle C \wedge G; D_1\rangle \mapsto \langle G; D_2\rangle$

Fig. 1. CHR$^\vee$ transition rules

3.1 Petri Nets by Example

Among the most important features of Petri nets is the fact that they are both mathematically and graphically founded. In this section we will introduce by example the graphical representation of two important variants of Petri nets: Place/transition nets (P/T nets) and coloured Petri nets (CPN).
We will model variants of the dining philosophers problem, which is a classic example for a computing problem in concurrency. The problem consists of five (or an arbitrary number of) philosophers sitting at a round table and doing two different things – eating and thinking. Between each philosopher a fork is placed such that each philosopher has a fork to his left and one two his right. In order to eat, a philosopher needs two forks, i.e. the one to his left and the one to his right. At arbitrary points in time, a thinking philosopher will change his state from thinking to eating – provided the forks to his immediate left and right are available. After a while, an eating philosopher will change his state back to thinking, thereby releasing the forks to his left and right and thus making them available to his immediate neighbors again.
In the following, we will assume that the philosophers and the forks are numbered: Philosopher #1 needs fork #1 and fork #2 to eat, philosopher #2 needs fork #2 and #3, and so on.

Example 1. In Fig. 2 we have modeled the dining philosophers problem for three philosophers as a Petri net. Obviously, the net is a bipartite, directed graph with two types of nodes: *places* which we draw as circles and *transitions* which we draw as rectangles. There is a *token* in six of the nine places. (Only places can carry tokens.) Both places and transitions are annotated with their names.
The places $t_1, \ldots, t_3$ respectively represent the fact that philosopher #1, #2, #3 is eating. When a place carries a token, we assume that the respective fact is true. The places $e_1, \ldots, e_3$ represent the facts that the philosophers are eating. As they do not carry tokens, these facts do not

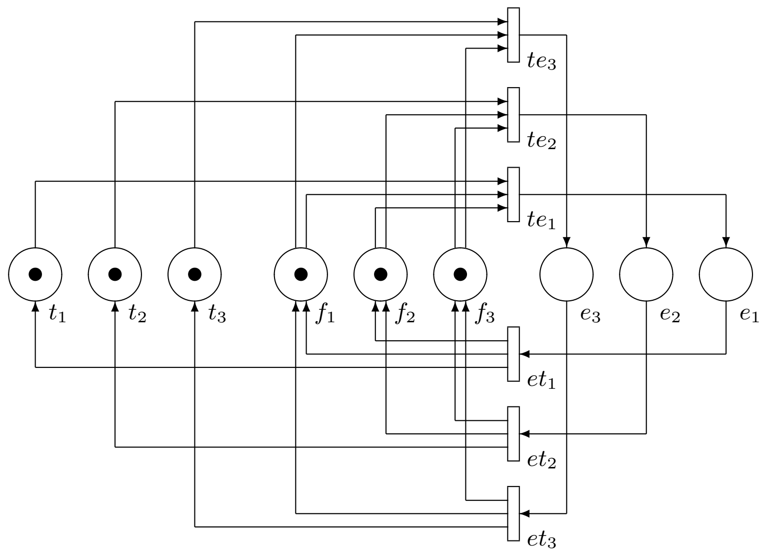

Fig. 2. The dining philosophers problem modeled as a Petri net

hold at the moment. (Obviously, if our model is correct, it should never be possible for a philosopher to eat and think at the same time.) The places $f_1, \ldots, f_3$ each represent the fact that the respective fork is not in use and therefore available. The distribution of tokens in a net at one point in time is called a *marking* of the net.
The transitions allow for changes in the net: I there is at least one token at every arc leading *into* a transition, we call the transition *active*. An activated transition can *fire*. In this case, one token will be removed from every place from which an arc leads *into* the transition, and one token will be added to every place into which an arc leads *from* the transition. As for example, the transition $te1$ represents the event in which philosopher #1 stops thinking and takes up eating. In the figure, the transition is currently active and could be fired. On firing the transition, tokens would be removed from places t_1, f_1, f_2 and one would be added to place e_1. Thus, the facts that philosopher #1 is thinking and that fork #1 and #2 are available would become untrue, whereas the fact that philosopher #1 is eating would become true.

As a slight modification of a net of the type presented in Example 1, we could annotate each arc with a weight $n \in \mathbb{N}$. On firing a transition, we would not necessarily remove or add exactly one token from the adjoining places, but a number of tokens equal to the weight of the connecting arc. Jantzen and Valk have introduced the notion place/transition net (P/T net)[9] for this type of Petri net.

Example 2. In this example, we will model the dining philosophers problem with three philosophers as a so-called coloured Petri net. The net is given in Fig. 3. Note that we have changed the numbering of the philosophers to counting from zero.

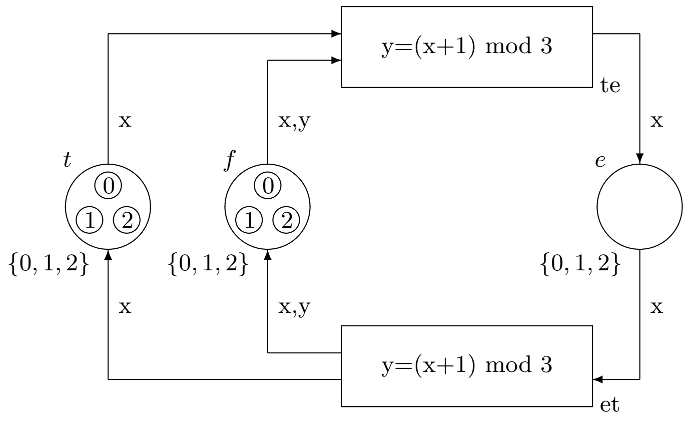

Fig. 3. The dining philosophers problem with three philosophers as a CPN

Obviously, the net is much less complex than the one in Example 1, consisting only of three places t, f, e and two transitions et, te. At the same time, the tokens are no longer *anonymous* – i.e. black dots. Instead they are inscribed with natural numbers $0, 1, 2$. We say that the tokens are *coloured* and the numbers $0, 1, 2$ are the *colors*. Places are not only inscribed with their names – t, f, e – but also with their colour domains, i.e. the set of colours that tokens in the respective places may have.
In our model, a token of color 1 in place t represents that philosopher #1 is thinking, a token of color 2 in place f represents that fork #2 is available, a token of color 0 in place e represents that philosopher #0 is eating, and so on.
Transitions are inscribed with formulae and arcs are inscribed with bags (multisets) of parametrized terms. More precisely, the inscriptions are mappings from *variable bindings* to bags over the colour domain of the associated place. The formulae that transitions are inscribed with can be understood as a mapping from variable bindings to a truth value.
A transition is *active* for a specific variable binding if (1) the inscription of each incoming arc maps the variable binding to a bag of tokens that is contained in the bag of tokens that the associated place currently holds and (2) if the formula inscribed in the transition evaluates to true for that binding. If a transition is fired with a specific variable binding, the respective bags of tokens are removed from the places associated with the incoming arcs and tokens are added to the places associated with the

outgoing arcs according to the insciptions on those arcs. The set of all variable bindings for which a transition could possibly fire is called the *colour domain* of the transition.
In our example, firing transition te with the variable binding $[x = 0, y = 1]$ would remove the bag $\{0\}$ from place t, remove the bag $\{0, 1\}$ from place f and add the bag $\{0\}$ to place e.

3.2 Formal Definitions

In this section we present the formal definitions of place/transition nets and coloured Petri nets.

Definition 1 (Place/Transition Net). *A* place/transition net *(P/T net) is a tuple* $\mathcal{N} = \langle Pl, Tr, \mathbf{Pre}, \mathbf{Post} \rangle$ *where*
- *Pl is the finite set of* places
- *Tr is the finite set of* transitions
- $\mathbf{Pre} \in \mathbb{N}^{|Pl| \times |Tr|}$ *is the* backward incidence matrix
- $\mathbf{Post} \in \mathbb{N}^{|Pl| \times |Tr|}$ *is the* forward incidence matrix

$\mathcal{C} = \mathbf{Post} - \mathbf{Pre}$ *is called the* incidence matrix *of $\mathcal{N}$. For a transition $tr \in Tr$, the expressions $\mathbf{Pre}[\bullet, tr]$ and $\mathbf{Post}[\bullet, tr]$ denote the column vectors in* **Pre** *and* **Post** *associated with tr.*

In the graphical representation of $\mathcal{N}$, there is a an incoming arc from a place $pl \in Pl$ to a transition $tr \in Tr$ iff $\mathbf{Pre}[pl, tr] > 0$ where the weight of the arc is $n = \mathbf{Pre}[pl, tr]$. There is a an outgoing arc from a transition $tr \in Tr$ to a place $pl \in Pl$ iff $\mathbf{Post}[pl, tr] > 0$ where the weight of the arc is $n = \mathbf{Post}[pl, tr]$.

Definition 2 (Marking of a P/T Net). *A* marking *of a P/T net $\mathcal{N} = \langle Pl, Tr, \mathbf{Pre}, \mathbf{Post} \rangle$ is a vector $\mathbf{m} \in \mathbb{N}^{|Pl|}$. A transition $tr \in Tr$ is* enabled *in a marking $\mathbf{m}$ if $\mathbf{m} \geq \mathbf{Pre}[\bullet, tr]$, where $\geq$ denotes component-wise comparison.*

Definition 3 (Successor-Marking Relation on P/T Nets). *For an enabled transition $tr \in Tr$, the* successor-marking relation *between two markings $\mathbf{m}, \mathbf{m}'$ is defined as $\mathbf{m} \xrightarrow{tr} \mathbf{m}' \Leftrightarrow \mathbf{m} \geq \mathbf{Pre}[\bullet, tr] \wedge \mathbf{m}' = \mathbf{m} + \mathcal{C}[\bullet, tr]$.*
For a possibly empty sequence of transitions $\omega \in Tr^*$ $\mathbf{m} \xrightarrow{\omega tr} \mathbf{m}' \Leftrightarrow \exists \mathbf{m}''.\mathbf{m} \xrightarrow{\omega} \mathbf{m}'' \wedge \mathbf{m}'' \xrightarrow{tr} \mathbf{m}'$. If ω is the empty word, $\mathbf{m} \xrightarrow{\omega} \mathbf{m}$ holds.

Example 3. The Petri net in Example 1 can be described formally as $\mathcal{N} = \langle Pl, Tr, \mathbf{Pre}, \mathbf{Post} \rangle$ where
- $Pl = \{t_1, t_2, t_3, f_1, f_2, f_3, e_1, e_2, e_3\}$
- $Tr = \{te_1, te_2, te_3, et_1, et_2, et_3\}$
- $\mathbf{Pre} = \begin{array}{c} (t_1) \\ (t_2) \\ (t_3) \\ (f_1) \\ (f_2) \\ (f_3) \\ (e_1) \\ (e_2) \\ (e_3) \end{array} \left(\begin{array}{cccccc} 1 & . & . & . & . & . \\ . & 1 & . & . & . & . \\ . & . & 1 & . & . & . \\ 1 & . & 1 & . & . & . \\ 1 & 1 & . & . & . & . \\ . & 1 & 1 & . & . & . \\ . & . & . & 1 & . & . \\ . & . & . & . & 1 & . \\ . & . & . & . & . & 1 \end{array}\right)$ $\mathbf{Post} = \begin{array}{c} (t_1) \\ (t_2) \\ (t_3) \\ (f_1) \\ (f_2) \\ (f_3) \\ (e_1) \\ (e_2) \\ (e_3) \end{array} \left(\begin{array}{cccccc} . & . & . & 1 & . & . \\ . & . & . & . & 1 & . \\ . & . & . & . & . & 1 \\ . & . & . & 1 & . & 1 \\ . & . & . & 1 & 1 & . \\ . & . & . & . & 1 & 1 \\ 1 & . & . & . & . & . \\ . & 1 & . & . & . & . \\ . & . & 1 & . & . & . \end{array}\right)$

For clarity, we have replaced zeroes with dots in the above matrices. The initial marking of the state is denoted as $\mathbf{m}_0 = [t_1 = 1, t_2 = 1, t_3 = 1, f_1 = 1, f_2 = 1, f_3 = 1, e_1 = 0, e_2 = 0, e_3 = 0]$.

In order to define coloured Petri nets, we need to the notion of *bags* (*multisets*).

Definition 4 (Bag). *A* bag *bg over a non-empty set A is a function* $bg : A \rightarrow \mathbb{N}$*. If there is no ambiguity, we may omit the curly brackets.*
Assuming that bg_1, bg_2 *are bags,* partial order*,* sum *and* difference *are defined as follows:*
- $bg_1 \leq bg_2 \Leftrightarrow \forall a \in A.(bg_1(a) \leq bg_2(a)'a)$
- $bg_1 + bg_2 = \sum_{a \in A}(bg_1(a) + bg_2(a)'a)$
- $bg_1 - bg_2 = \sum_{a \in A}(bg_1(a) - bg_2(a)'a)$ *if* $bg_2 \leq bg_1$

For a non-empty set A, Bag(A) denotes the set of all bags over A.

Definition 5 (Coloured Petri Net). *A* coloured Petri net *(P/T net) is a tuple* $\mathcal{N} = \langle Pl, Tr, \mathbf{Pre}, \mathbf{Post}, \mathcal{C}, cd \rangle$ *where*
- Pl *is the finite set of* places
- Tr *is the finite set of* transitions
- $\mathcal{C}$ *is the set of* colour classes
- $cd : Pl \cup Tr \rightarrow \mathcal{C}$ *is the colour domain mapping*
- $\mathbf{Pre} \in \mathcal{B}^{|Pl| \times |Tr|}$ *is the* backward incidence matrix
- $\mathbf{Post} \in \mathcal{B}^{|Pl| \times |Tr|}$ *is the* forward incidence matrix

In the above definition, $\mathcal{B}$ *is the set of mappings of the form* $f : cd(t) \rightarrow Bag(cd(p))$*.* $\mathcal{C} = \mathbf{Post} - \mathbf{Pre}$ *is called the* incidence matrix *of* $\mathcal{N}$*.* $\mathbf{Pre}[\bullet, tr]$ *and* $\mathbf{Post}[\bullet, tr]$ *again denote the column vectors in* $\mathbf{Pre}$ *and* $\mathbf{Post}$ *associated with a transition* $tr \in Tr$*.*

Note that each matrix entry $\mathbf{Pre}[pl, tr]$ and $\mathbf{Post}[pl, tr]$ is a mapping from $cd(tr)$ to $Bag(cd(pl))$.

Definition 6 (Marking of a CPN). *A* marking *of a CPN* $\mathcal{N} = \langle Pl, Tr, \mathbf{Pre}, \mathbf{Post}, \mathcal{C}, cd \rangle$ *is a vector* $\mathbf{m}$ *such that* $\mathbf{m}[pl] \in Bag(cd(pl))$ *for each* $pl \in Pl$*. A transition* $tr \in Tr$ *is* enabled *for a binding* β *in a marking* $\mathbf{m}$ *if* $\mathbf{m} \geq \mathbf{Pre}[\bullet, tr](\beta)$*.*

Definition 7 (Successor-Marking Relation on CPN). *For an enabled transition* $tr \in Tr$ *and a variable binding* β*, the* successor relation *between two markings* $\mathbf{m}, \mathbf{m}'$ *is defined as* $\mathbf{m} \xrightarrow{tr,\beta} \mathbf{m}' \Leftrightarrow \mathbf{m} \geq \mathbf{Pre}[\bullet, tr](\beta) \wedge \mathbf{m}' = \mathbf{m} + \mathcal{C}[\bullet, tr](\beta)$*. If the concrete binding* β *is not important, we write* $\mathbf{m} \xrightarrow{tr} \mathbf{m}'$ *instead of* $\mathbf{m} \xrightarrow{tr,\beta} \mathbf{m}'$*.*
For a possibly empty sequence of transitions $\omega \in Tr^*$ $\mathbf{m} \xrightarrow{\omega tr} \mathbf{m}' \Leftrightarrow \exists \mathbf{m}''.\mathbf{m} \xrightarrow{\omega} \mathbf{m}'' \wedge \mathbf{m}'' \xrightarrow{tr} \mathbf{m}'$*. If* ω *is the empty word,* $\mathbf{m} \xrightarrow{\omega} \mathbf{m}$ *holds.*

Example 4. The Petri net in Example 2 can be described formally as $\mathcal{N} = \langle Pl, Tr, \mathbf{Pre}, \mathbf{Post}, \mathcal{C}, cd \rangle$ where
- $Pl = \{t, f, e\}$
- $Tr = \{te, et\}$
- $cd(t) = cd(f) = cd(e) = \{0, 1, 2\}$
- $cd(te) = \left\{(x, y) \in \{1, 2, 3\}^2 \mid y = (x + 1) \, mod \, 3\right\}$
- $cd(et) = \{1, 2, 3\}$
- $\mathbf{Pre} = \begin{array}{c} (t) \\ (f) \\ (e) \end{array} \begin{pmatrix} \{x\} & \emptyset \\ \{x, y\} & \emptyset \\ \emptyset & \{x\} \end{pmatrix}$ $\mathbf{Post} = \begin{array}{c} (t) \\ (f) \\ (e) \end{array} \begin{pmatrix} \emptyset & \{x\} \\ \emptyset & \{x, y\} \\ \{x\} & \emptyset \end{pmatrix}$

4 Embedding P/T Nets in CHR

In this section we discuss a translation from P/T nets to CHR. This translation allows us to simulate P/T nets with any CHR implementation that features a fair selection strategy. On the other hand, it allows us to use CHR as a notation for Petri nets.

Definition 8 ($m_{\mathbf{m}}$). *Consider a marking* $\mathbf{m}$ *in a P/T net* $\mathcal{N} = (Pl, Tr, \boldsymbol{Pre}, \boldsymbol{Post})$*. The mapping* $m_{\mathbf{m}} : \mathbf{m} \mapsto S$ *which maps* $\mathbf{m}$ *to a CHR state is defined by:*

$$S = \langle \bigwedge_{i=1}^{n} pl_i^{\mathbf{m}[pl_i]}; \top \rangle$$

Note that as a marking is defined as an element of $\mathbb{N}^{|Pl|}$. Consequently, $m_{\mathbf{m}}$ is also defined on column vectors in **Pre** and **Post**, which we will apply in the following definition.

Definition 9 (m_N). *Let* $\mathcal{N} = (Pl, Tr, \boldsymbol{Pre}, \boldsymbol{Post})$ *be a P/N net, where we assume that* $Pl = \{pl_1, \dots, pl_n\}$ *and* $Tr = \{tr_1, \dots, tr_k\}$*. The mapping* $m_N : \mathcal{N} \mapsto P$ *which maps the net to a CHR program is defined by:*

$$P = R_1 R_2 \dots R_k$$

$$R_j = tr_j @ m_{\mathbf{m}} (\mathbf{Pre}[\bullet, tr_j]) \Leftrightarrow m_{\mathbf{m}} (\mathbf{Post}[\bullet, tr_j])$$

Example 5. Consider again the Petri net system representation of the dining philosophers problem in Example 1. The net translates to the following CHR program:

$$\begin{array}{l} te1@\ t1, f1, f2 \Leftrightarrow e1. \\ te2@\ t2, f2, f3 \Leftrightarrow e2. \\ te3@\ t3, f3, f1 \Leftrightarrow e3. \\ \\ et1@\ e1 \Leftrightarrow t1, f1, f2. \\ et2@\ e2 \Leftrightarrow t2, f2, f3. \\ et3@\ e3 \Leftrightarrow t3, f3, f1. \end{array}$$

The initial marking $\mathbf{m}_0$ translates to the CHR state $S_0 = \langle m1 \wedge m2 \wedge m3 \wedge f1 \wedge f2 \wedge f3; \top \rangle$.

Theorem 1 (Soundness and Completeness). *Consider a P/T net* $\mathcal{N} = (Pl, Tr, \mathbf{Pre}, \mathbf{Post})$ *and a CHR program* P *such that* $P = m_N(\mathcal{N})$*. Let* $\mathbf{m}_0, \mathbf{m}_n$ *be markings in* $\mathcal{N}$ *and let* S_0, S_n *be CHR states in* P *such that* $S_0 = m_{\mathbf{m}}(\mathbf{m}_0)$ *and* $S_n = m_{\mathbf{m}}(\mathbf{m}_n)$*. Then we have*

$$S_0 \mapsto^* S_n \text{ iff } \exists \omega \in Tr^*. \mathbf{m}_0 \xrightarrow{\omega} \mathbf{m}_n \quad (1)$$

Proof of Theorem 1 As P contains only 0-ary user-defined constraints and no built-in constraints, P is in a segment of CHR where we reduce the **Simplify** transition to the following form:

$$\boxed{\begin{array}{l}\text{If} \quad (F \Leftrightarrow H) \text{ is a rule in P} \\ \text{then } \langle F \wedge G; \top \rangle \mapsto \langle H \wedge G; \top \rangle\end{array}}$$

Consider a marking $\mathbf{m}_1$ of $\mathcal{N}$ such that $\exists tr \in Tr.\mathbf{m}_0 \xrightarrow{tr} \mathbf{m}_1$. This is equivalent to $\exists tr \in Tr, \tilde{\mathbf{m}}. (\mathbf{m}_0 = \mathbf{Pre}[\bullet, tr] + \tilde{\mathbf{m}} \wedge \mathbf{m}_1 = \mathbf{Post}[\bullet, tr] + \tilde{\mathbf{m}})$.
As $tr \in Tr$, there must be a rule $R \in P$ with $R = (tr@ \ F \Leftrightarrow H)$ where $F = m_{\mathbf{m}}(\mathbf{Pre}[\bullet, tr])$ and $H = m_{\mathbf{m}}(\mathbf{Post}[\bullet, tr])$. From $\mathbf{m}_0 = \mathbf{Pre}[\bullet, tr] + \tilde{\mathbf{m}}$ and $\mathbf{m}_1 = \mathbf{Post}[\bullet, tr] + \tilde{\mathbf{m}}$ follows that $S_0 = F \wedge m_{\mathbf{m}}(\tilde{\mathbf{m}})$ and $S_1 = H \wedge m_{\mathbf{m}}(\tilde{\mathbf{m}})$. Hence, we can apply the **Simplify** transition to yield $S_0 \mapsto S_1$.
Analogously, we can show that $S_0 \mapsto S_1$ implies $\exists tr \in Tr.\mathbf{m}_0 \xrightarrow{tr} \mathbf{m}_1$. Therefore, we have

$$S_0 \mapsto S_1 \text{ iff } \exists tr \in Tr.\mathbf{m}_0 \xrightarrow{tr} \mathbf{m}_1$$

Theorem 1 follows by induction.□
Besides the fact that we can use CHR for the simulation and for a simpler textual notation of P/T nets, our result shows, that P/T nets correspond to a rather insignificant segment of CHR, i.e. the segment of 0-ary user-defined constraints. This insight backs our approach of using a higher-level variant of Petri nets for the representation of CHR programs in the following section.

5 Translation of CHR Programs into Coloured Petri Nets

In this section we will present a faithful translation of CHR programs in the *positive range-restricted ground* segment of CHR into coloured Petri nets. A constraint handling rule is called *range-restricted* if grounding all the variables in the head results in all the variables in the guard and body being ground as well. A constraint handling rule is called *positive* if there are no built-in constraints in the body except $\top$.
In positive range-restricted ground CHR, all rules are positive and range-restricted and all queries are ground. The range-restricted ground segment is a quite common segment of CHR, it was e.g. recently investigated in [2].
Additionally, we require that programs consist only of simplification rules. However, as we consider only the abstract operational semantics, a propagation rule of the form $N@F \Rightarrow D \mid H$ is equivalent to a simplification rule $N@F \Leftrightarrow D \mid F \wedge H$.
Note that in the positive segment of CHR, derivations cannot fail as a built-in constraint in the guard will never cause the constraint store to

evaluate to $\bot$. A non-positive CHR rule of the form $N@F \Leftrightarrow D_1 \mid D_2 \wedge H$ can be emulated in positive CHR by the two rules given below, where we assume that $false$ is a 0-ary user-defined constraint that does not appear in the head of any CHR rule:

$$\begin{array}{l} N_1@\ F \Leftrightarrow D_1 \wedge D_2 \mid H \\ N_2@\ F \Leftrightarrow D_1 \wedge \neg D_2 \mid false \wedge H \end{array}$$

Lemma 1 states an important property of the positive range-restricted ground segment of CHR.

Lemma 1. *In the positive range-restricted ground segment of CHR, every state $\langle E; C\rangle$ has an operationally equivalent normal form $\langle E'; \top\rangle$ where the built-in store equals $\top$.*

Proof of Lemma 1 We assume that every state S in our segment of CHR is of the form $S = \langle E; X \doteq T \wedge C'\rangle$, where E is a goal, $X \doteq T$ is a (possibly empty) conjunction of syntactic equality constraints that bind all the variables in S to ground terms and C' is the (possibly empty) conjunction of all other built-in constraints.
As our segment is range-restricted and ground, *all* variables in C' are bound to ground terms by $X \doteq T$. As CT is complete, C' must therefore evaluate to either $\top$ or $\bot$ under this binding. In the positive segment, the constraint store cannot evaluate to $\bot$, therefore we have:

$$CT \models (X \doteq T) \rightarrow C'$$

Consequently, the state $\langle E; X \doteq T \wedge C'\rangle$ is operationally equivalent to $\langle E; X \doteq T\rangle$. This in turn can be transformed to another state in which all variables in E are substituted by the ground terms to which they are bound by $X \doteq T$. We denote this as $\langle E\,[X \doteq T]\,; \top\rangle$.
In the range-restricted ground segment, this state is again operationally equivalent to S. We will call this the *normal form* of S. $\Box$

In the following, we will assume that CHR states in the positive range-restricted ground segment are *always* in normal form. We can therefore reduce the **Simplify** transition for this segment to the following:

$$\boxed{\begin{array}{ll} \text{If} & (F \Leftrightarrow D|H) \text{ is a fresh variant of a rule in P with variables } \bar{x} \\ \text{and} & CT \models \exists\bar{x}(F \doteq E \wedge D) \\ \multicolumn{2}{l}{\text{then } \langle E \wedge G; \top\rangle \mapsto \langle H\,[F \doteq E] \wedge G; \top\rangle} \end{array}}$$

Informal construction of a CPN from a CHR program: Informally, we construct the mapping m_P from a program P to a coloured Petri net $\mathcal{N}$ as follows:
For each constraint name e_i appearing in P there is a place e_i in $\mathcal{N}$ and for each rule named n_j in P there is a transition n_j in $\mathcal{N}$. There is an arc from a place e_i to a transition n_j iff e_i appears in the head of the rule named n_j. Similarly, there is an arc from a transition n_j to a place e_i iff e_i appears in the body of rule n_j.

Inscriptions are constructed as follows: An incoming arc from a place e_i to a transition n_j is inscripted with the argument terms of all occurrences of e_i in the *head* of n_j. Similarly, an outgoing arc from a transition n_j to a place e_i is inscripted with the argument terms of all occurrences of e_i in the *body* of n_j. Finally, each transition n_j is inscribed with the guard of the rule named n_j.
CHR states translate to markings in that the ground argument term of each occurrence of a user-defined constraint e_i is added as a token to the place e_i.

Example 6. Consider the following CHR program implementing the dining philosophers problem with three philosophers.

$$\begin{array}{l} te@\ t(X), f(X), f(Y) \Leftrightarrow Y = (X+1)\, mod\, 3 \mid e(X). \\ et@\ e(X) \Leftrightarrow Y = (X+1)\, mod\, 3 \mid t(X), f(X), f(Y). \end{array}$$

This program maps exactly to the coloured Petri net representation of the dining philosophers problem in Example 2.

Definition 10 **(m_E).** *The mapping $m_E : E \mapsto \mathbf{m}$ from user-defined constraints to markings is defined by:*

$$m_E\,(E \wedge F) = m_E(E) + m_E(F)$$
$$m_E\left(e'(\bar{t})\right)[e] = \begin{cases} \{\bar{t}\} & \textit{if } e = e' \\ \emptyset & \textit{if } e \neq e' \end{cases} \quad \textit{for any user-defined constraint symbol } e$$

Definition 11 **(m_S).** *Consider a positive range-restricted ground CHR state $\langle E; \top\rangle$. The mapping $m_S : \langle E; \top\rangle \mapsto \mathbf{m}$ is defined by $m_S(\langle E; \top\rangle) = m_E(E)$.*

Definition 12 **(m_P).** *Consider a range-restricted ground CHR program consisting of l simplification rules of the form $n_j@F_j \Leftrightarrow D_j \mid H_j$, using the user-defined constraint names $e_1, \ldots, e_k$. We assume w.l.o.g. that constraint names are not overloaded, i.e. each e_i has exactly one arity $ar(e_i)$.*
We define the mapping $m_P : P \mapsto \mathcal{N}$ from CHR programs to coloured Petri nets by $P \mapsto \langle Pl, Tr, \mathbf{Pre}, \mathbf{Post}, \mathcal{C}, cd\rangle$ where
- $Pl = \{e_1, \ldots, e_k\}$
- $Tr = \{n_1, \ldots, n_l\}$
- $cd(e_i) = \mathcal{T}^a$ *where* $a = ar(e_i)$
- $cd(n_j) = \{\beta \mid (CT \models D_j\beta)\}$
- $\mathbf{Pre}\,[\bullet, n_j] = m_E(F_j)$
- $\mathbf{Post}\,[\bullet, n_j] = m_E(H_j)$

In the above definition, let $\mathcal{T}$ be the set of terms.

Theorem 2 (Soundness and Completeness). *Consider a positive range-restricted ground CHR program P and a coloured Petri net $\mathcal{N} = (Pl, Tr, \mathbf{Pre}, \mathbf{Post}, \mathcal{C}, cd)$ such that $\mathcal{N} = m_P(P)$. Let S_0, S_n be CHR states in P and let $\mathbf{m}_0, \mathbf{m}_n$ be markings in $\mathcal{N}$ such that $\mathbf{m}_0 = m_S(S_0)$ and $\mathbf{m}_n = m_S(S_n)$. Then we have*

$$\exists \omega \in Tr^*.\mathbf{m}_0 \xrightarrow{\omega} \mathbf{m}_n \textit{ iff } S_0 \mapsto^* S_n \qquad (2)$$

Proof of Theorem 2 Consider a state S_1 such that $S_0 \mapsto S_1$. We assume that S_0, S_1 are in normal form. From $S_0 \mapsto S_1$ we deduce that the must be a rule $n @ F \Leftrightarrow D \mid H$ in P and the states S_0, S_1 must be of the form $S_0 = \langle E \wedge G \rangle$ and $S_1 = \langle H [F \doteq E] \wedge G \rangle$, respectively. Consequently, there must be a transition $n \in Tr$ such that $\mathbf{Pre}[\bullet, n] = m_E(F)$ and $\mathbf{Post}[\bullet, n] = m_E(H)$.
Let β be the variable binding $[F \doteq E]$. Obviously, $F [F \doteq E] = E$ from which follows that $\mathbf{Pre}[\bullet, n](\beta) = m_E(E)$ and thus $\mathbf{m}_0 = m_E(E) + m_E(G) \geq \mathbf{Pre}[\bullet, n](\beta)$. From the precondition $CT \models \exists \bar{x}(F \doteq E \wedge D)$ we deduce that $CT \models D\beta$. We conclude that for the variable binding β, transition n is active. Hence we have:

$$\mathbf{m}_0 \xrightarrow{tr,\beta} \mathbf{m}_0 - \mathbf{Pre}[\bullet, n](\beta) + \mathbf{Post}[\bullet, n](\beta) = \mathbf{m}_1$$

Similarly, we can show that $\exists tr \in Tr . \mathbf{m}_0 \xrightarrow{tr} \mathbf{m}_1$ implies $S_0 \mapsto S_1$. Therefore, we have

$$\exists tr \in Tr . \mathbf{m}_0 \xrightarrow{tr} \mathbf{m}_1 \text{ iff } S_0 \mapsto S_1$$

Theorem 1 follows by induction.□

6 Conclusion

In this paper, we have presented a first result towards the development of a general framework for the application of algorithms for the analysis and design of Petri nets to CHR programs.
Our first contribution is a sound and complete translation of the full segment of P/T nets to a small segment of CHR. This result backs the assumption that we will have to work with *higher-level* Petri nets in order to achieve useful results for a significant segment of CHR.
Secondly, we defined an interesting and significant subsegment of CHR in which we can represent CHR states in an especially clear normal form. For this segment, we defined a sound and complete translation into coloured Petri nets.

Future Work We are confident that we can use our translation to provide useful contributions to the analysis and design of CHR programs. As an immediately follow-up work, we therefore plan to use it in order to apply standard analysis methods from Petri nets to CHR.
Furthermore, we will investigate how to extend our result to a larger segment of CHR. Two approaches seem feasible: On the one hand, using a more powerful variant of Petri nets could allow the direct translation of CHR programs into the respective net variant. On the other hand, it might be possible to develop non-injective mappings from CHR programs to less powerful Petri net variants that would nevertheless preserve certain properties.

Acknowledgements The author wishes to express his gratitude to Claus Reinke for fruitful discussions, helpful comments, as well as for pointing him into the right direction, and to the University of Ulm for funding him with LGFG grant #0518.

References

1. S. Abdennadher: *Operational Semantics and Confluence of Constraint Handling Rules.* Proceedings of the 3rd International Conference on Principles and Practice of Constraint Programming (CP 1997), Austria, October 1997.
2. K. Djelloul, M. Meister, J. Robin. *A Unified Semantics for Constraint Handling Rules in Transaction Logic.* Proceedings of LP-NMR'2007, Tempe, AZ, USA, 2007.
3. G. J. Duck, P. J. Stuckey, M. G. de la Banda, and C. Holzbaur: *The Refined Operational Semantics of Constraint Handling Rules.* Proceedings of the 20th International Conference on Logic Programming, 2004.
4. T. Frühwirth: *Constraint Handling Rules.* Constraint Programming: Basics and Trends, Springer LNCS 910, 1995.
5. T. Frühwirth: *Theory and Practice of Constraint Handling Rules*, Journal of Logic Programming, 37(1-3):95-138, 1998.
6. T. Frühwirth, A. Di Pierro, and H. Wiklicky: *Probabilistic Constraint Handling Rules*, 11th International Workshop on Functional and (Constraint) Logic Programming (WFLP 2002), 2002.
7. T. Frühwirth, S. Abdennadher: *Essentials of Constraint Programming*, Springer, 2003.
8. T. Frühwirth: *Parallelizing Union-Find in Constraint Handling Rules Using Confluence*, Proceedings of ICLP 2005, Springer, 2005.
9. M. Jantzen, R. Valk: *Formal properties of place/transition nets.* Springer LNCS 407, 1980.
10. K. Jensen: *Coloured Petri nets and the invariant-method.* Theoretical Computer Science, 14:317-336, 1989.
11. M. Meister: *Fine-grained Parallel Implementation of the Preflow-Push Algorithm in CHR.* Proceedings of WLP 2006, Vienna, 2006.
12. M. Meister: *Concurrency of the Preflow-Push Algorithm in Constraint Handling Rules.* Proceedings of CSCLP 2007, Roquencourt, 2007.
13. C.A. Petri: *Kommunikation mit Automaten.* Dissertation, TU Darmstadt, 1962.