Linear-Logic Based Analysis of Constraint Handling Rules with Disjunction

HARIOLF BETZ and THOM FRÜHWIRTH
University of Ulm

Constraint Handling Rules (CHR) is a declarative rule-based programming language that has cut out its niche over the course of the last 20 years. It generalizes concurrent constraint logic programming to multiple heads, thus closing the gap to multiset transformation systems. Its popular extension CHR with Disjunction ($CHR^\vee$) is a multi-paradigm declarative programming language that allows the embedding of Horn programs with SLD resolution.

We analyse the assets and the limitations of the classical declarative semantics of $CHR^\vee$ and highlight its natural relationship with linear logic. We furthermore develop two linear logic semantics for $CHR^\vee$ that differ in the reasoning domain for which they are instrumental. We show their idempotence and their soundness and completeness with respect to the operational semantics. We show how to apply the linear-logic semantics to decide program properties and to reason about operational equivalence of $CHR^\vee$ programs.

Categories and Subject Descriptors: F.3.1 [**Theory of Computation**]: Logics and Meanings of Programs—*Specifying and Verifying and Reasoning about Programs*; F.3.2 [**Theory of Computation**]: Logics and Meanings of Programs—*Semantics of Programming Languages*

General Terms: Languages, Theory, Verification

Additional Key Words and Phrases: Constraint Handling Rules, Linear Logic, Declarative Semantics

1. INTRODUCTION

A declarative semantics is a highly desirable property for a programming language. It offers a clean theoretical foundation for the language, allows to prove program properties such as correctness and operational equivalence and guarantees platform independence. Declarative programs tend to be clearer and more concise as they contain, ideally, only information about the modeled problem and not about control.

Constraint Handling Rules (CHR) [Frühwirth 1994; 1998; 2009] is a declarative committed-choice general-purpose programming language developed in the 1990s as a portable language extension to implement user-defined constraint solvers. Operationally, it mixes rule-based multiset rewriting over constraints with calls to a built-in constraint solver. By definition, the built-in solver implements at least a theory of equality. In practice, it is often a powerful constraint handler for a broad range of constraint domains. Irrespective of the built-in solver, CHR is Turing complete and it has been shown that un-

der some reasonable assumption about the buit-in constraint solver, every algorithm can be implemented in CHR with optimal time complexity [Sneyers et al. 2009]. Hence, it makes an efficient stand-alone general-purpose programming language.

Constraint Handling Rules with Disjunction ($\text{CHR}^\vee$) [Abdennadher and Schütz 1998] extends the inherently comitted-choice formalism of CHR with the possibility to include don't-know non-determinism and thus to embed Horn programs. We can justly describe it as a multi-paradigm declarative programming language. $\text{CHR}^\vee$ is a popular extension as CHR is usually implemented in Prolog, where don't-know non-determinism comes practically for free.

Owing to their heritage in logic programming and constraint logic programming, CHR and $\text{CHR}^\vee$ feature a declarative semantics in classical logic. We have shown, however, that certain classes of useful CHR programs have a logically inconsistent reading in this declarative semantics [Betz and Frühwirth 2005]. For others it is at least misleading. Operationally, CHR is a state transition system whereas the classical declarative semantics considers all states in a derivation as logically equivalent. Hence, the directionality of the rules, the inherent non-determinism of their execution and any change of state eludes this declarative semantics. While the don't-know non-determinism of $\text{CHR}^\vee$ in particular is expressed faithfully in the classical declarative semantics, the limitations observed in CHR carry over to $\text{CHR}^\vee$ as well. An alternative declarative semantics seems therefore desirable.

Linear logic is a sub-structural logical formalism [Girard 1987] that has been shown to bear a close relationship to concurrent committed-choice systems [Miller 1992; Fages et al. 2001]. We will see that it is well-suited to model the committed-choice rules of CHR. It furthermore allows a faithful embedding of classical logic, so we can straightforwardly embed the constraint theory underlying the built-in constraint solver into linear logic. Linear logic thus enables us to model the two reasoning mechanisms of CHR in a single formalism. Moreover, we can encode $\text{CHR}^\vee$ into linear logic in a way that preserves its characteristic dichotomy of don't-know and don't-care non-determinism.

In this article, we propose a linear-logic semantics for $\text{CHR}^\vee$ that incorporates all the features mentioned above. We found the semantics on the intuitionistic segment of linear logic as it suffices for our purpose while being easier to handle than the full segment. We propose two variants of the semantics. The first variant is based on introducing proper axioms in the sequent calculus of linear logic. The second variant is similar to the semantics previously published in Betz [2004], Betz and Frühwirth [2005] and Betz [2007]. The first formulation allows for considerably more elegant proofs, in particular of its soundness and completeness. The second formulation allows to perform a broader range of reasoning tasks. As we formalize and prove the idempotence of both representations, we can use either representation according to the respective application. We furthermore investigate the conditions under which we can apply our semantics to reason about $\text{CHR}^\vee$ programs. We identify a segment of $\text{CHR}^\vee$ where precise reasoning is possible. This segment includes pure CHR.

This article is structured as follows: In Sect. 2, we recall the syntax and operational semantics of $\text{CHR}^\vee$. In Sect. 3, we introduce the intuitionistic segment of linear logic. In Sect. 4, we develop two linear-logic semantics for $\text{CHR}^\vee$, we show their idempotence and their soundness and completeness with respect to the operational semantics. In Sect. 5, we discuss the conditions under which we can apply the linear-logic semantics to reason about

CHR$^\vee$ programs. In Sect. 6, we discuss related work before we conclude in Sect. 7.

2. CONSTRAINT HANDLING RULES WITH DISJUNCTION

In this section, we recall the syntax of Constraint Handling Rules with Disjunction (CHR$^\vee$) and its operational semantics $\omega_e^\vee$.

2.1 The Syntax of CHR$^\vee$

A *CHR system* is a tuple $\langle \Pi_u, \Pi_b, \mathcal{F}, \mathcal{V} \rangle$, where $\mathcal{V}$ is a set of variables, $\mathcal{F}$ is a set of function symbols and Π_u, Π_b are two sets of predicate symbols called the *user-defined* and *built-in* constraint symbols. From variables and function symbols we build *terms* in the usual manner.

An *atomic constraint* is a predicate of first-order logic. A *constraint* is a conjunction of first-order predicates. In CHR, we distinguish two disjoint classes of atomic constraints: *atomic built-in constraints* and *atomic user-defined constraints*, distinguished by the two sets of constraint symbols. *Built-in constraints* and *user-defined constraints* are possibly empty conjunctions of their respective atomic constraints. A formula built over atomic constraints using $\wedge$ and $\vee$ is called a *goal*[1].

The syntax of constraints is summarized in Def. 2.1:

Definition 2.1 *Constraint Syntax*. Let $c_b(\bar{t}), c_u(\bar{t})$ denote an n-ary atomic built-in or user-defined constraint, respectively, where $\bar{t}$ is an n-ary sequence of terms:

Built-in constraint:	$\mathbb{B} ::= \top \mid c_b(\bar{t}) \mid \mathbb{B} \wedge \mathbb{B}'$
User-defined constraint:	$\mathbb{U} ::= \top \mid c_u(\bar{t}) \mid \mathbb{U} \wedge \mathbb{U}'$
Goal:	$\mathbb{G} ::= \top \mid c_u(\bar{t}) \mid c_b(\bar{t}) \mid \mathbb{G} \wedge \mathbb{G}' \mid \mathbb{G} \vee \mathbb{G}'$

$\top$ stands for the *empty constraint* or the *empty goal*, respectively. The set of built-in constraints furthermore contains at least *falsity* $\bot$, and *syntactic equality* $\doteq$.

For any two goals $\mathbb{G}, \mathbb{G}'$, goal equivalence $\mathbb{G} \equiv_g \mathbb{G}'$ denotes equivalence between goals with respect to *commutativity* and *associativity* of $\wedge$, the *neutrality* of $\top$ with respect to $\wedge$, and the *distributivity* of $\wedge$ over $\vee$. A goal which does not contain disjunctions is called *flat*. The set of variables occurring in a goal $\mathbb{G}$ is denoted as $vars(\mathbb{G})$.

The goal equivalence relation $\equiv_g$ does not account for idempotence of $\wedge$ and $\vee$. It thus enforces a multiset semantics for conjunctions and disjunctions. For example, $c_u(\bar{t}) \wedge c_u(\bar{t}) \not\equiv_g c_u(\bar{t})$. (We observe that goal equivalence is thus stronger than logical equivalence.) Both built-in and user-defined constraints are special cases of flat goals.

Allowing $\wedge$ to distribute over $\vee$ guarantees that every goal is equivalent to its disjunctive normal form (DNF). The opposite law of distributivity is not allowed. For example, we have $\mathbb{G}_1 \wedge (\mathbb{G}_2 \vee \mathbb{G}_3) \equiv_G (\mathbb{G}_1 \wedge \mathbb{G}_2) \vee (\mathbb{G}_1 \wedge \mathbb{G}_3)$ but $\mathbb{G}_1 \vee (\mathbb{G}_2 \wedge \mathbb{G}_3) \not\equiv_G (\mathbb{G}_1 \vee \mathbb{G}_2) \wedge (\mathbb{G}_1 \vee \mathbb{G}_3)$. Thus any finite goal has only a finite number of equivalent representations. Alluding to its operational meaning, we also refer to $\vee$ as the *split operator*.

A CHR program is a set of rules adhering to the following definition:

[1]The term *goal* is used in CHR for historical reasons and does not imply that program execution is understood as proof search. The use of the terms *head* and *body* is closer to their use in production rule systems than to their use in logic programming.

Definition 2.2 *Rule Syntax.* (1) A CHR rule is of the form

$$r \ @ \ H_1 \setminus H_2 \Leftrightarrow G \mid B$$

The *rule head* $H_1 \setminus H_2$ consists of the *kept head* H_1 and the *removed head* H_2. Both H_1, H_2 are user-defined constraints. At least one of them must be non-empty. The guard G is a built-in constraint. The rule body B is a goal, whereas r serves as an identifier for the rule.

(2) The identifier r can be omitted along with the @. An empty guard $G = \top$ can be omitted along with the |. A rule with an empty kept head H_1 can be written as $r \ @ \ H_2 \Leftrightarrow G \mid B$. Such a rule is called a *simplification rule*. A rule where the removed head H_2 is empty can be written as $r \ @ \ H_1 \Rightarrow G \mid B$. Such a rule is called a *propagation rule*. A rule where neither H_1 nor H_2 is empty is called a *simpagation rule*.

(3) A *variant* of a rule $r \ @ \ H_1 \setminus H_2 \Leftrightarrow G \mid B$ with variables $\bar{x}$ is of the form $(r \ @ \ H_1 \setminus H_2 \Leftrightarrow G \mid B)[\bar{x}/\bar{y}]$ where $\bar{y}$ is an arbitrary sequence of pairwise distinct variables and $[\bar{x}/\bar{y}]$ denotes substitution of the variables in $\bar{x}$ with those in $\bar{y}$.

(4) A *CHR$^\vee$ program* is a set of CHR$^\vee$ rules.

While the rule identifier is operationally ignored, we need it to discuss CHR programs and use it in the definion of the operational semantics.

Example 2.3. The following rules are samples from programs discussed in the following section. The predicate symbols **leq** and **min** are user-defined constraint symbols, the infix symbol $\leq$ is a built-in constraint symbol. By definition, $\top$ and $\doteq$ are also built-in constraint symbols.

$$\begin{array}{lllll} r_M & @ & \mathbf{min}(x) \setminus \mathbf{min}(y) & \Leftrightarrow & x \leq y \mid \top \\ r_S & @ & \mathbf{leq}(x,y) \wedge \mathbf{leq}(y,x) & \Leftrightarrow & x \doteq y \\ r_T & @ & \mathbf{leq}(x,y) \wedge \mathbf{leq}(y,z) & \Rightarrow & \mathbf{leq}(x,z) \\ b_1 & @ & \mathbf{bird} & \Leftrightarrow & \mathbf{albatross} \vee \mathbf{penguin} \end{array}$$

Rule r_M is a simpagation rule with an atomic user-defined constraint in the kept head and one in the removed head. The guard is a built-in constraint and the body is empty. Rule r_S has an empty kept head, which makes is a simplification rule. Rule r_T is a propagation rule, since its removed head is empty. The bodies of the two rules hold a built-in constraint and a CHR constraint, respectively. Rule b_1 is a simplification rule with a disjunction / split operator in the body.

In anticipation of Section 2.2, we point out that in a so-called naïve or "very abstract" [Frühwirth 2009] operational semantics, propagation rules cause trivial non-termination of programs because they do not eliminate the pre-conditions of their firings. Hence, precautions have to be taken.

The most common strategy to avoid trivial non-termination consists in keeping a history of propagation rule firings and preventing redundant rule firings. We refer the reader to Abdennadher [1997] and Duck et al. [2004] for a discussion of this approach. A more recent approach [Betz et al. 2010] is based on finite representations of infinite program states and computations.

While operational semantics based on propagation histories prevail in practice, only naïve approaches are natural models of linear logic. Furthermore, they are the most general formulations: This means that (to our knowledge) every operational semantics proposed to this day is sound with respect to a naïve semantics while termination is usually gained at

the cost of completeness. This makes the naïve approach an adequate low-level abstraction over concrete CHR implementations to found our semantics upon.

We will therefore found our semantics on top of the semantics $\omega_e^\vee$, a naïve operational semantics for CHR$^\vee$ that we propose in the following section. Applicability of our results to other operational semantics is recovered by consideration of their soundness and completeness with respect to $\omega_e^\vee$. For example, standard results guarantee full applicability for the popular semantics ω_r [Duck et al. 2004] at least for the segment of confluent (cf. Definition 2.21) programs without propagation rules. The majority of example programs in this paper falls into this category. For a detailled discussion of the relationship between the various operational semantics, we refer the reader to Frühwirth [2009].

2.2 The Equivalence-Based Semantics $\omega_e^\vee$

In this section, we recall the operational semantics of $CHR^\vee$. The operational semantics we present here is a straightforward extension of the *equivalence-based semantics* ω_e [Raiser et al. 2009] for pure CHR (without disjunction). We denote it as $\omega_e^\vee$.

Operationally, built-in and user-defined constraints are handled separately. For the handling of built-in constraints, CHR requires a so-called *predefined constraint handler* or *built-in handler* whereas user-defined constraints are handled by the actual user program. We assume that the predefined solver implements a *decidable* first-order *constraint theory* CT over the built-in constraints.

While decidability may sound like a harsh requirement, it serves as a necessary device that allows us to identify logical judgements with judgements made by the built-in solver and disregard solver limitations. If practice, we can assume that CT is the theory that a specific built-in solver actually decides. For example, it is well-known that the theory of natural numbers is not decidable. Rather than considering the built-in solver an incomplete solver over an undecidable theory, we assume that it is a complete solver over a decidable subset of this theory. We then call this subset CT.

For automated proof search over CHR$^\vee$, we get a correct implementation of this CT for free if the automated prover is founded on the same built-in solver as the CHR$^\vee$ implementation over which we want to reason.

Definition 2.4 *Constraint Theory*. (1) A *coherent theory* as a set of axioms of the form

$$\alpha ::= \forall(\exists \bar{x}.\mathbb{G} \rightarrow \exists \bar{x}'.\mathbb{G}')$$

where $\mathbb{G}, \mathbb{G}'$ are possibly empty flat goals and x, x' are possible empty sequences of variables.

(2) A *constraint theory* CT for a CHR system is a decidable coherent theory over built-in constraints.

CHR program states are defined as follows:

Definition 2.5 *State*. (1) A state is a tuple of the form $S = \langle \mathbb{G}; \mathbb{V} \rangle$ where $\mathbb{G}$ is a goal called the *store* and $\mathbb{V}$ is a set of variables called *global variables*.

(2) A state $\langle \mathbb{G}; \mathbb{V} \rangle$ where $\mathbb{G}$ is flat is also called *flat*.

(3) For a flat state $S = \langle \mathbb{U} \wedge \mathbb{B}; \mathbb{V} \rangle$, where $\mathbb{U}$ is a user-defined constraint and $\mathbb{B}$ is a built-in constraint we call $\mathbb{U}$ the user-defined store and $\mathbb{B}$ the built-in store.

(4) For a flat state $S = \langle \mathbb{U} \wedge \mathbb{B}; \mathbb{V} \rangle$, we call
   (a) $\bar{l}_S ::= (vars(\mathbb{U}) \cup vars(\mathbb{B})) \setminus \mathbb{V}$ the *local variables* of S and

(b) $\bar{s}_S ::= \bar{l}_S \setminus vars(\mathbb{U})$ the *strictly local variables* of S.

(5) A *variant* of a flat state $S = \langle \mathbb{G}; \mathbb{V} \rangle$ with local variables $\bar{l}_S$ is a state S' of the form $S' = \langle \mathbb{G}[\bar{l}/\bar{x}]; \mathbb{V} \rangle$, where $\bar{x}$ is a sequence of pairwise distinct variables that do not occur in $\mathbb{V}$.

The following definition gives an equivalence relation over states:

Definition 2.6 *Equivalence of States*. In the following, let $\mathbb{U}, \mathbb{U}'$ denote arbitrary user-defined constraints, $\mathbb{B}, \mathbb{B}'$ built-in constraints, $\mathbb{G}, \mathbb{G}'$ goals, $\mathbb{V}, \mathbb{V}'$ sets of variables. Let x denote a variable and t denote a term. Equivalence of states, written as $\cdot \equiv_e \cdot$, is the smallest equivalence relation over states that satisfies all of the following conditions:

(1) *(Goal Transformation)*

$$\mathbb{G} \equiv_g \mathbb{G}' \quad \Rightarrow \quad \langle \mathbb{G}; \mathbb{V} \rangle \equiv_e \langle \mathbb{G}'; \mathbb{V} \rangle$$

(2) *(Equality as Substitution)*

$$\langle \mathbb{U} \wedge x \doteq t \wedge \mathbb{B}; \mathbb{V} \rangle \equiv_e \langle \mathbb{U}\,[x/t] \wedge x \doteq t \wedge \mathbb{B}; \mathbb{V} \rangle$$

(3) *(Application of CT)* Let $\langle \mathbb{U} \wedge \mathbb{B}; \mathbb{V} \rangle, \langle \mathbb{U} \wedge \mathbb{B}'; \mathbb{V} \rangle$ be flat states with strictly local variables $\bar{s}, \bar{s}'$. If $CT \models \exists \bar{s}.\mathbb{B} \leftrightarrow \exists \bar{s}'.\mathbb{B}'$ then:

$$\langle \mathbb{U} \wedge \mathbb{B}; \mathbb{V} \rangle \equiv_e \langle \mathbb{U} \wedge \mathbb{B}'; \mathbb{V} \rangle$$

(4) *(Neutrality of Redundant Global Variables)*

$$x \notin vars(\mathbb{G}) \quad \Rightarrow \quad \langle \mathbb{G}; \{x\} \cup \mathbb{V} \rangle \equiv_e \langle \mathbb{G}; \mathbb{V} \rangle$$

(5) *(Equivalence of Failed States)* For all goals $\mathbb{G}, \mathbb{G}'$ and all sets of variables $\mathbb{V}, \mathbb{V}'$:

$$\langle \mathbb{G} \wedge \bot; \mathbb{V} \rangle \equiv_e \langle \mathbb{G}' \wedge \bot; \mathbb{V}' \rangle$$

Where there is no ambiguity, we often write $\equiv$ rather than $\equiv_e$.

While we generally impose a multiset semantics on constraints, Definition 2.6.3 implicitly restores the set semantics for built-in constraints within states. Note also, that some of the axioms given above only apply to flat states. This limitation will be amended with the equivalence relation on configurations given in Definition 2.14.

A state with an inconsistent store is called a *failed state* as formalized in the following definiton:

Definition 2.7 *Failed State*. A state $S \equiv \langle \bot; \emptyset \rangle$ is called a *failed state*. We use $S_\bot = \langle \bot; \emptyset \rangle$ as the default representative for the set of failed states.

The following lemma states two properties following from Def. 2.6 that have been presented and proven in Raiser et al. [2009]:

Lemma 2.8 Properties of State Equivalence. *The following properties hold in general for flat states:*

(1) (Renaming of Local Variables)

$$\langle \mathbb{G}; \mathbb{V} \rangle \equiv \langle \mathbb{G}\,[x/y]\,; \mathbb{V} \rangle$$

for $x \notin \mathbb{V}$ and $y \notin \mathbb{V}$ and y does not occur in $\mathbb{G}$.

(2) (Logical Equivalence) *If*

$$\langle \mathbb{U} \wedge \mathbb{B}; \mathbb{V} \rangle \equiv \langle \mathbb{U}' \wedge \mathbb{B}'; \mathbb{V}' \rangle$$

then $CT \models (\exists \bar{l}.\mathbb{U} \wedge \mathbb{B}) \leftrightarrow (\exists \bar{l}'.\mathbb{U}' \wedge \mathbb{B}')$*, where* $\bar{l}, \bar{l}'$ *are the local variables of* $\langle \mathbb{U} \wedge \mathbb{B}; \mathbb{V} \rangle, \langle \mathbb{U}' \wedge \mathbb{B}'; \mathbb{V}' \rangle$*, respectively.*

Lemma 2.8.1 allows us to assume without loss of generality that the local variables of any two specific flat states are renamed apart. Concerning Lemma 2.8.2, note that logical equivalence of $\exists \bar{l}.\mathbb{U} \wedge \mathbb{B}$ and $\exists \bar{l}'.\mathbb{U}' \wedge \mathbb{B}'$ is a *necessary* but not a *sufficient* condition for state equivalence. The linear logic semantics will enable us to formulate a similar condition that is both necessary *and* sufficient (cf. Sect. 4.2).

The task of deciding equivalence – and more so: non-equivalence – of states is not always trivial using the axiomatic definition. We quote Theorem 2.12 which gives a necessary, sufficient, and decidable criterion to decide equivalence of flat states. It uses the following notion of *matching*:

Definition 2.9 *Matching of Constraints*. (1) For an n-ary sequence of variables $\bar{x} = x_1, \ldots, x_n$ and an n-ary sequence of terms $\bar{t} = t_1, \ldots, t_n$, we write $\bar{x} \doteq \bar{t}$ as a shorthand for $x_1 \doteq t_1 \wedge \ldots \wedge x_n \doteq t_n$.

(2) For user-defined constraints $\mathbb{U}, \mathbb{U}'$, we define its *match* as:

$$match(\mathbb{U}, \mathbb{U}') ::= \left\{ \bar{x} \doteq \bar{t} \mid \mathbb{U}[\bar{x}/\bar{t}] \equiv_g \mathbb{U}'[\bar{x}/\bar{t}] \right\}$$

(3) A *reduced match* $rm(\mathbb{U}, \mathbb{U}')$ is a subset of $match(\mathbb{U}, \mathbb{U}')$ where for every $\bar{x} \doteq \bar{t} \in match(\mathbb{U}, \mathbb{U}')$ there is a $\bar{x}' \doteq \bar{t}' \in rm(\mathbb{U}, \mathbb{U}')$ such that $CT \models \bar{x} \doteq \bar{t} \rightarrow \bar{x}' \doteq \bar{t}'$.

(4) A *fully reduced match* $frm(\mathbb{U}, \mathbb{U}')$ is a reduced match of $\mathbb{U}, \mathbb{U}'$ such that no subset of $frm(\mathbb{U}, \mathbb{U}')$ is a reduced match.

(5) We furthermore define the *matching* $\mathbb{U} \doteq \mathbb{U}'$ as follows, where $frm(\mathbb{U}, \mathbb{U}')$ is a fully reduced match of $\mathbb{U}$,$\mathbb{U}$':

$$\mathbb{U} \doteq \mathbb{U}' ::= \bigvee_{(\bar{x} \doteq \bar{t}) \in frm(\mathbb{U}, \mathbb{U}')} \bar{x} \doteq \bar{t}$$

A match is similar to the common notion of unifier. A fully reduced match roughly corresponds to a most general unifier.

Example 2.10 *Matching*. Some examples shall illustrate Definition 2.9:

$$\begin{array}{ll}
(c(1) \doteq c(1)) & = \top \\
(c(x) \doteq c(x)) & = \top \\
(c(x) \doteq c(1)) & = (x \doteq 1) \\
(c(x) \doteq d(x)) & = \bot \\
(c(0) \doteq c(1)) & = \bot \\
(c(x) \wedge c(y) \doteq c(0) \wedge c(1)) & = (x \doteq 0 \wedge y \doteq 1) \vee (x \doteq 1 \wedge y \doteq 0)
\end{array}$$

Theoretical papers about CHR and CHR$^\vee$ usually assume that syntactic equality $\doteq$ is a symbol of the underlying logic. We make the handling of equality explicit here, so we can properly translate it to intutionistic linear logic in the following:

Definition 2.11. For a given CHR system, the equality theory ET is the smallest coherent theory such that for every goal $\mathbb{G}[x]$ parametric in a variable x, we have:

$$ET \models \mathbb{G}[x] \wedge x \doteq t \rightarrow \mathbb{G}[t]$$

The following theorem has been published and proven in [Raiser et al. 2009]. It extends the criterion $CT \models \exists \bar{s}.\mathbb{B} \leftrightarrow \exists \bar{s}'.\mathbb{B}'$ of Definition 2.6.3 with a matching $\mathbb{U} \doteq \mathbb{U}'$ to express goal transformation and substitution. We get neutrality of redundant global variables and equivalence of failed states for free as properties of first-order logic.

THEOREM 2.12 CRITERION FOR $\equiv_e$. *Consider CHR states* $S = \langle \mathbb{U} \wedge \mathbb{B}; \mathbb{V} \rangle, S' = \langle \mathbb{U}' \wedge \mathbb{B}'; \mathbb{V} \rangle$ *with local variables* $\bar{l}, \bar{l}'$ *that have been renamed apart. Then* $S \equiv_e S'$ if and only if*:*

$$ET, CT \models \forall(\mathbb{B} \rightarrow \exists \bar{l}'.((\mathbb{U} \doteq \mathbb{U}') \wedge \mathbb{B}')) \wedge \forall(\mathbb{B}' \rightarrow \exists \bar{l}.((\mathbb{U} \doteq \mathbb{U}') \wedge \mathbb{B}))$$

A configuration can be thought of as a possibly empty multiset of independent CHR$^\vee$ states, denoted as a disjunction.

Definition 2.13 *Configuration*. (1) A configuration $\bar{S}$ is either an empty configuration $\bar{S} = \varepsilon$ or of the form $\bar{S} = S_1 \vee \ldots \vee S_n$, where $S_1, \ldots, S_n$ are states.

(2) A configuration is called *flat* if it is either empty or of the form $\bar{S} = S_1 \vee \ldots \vee S_n$ where $S_1, \ldots, S_n$ are flat states.

(3) A configuration is called *singular* if it is either empty or of the form $\bar{S} = S$ where S is a state.

The following definition extends the notion of equivalence from flat states to configurations:

Definition 2.14 *Equivalence of Configurations*. Equivalence of configurations, denoted as $\equiv_\vee$, is the smallest equivalence relation over configurations satisfying all of the following properties:

(1) *Commutativity and Associativity:*

$$\bar{S} \vee \bar{T} \equiv_\vee \bar{T} \vee \bar{S} \quad \text{and} \quad (\bar{S} \vee \bar{T}) \vee \bar{U} \equiv_\vee \bar{S} \vee (\bar{T} \vee \bar{U})$$

(2) *State Equivalence*

$$S \equiv_e S' \quad \Rightarrow \quad S \vee \bar{T} \equiv_\vee S' \vee \bar{T}$$

(3) *Neutrality of Failed States:*

$$S_\perp \vee \bar{T} \equiv_\vee \bar{T}$$

(4) *Split:*

$$\langle \mathbb{G}_1 \vee \mathbb{G}_2; \mathbb{V} \rangle \vee \bar{T} \equiv_\vee \langle \mathbb{G}_1; \mathbb{V} \rangle \vee \langle \mathbb{G}_2; \mathbb{V} \rangle \vee \bar{T}$$

We denote equivalence classes of configurations by square brackets: $[S] ::= \{T \mid S \equiv_\vee T\}$.

From Definition 2.14 follows the following important property:

PROPERTY 2.15 NORMAL FORMS OF CONFIGURATIONS. *(1) Every configuration* $\bar{S}$ *has an equivalent representation* $\bar{S}_F \equiv \bar{S}$ *such that* $\bar{S}_F$ *is flat. We call* $\bar{S}_F$ *a* flat normal form *(FNF) of* $\bar{S}$.

(2) *Every configuration* $\bar{S}$ *has an equivalent representation* $\bar{S}_S \equiv \bar{S}$ *such that* $\bar{S}_S$ *is singular. We call* $\bar{S}_S$ *a* singular normal form *(SNF) of* $\bar{S}$.

The following example shall illustrate the two normal forms:

Example 2.16 *Normal Forms*. Consider the configurations $\bar{S}_F = \langle c(x) \wedge d(x); \emptyset \rangle \vee \langle c(x) \wedge d'(x); \emptyset \rangle$ and $\bar{S}_S = \langle c(x) \wedge (d(x) \vee d'(x)); \emptyset \rangle$. While the two configuration are equivalent, $\bar{S}_F$ is in FNF and $\bar{S}_S$ is in SNF.

The flat normal form of configurations is of special importance as it implicitly extends properties of flat states, such as variable renaming, to configurations in general. For example, we have $\langle c(x) \vee d(x); \emptyset \rangle \not\equiv_e \langle c(y) \vee d(y); \emptyset \rangle$, but $\langle c(x) \vee d(x); \emptyset \rangle \not\equiv_\vee \langle c(y) \vee d(y); \emptyset \rangle$.

We define the notion of *local variables* of CHR rules, which is necessary for the definition of the operational semantics:

Definition 2.17 *Local Variables in Rules*. For a CHR rule $r \ @ \ H_1 \setminus H_2 \Leftrightarrow G \mid B$, we call the set

$$\bar{y}_r = vars(B, G) \setminus vars(H_1, H_2)$$

the *local variables* of r.

The following definition presents the transition system of CHR$^\vee$.

Definition 2.18 *Transition System of* $\omega_e^\vee$. CHR is a state transition system over equivalence classes of configurations. It is defined by the following transition rule, where $(r \ @ \ H_1 \setminus H_2 \Leftrightarrow G \mid B)$ is a variant of a CHR rule whose local variables $\bar{y}_r$ are renamed apart from the variables $vars(\mathbb{G}, \mathbb{V})$:

$$\frac{r \ @ \ H_1 \setminus H_2 \Leftrightarrow G \mid B \qquad ET, CT \models \exists(G \wedge \mathbb{G})}{[\langle H_1 \wedge H_2 \wedge G \wedge \mathbb{G}; \mathbb{V} \rangle \vee \bar{T}] \mapsto^r [\langle H_1 \wedge G \wedge B \wedge \mathbb{G}; \mathbb{V} \rangle \vee \bar{T}]}$$

If the applied rule is obvious from the context or irrelevant, we write transition simply as $\mapsto$. We denote its reflexive-transitive closure as $\mapsto^*$.

The required disjointness of the local variables $\bar{y}_r$ from all variables occurring in the pre-transition state outside G enforces that fresh variables are introduced for the local variables of the rule.

2.3 Properties

When reasoning about programs, we usually refer to the following observables:

Definition 2.19 *Observables*. Let S be a CHR$^\vee$ state, $\mathbb{P}$ be a program, and CT be a constraint theory. We distinguish three sets of observables, where $\bar{\mathbb{B}}$ stands for a disjunction of built-in constraints:

Computable configuration:	$\mathcal{C}_{\mathbb{P},CT}(S) ::= \{[\bar{T}] \mid [S] \mapsto^* [\bar{T}]\}$
Answer:	$\mathcal{A}_{\mathbb{P},CT}(S) ::= \{[\bar{T}] \mid [S] \mapsto^* [\bar{T}] \not\mapsto\}$
Data-sufficient answer:	$\mathcal{S}_{\mathbb{P},CT}(S) ::= \{[\langle \bar{\mathbb{B}}; \mathbb{V} \rangle] \mid [S] \mapsto^* [\langle \bar{\mathbb{B}}; \mathbb{V} \rangle]\}$

For all three sets, if the respective constraint theory CT is clear from the context or not important, it may be omitted from the identifier of the respective set. The set $\mathcal{S}_{\mathbb{P},CT}(S) \setminus \{[S_\perp]\}$ is called *successful answers*.

As the transition system does not allow transitions from an empty user-defined store (nor from failed states), the data-sufficient answers $\mathcal{S}_{\mathbb{P},CT}(S)$ are a subset of the answers $\mathcal{A}_{\mathbb{P},CT}(S)$ of any state S. The following property follows directly:

Property 2.20 Hierarchy of Observables. *For any state S, program $\mathbb{P}$ and constraint theory CT, we have:*

$$\mathcal{S}_{\mathbb{P},CT}(S) \subseteq \mathcal{A}_{\mathbb{P},CT}(S) \subseteq C_{\mathbb{P},CT}(S)$$

Confluence is an important property of transition systems. We define it in the usual manner:

Definition 2.21 *Confluence*. A CHR$^\vee$ program $\mathbb{P}$ is called *confluent* if for arbitrary configurations $\bar{S}, \bar{T}, \bar{U}$ such that $[\bar{S}] \mapsto^* [\bar{T}]$ and $[\bar{S}] \mapsto^* [\bar{U}]$, there exists a configuration $\bar{V}$ such that $[\bar{T}] \mapsto^* [\bar{V}]$ and $[\bar{U}] \mapsto^* [\bar{V}]$.

Confluence restricts the number of possible answers to a query:

Property 2.22. *Let $\mathbb{P}$ be a confluent CHR program. Then for every CHR state S, we have $|\mathcal{S}_{\mathbb{P}}(S)| \in \{0, 1\}$ and $|\mathcal{A}_{\mathbb{P}}(S)| \in \{0, 1\}$, where $|\cdot|$ denotes cardinality.*

Proof sketch. *We assume that for a state S, configurations T, T', and some confluent program $\mathbb{P}$, we have $S \mapsto^* T \not\mapsto$ and $S \mapsto^* T' \not\mapsto$ and $[T] \neq [T']$. Applying Def. 2.21 leads to a contradiction.* □

A necessary, sufficent and decidable criterion for confluence has been given for *pure CHR* in Abdennadher et al. [1996]. It can straightforwardly be extended to CHR$^\vee$.

One of the most significant features of CHR$^\vee$ is that it naturally embeds Horn programs with SLD resolution. The following definition and theorem were published in [Abdennadher and Schütz 1998], though we adapted their notation.

Definition 2.23 *Horn Embedding*. Let p/n be an n-ary Horn predicate, defined by m clauses of the form

$$p(\bar{t}_i) \leftarrow G_i$$

Then its embedding into CHR$^\vee$ is defined as

$$p(\bar{x}) \Leftrightarrow \bigvee_{i=0}^{m} (\bar{x} \doteq \bar{t}_i) \wedge G_i$$

The embedding of a Horn program P is the set of the embeddings of its predicates. We denote this embedding as $P^\vee$.

The following theorem was presented in [Abdennadher and Schütz 1998]. A proof sketch can be found there. We adapted it to our terminology.

Theorem 2.24. *Let $\mathbb{H}$ be a Horn program, let $\mathbb{P}$ be its embedding in CHR$^\vee$ and let $p(\bar{t})$ be a predicate. For every succesful leaf $\langle \top; \theta \rangle$ in the SLD tree of $p(\bar{t})$, there is a state $\langle \mathbb{B}; vars(\bar{t}) \rangle$ in a computable configuration of $\langle p(\bar{t}); vars(\bar{t}) \rangle$ such that*

$$ET, CT \models \mathbb{B} \leftrightarrow (\bar{t} \doteq \bar{t}\theta)$$

and vice versa.

2.4 Examples

We shall illustrate our definitions with several examples. Example 2.25 presents a well-known example program:

Example 2.25. We consider the following program which implements a constraint solver for the partial-order relation. To emphasize that the partial-order relation is a user-defined constraint here, we denote it with the prefix symbol **leq**.

$$\mathbb{P}_{leq} = \left\{ \begin{array}{lcll} r_I & @ & \mathbf{leq}(x,y) \wedge \mathbf{leq}(x,y) & \Leftrightarrow \ \mathbf{leq}(x,y) \\ r_R & @ & \mathbf{leq}(x,x) & \Leftrightarrow \ \top \\ r_S & @ & \mathbf{leq}(x,y) \wedge \mathbf{leq}(y,x) & \Leftrightarrow \ x \doteq y \\ r_T & @ & \mathbf{leq}(x,y) \wedge \mathbf{leq}(y,z) & \Rightarrow \ \mathbf{leq}(x,z) \end{array} \right\}$$

The following is a sample derivation, starting from an initial state $S_0 = \langle a \leq b \wedge b \leq c \wedge c \leq a; \mathbb{V}\rangle$, where $\mathbb{V} = \{a, b, c\}$. According to the usual practice, all variables occurring in the initial state are global. Equivalence transformations are stated explicitly:

$$\begin{aligned} & \langle \mathbf{leq}(a,b) \wedge \mathbf{leq}(b,c) \wedge \mathbf{leq}(c,a); \mathbb{V}\rangle && (1) \\ \equiv\ & \langle \underline{\mathbf{leq}(x,y)} \wedge \underline{\mathbf{leq}(y,z)} \wedge \mathbf{leq}(c,a) \wedge x \doteq a \wedge y \doteq b \wedge z \doteq c; \mathbb{V}\rangle \\ \mapsto^{r_T}\ & \langle \underline{\mathbf{leq}(x,z)} \wedge \mathbf{leq}(x,y) \wedge \mathbf{leq}(y,z) \wedge \mathbf{leq}(c,a) \wedge x \doteq a \wedge y \doteq b \wedge z \doteq c; \mathbb{V}\rangle \\ \equiv\ & \langle \mathbf{leq}(a,c) \wedge \mathbf{leq}(a,b) \wedge \mathbf{leq}(b,c) \wedge \mathbf{leq}(c,a); \mathbb{V}\rangle && (2) \\ \equiv\ & \langle \underline{\mathbf{leq}(x,y)} \wedge \underline{\mathbf{leq}(y,x)} \wedge \mathbf{leq}(a,b) \wedge \mathbf{leq}(b,c) \wedge x \doteq a \wedge y \doteq c; \mathbb{V}\rangle \\ \mapsto^{r_S}\ & \langle \mathbf{leq}(a,b) \wedge \mathbf{leq}(b,c) \wedge \underline{x \doteq y} \wedge x \doteq a \wedge y \doteq c; \mathbb{V}\rangle \\ \equiv\ & \langle \mathbf{leq}(a,b) \wedge \mathbf{leq}(c,b) \wedge a \doteq c; \mathbb{V}\rangle && (3) \\ \equiv\ & \langle \underline{\mathbf{leq}(x,y)} \wedge \underline{\mathbf{leq}(y,x)} \wedge x \doteq a \wedge y \doteq b \wedge a = c; \mathbb{V}\rangle \\ \mapsto^{r_S}\ & \langle \underline{x \doteq y} \wedge x \doteq a \wedge y \doteq b \wedge a \doteq c; \{a,b,c\}\rangle \\ \equiv\ & \langle a \doteq b \wedge a \doteq c; \mathbb{V}\rangle \quad \not\mapsto && (4) \end{aligned}$$

Usually, we do not make equivalence transformations explicit and list only states where local variables are eliminated as far as possible such as the labeled states (1)-(4). The derivation is then reduced to:

$$\begin{aligned} & \langle \underline{\mathbf{leq}(a,b)} \wedge \underline{\mathbf{leq}(b,c)} \wedge \mathbf{leq}(c,a); \mathbb{V}\rangle && (1) \\ \mapsto^{r_T}\ & \langle \underline{\mathbf{leq}(a,c)} \wedge \mathbf{leq}(a,b) \wedge \mathbf{leq}(b,c) \wedge \underline{\mathbf{leq}(c,a)}; \mathbb{V}\rangle && (2) \\ \mapsto^{r_S}\ & \langle \underline{\mathbf{leq}(a,b)} \wedge \underline{\mathbf{leq}(b,c)} \wedge a \doteq c; \mathbb{V}\rangle && (3) \\ \mapsto^{r_S}\ & \langle a \doteq b \wedge a \doteq c; \mathbb{V}\rangle \quad \not\mapsto && (4) \end{aligned}$$

With respect to our observables, we have:

$$\mathcal{S}_{\mathbb{P},CT}(S_0) = \mathcal{A}_{\mathbb{P},CT}(S_0) = \{[\langle \top; a \doteq b \wedge a \doteq c; \{a,b,c\}\rangle]\}$$

The set $C_{\mathbb{P},CT}(S_0)$ is infinite as the operational semantics ω_e allows potentially unlimited applications of r_T.

While a built-in constraint in the guard inhibits rule application when it is not implied by the program store, a built-in constraint in the rule body can lead to a contradiction in the store and thus cause a computation to fail. Consider the following computation:

$$\begin{aligned} & \langle \underline{\mathbf{leq}(0,1)} \wedge \underline{\mathbf{leq}(1,0)}; \mathbb{V}\rangle && (1) \\ \mapsto^{r_S}\ & \langle \underline{0 \doteq 1}; \mathbb{V}\rangle \equiv_{\vee} \langle \bot; \mathbb{V}\rangle && (2) \end{aligned}$$

Example 2.26. The following program computes the minimum of a set, provided the initial state contains a set of candidates wrapped in unary constraints **min**. Note that in contrast to Example 2.25, we now use the the partial-order relation as a built-in constraint. To emphasize the difference, we use the infix constraint symbol $\leq$ rather than the prefix symbol **leq** in this example:

$$\mathbb{P}_{min} = \left\{ r_M \;@\; \mathbf{min}(x) \setminus \mathbf{min}(y) \;\Leftrightarrow\; x \leq y \mid \top \right\}$$

The following is a sample derivation, starting from an initial state $S_0 = \langle \mathbf{min}(1), \mathbf{min}(2), \mathbf{min}(3), \mathbf{min}(4); \emptyset \rangle$.

$$\begin{aligned}
& \langle \underline{\mathbf{min}(1)} \wedge \mathbf{min}(2) \wedge \mathbf{min}(3) \wedge \underline{\mathbf{min}(4)}; \emptyset \rangle && (1)\\
\mapsto^{r_M} & \langle \mathbf{min}(1) \wedge \underline{\mathbf{min}(2)} \wedge \underline{\mathbf{min}(3)}; \emptyset \rangle && (2)\\
\mapsto^{r_M} & \langle \underline{\mathbf{min}(1)} \wedge \underline{\mathbf{min}(2)}; \emptyset \rangle && (3)\\
\mapsto^{r_M} & \langle \mathbf{min}(1); \emptyset \rangle \quad \not\mapsto && (4)
\end{aligned}$$

With respect to our observables, we have $\mathcal{S}_{\mathbb{P},CT}(S_0) = \emptyset$ and $\mathcal{A}_{\mathbb{P},CT}(S_0) = \{[\langle \mathbf{min}(1); \emptyset \rangle]\}$. The set $C_{\mathbb{P},CT}(S_0)$ contains every configuration that can be reached on the non-deterministic path from S_0 to $\langle \mathbf{min}(1); \emptyset \rangle$. Its cardinality is 8.

A guarded rule only fires, if the guard is implied by the store. Consider the following computation:

$$\begin{aligned}
& \langle \underline{\mathbf{min}(a)} \wedge \underline{\mathbf{min}(b)} \wedge \mathbf{min}(c) \wedge \underline{a \leq b}; \{a, b, c\} \rangle && (1)\\
\mapsto^{r_M} & \langle \mathbf{min}(a) \wedge \mathbf{min}(c); \{a, c\} \rangle \quad \not\mapsto && (2)
\end{aligned}$$

Example 2.27. The following program is a minimal example of a knowledge base implemented in CHR$^\vee$:

$$\mathbb{P}_{birds} = \left\{ \begin{array}{lll} b_1 \;@\; \mathbf{bird} & \Leftrightarrow & \mathbf{albatross} \vee \mathbf{penguin} \\ b_2 \;@\; \mathbf{penguin} \wedge \mathbf{flies} & \Leftrightarrow & \bot \end{array} \right\}$$

Using $\omega_e^\vee$, we can construct the following derivation starting from the initial state $S_0 = \langle \mathbf{bird} \wedge \mathbf{flies}; \emptyset \rangle$:

$$\begin{aligned}
& [\langle \mathbf{bird} \wedge \mathbf{flies}; \emptyset \rangle] \\
\mapsto_\vee\ & [\langle (\mathbf{albatross} \vee \mathbf{penguin}) \wedge \mathbf{flies}; \emptyset \rangle] \\
=\ & [\langle \mathbf{albatross} \wedge \mathbf{flies}; \emptyset \rangle \vee \langle \mathbf{penguin} \wedge \mathbf{flies}; \emptyset \rangle] \\
\mapsto_\vee\ & [\langle \mathbf{albatross} \wedge \mathbf{flies}; \emptyset \rangle \vee \langle \bot; \emptyset \rangle] \\
=\ & [\langle \mathbf{albatross} \wedge \mathbf{flies}; \emptyset \rangle]
\end{aligned}$$

With respect to the observables, we have $C_{\mathbb{P}}(S_0) = \{[S_0], [\langle (\mathbf{albatross} \vee \mathbf{penguin}) \wedge \mathbf{flies}; \emptyset \rangle], [\langle \mathbf{albatross} \wedge \mathbf{flies} \rangle]\}$, $\mathcal{A}_{\mathbb{P}}(S_0) = \{[\langle \mathbf{albatross} \wedge \mathbf{flies} \rangle]\}$, and $\mathcal{S}_{\mathbb{P}}(S_0) = \emptyset$.

3. INTUITIONISTIC LINEAR LOGIC

Linear logic is a substructural logical formalism introduced by Girard [1987]. Unlike classical logic, linear logic does not allow free copying or discarding of assumptions. It furthermore features a fine distinction between internal and external choice and a sound and complete embedding of classical and intuitionistic logic. In this section, we recall the

intuitionistic fragment of linear logic. It allows for a straightforward, faithful embedding of intuitionistic logic.

Note that in contrast to the classical case, intuitionistic linear logic is distinguished from the full fragment of linear logic by the absence of several logical connectives. Hence, the choice of the intuitionistic fragment comes naturally as the connectives in this fragment appear most suitable to model $\text{CHR}^\vee$.

3.1 Definition

We will give the formal definition in terms of a *sequent calculus*. The calculus is based on binary sequents of the form

$$\Gamma \vdash \alpha$$

where Γ is a multiset of formulas (written without braces) called *antecedent* and α is a formula called *consequent*. A sequent $\Gamma \vdash \alpha$ represents the fact that assuming the formulas in Γ, we can conclude α. A sequent is a formalization of logical judgement. We will therefore call $\vdash$ the *judgement* relation.

The sequent calculus is given as a set of *inference rules*. An inference rule is composed of zero, one, or several premises and exactly one conclusion. Inference rules without premises introduce *axioms* of the system. Most inference rules are dedicated to the introduction of a symbol or connective of the logic and named accordingly. Those that do no introduce a specific symbol are called *structural rules*.

A *proof tree* – or simply: *proof* – is a finite labeled tree whose nodes are labeled with sequents such that every sequent node is the consequence of its direct children according to one of the inference rules of the calculus. A proof tree is called *complete* if all its leaves are axioms. We call a sequent $\Gamma \vdash \alpha$ *valid* if there exists a complete proof tree π with $\Gamma \vdash \alpha$ at the root.

The following two structural rules are common to many logical systems. They establish reflexivity and a form of transitivity of the judgement relation.

$$\frac{}{\alpha \vdash \alpha}\ (\mathit{Identity}) \qquad \frac{\Gamma \vdash \alpha \quad \alpha, \Delta \vdash \beta}{\Gamma, \Delta \vdash \beta}\ (\mathit{Cut})$$

The tokens of (intuitionistic) linear logic are commonly considered as representing *resources* rather than *truths*. This terminology reflects the fact that assumptions may not be copied nor discarded freely in linear logic, but must be used exactly once. From a different point of view, we might say that linear logic *consumes* assumptions in judgements and is aware of their *multiplicities*.

Multiplicative conjunction is distinguished from classical or intuitionistic conjunction as it lacks idempotence. The conjunction $\alpha \otimes \beta$ represents *exactly* one instance of α and one instance of β. The atomic formula α represents exactly one instance of α, the conjunction $\alpha \otimes \alpha$ exactly two instances. Hence, α is not equivalent to $\alpha \otimes \alpha$. Multiplicative conjunction is introduced by the following inference rules:

$$\frac{\Gamma, \alpha, \beta \vdash \gamma}{\Gamma, \alpha \otimes \beta \vdash \gamma}\ (L \otimes) \qquad \frac{\Gamma \vdash \alpha \quad \Delta \vdash \beta}{\Gamma, \Delta \vdash \alpha \otimes \beta}\ (R \otimes)$$

For a complete picture, note that the term *multiplicative* refers to the fact that the two premises of the right-hand introduction rule $(R \otimes)$ have distinct antecedents Γ, Δ. This in

$$\dfrac{\dfrac{\dfrac{}{e \vdash e}\ (Identity) \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{c \vdash c}\ (Identity)}{!c \vdash c}\ (Dereliction) \quad \dfrac{\dfrac{}{c \vdash c}\ (Identity)}{!c \vdash c}\ (Dereliction)}{!c,\ !c \vdash c \otimes c}\ (R\otimes)}{!c \vdash c \otimes c}\ (Contraction)}{e \multimap !c, e \vdash c \otimes c}\ (L\multimap)}{e \multimap !c \vdash e \multimap c \otimes c}\ (R\multimap)}{!(e \multimap !c) \vdash e \multimap c \otimes c}\ (Dereliction)$$

Fig. 1. A sample proof tree

in contrast to the *additive* connectives introduced below, where the premises share the same context Γ. The constant $\mathbf{1}$ represents the empty resource and is consequently the neutral element with respect to multiplicative conjunction.

$$\frac{\Gamma \vdash \alpha}{\Gamma, \mathbf{1} \vdash \alpha}\ (L\mathbf{1}) \qquad \frac{}{\vdash \mathbf{1}}\ (R\mathbf{1})$$

Linear implication $\multimap$ allows the application of *modus ponens* where the preconditions of a linear implication are consumed on application. For example, the sequent $\alpha \otimes (\alpha \multimap \beta) \vdash \beta$ is valid whereas $\alpha \otimes (\alpha \multimap \beta) \vdash \alpha \otimes \beta$ is not. The following inference rules introduce $\multimap$:

$$\frac{\Gamma \vdash \alpha \quad \beta, \Delta \vdash \gamma}{\Gamma, \alpha \multimap \beta, \Delta \vdash \gamma}\ (L\multimap) \qquad \frac{\Gamma, \alpha \vdash \beta}{\Gamma \vdash \alpha \multimap \beta}\ (R\multimap)$$

The *! ("bang") modality* marks stable facts or unlimited resources, thus recovering propositions in the classical (or intuitionistic) sense. Like a classical proposition, a *banged* resource may be freely copied or discarded. Hence, $!\alpha \otimes !(\alpha \multimap \beta) \vdash !\alpha \otimes !\beta$ is a valid sequent. Four inference rules introduce the bang: *Weakening* allows the insertion of additional (redundant) assumptions into a judgement. *Contraction* restores the set semantics for banged resources. *R!* affirms that a banged resource always infers its non-banged counterpart. *Dereliction* completes the recovery of classical and intuitionistic logic by allowing banged conclusions, provided that all assumptions are banged.

$$\frac{\Gamma, !\alpha, !\alpha \vdash \beta}{\Gamma, !\alpha \vdash \beta}\ (Contraction) \qquad \frac{\Gamma \vdash \beta}{\Gamma, !\alpha \vdash \beta}\ (Weakening)$$

$$\frac{!\Gamma \vdash \alpha}{!\Gamma \vdash !\alpha}\ (R!) \qquad \frac{\Gamma, \alpha \vdash \beta}{\Gamma, !\alpha \vdash \beta}\ (Dereliction)$$

Example 3.1. We can model the fact that one cup of coffee (c) costs one euro (e) as $!(e \multimap c)$. A "bottomless cup" is an offer including an unlimited number of refills. We assume that any natural number of refills is possible. We model this as $!(e \multimap !c)$. From this, we may judge that it is possible to get two cups of coffee for one euro: $!(e \multimap !c) \vdash e \multimap c \otimes c$. Fig. 3.1 gives an examplary proof tree, proving this judgement.

Internal choice means that we can freely choose one out of two options. In classical (and intuitionistic) logic, internal choice is an aspect of conjunction, as exemplified by the judgement $\alpha \wedge \beta \vdash \alpha$. This is inherited by the *additive conjunction* & of linear logic. The formula $\alpha \& \beta$ expresses an internal choice between α and β, i.e. both sequents $\alpha \& \beta \vdash \alpha$

and $\alpha\&\beta \vdash \beta$ are valid. Internal choice also means that we cannot choose both options at the same time. Consequently, $\alpha\&\beta \vdash A \otimes B$ is not valid.

$$\frac{\Gamma, \alpha \vdash \gamma}{\Gamma, \alpha\&\beta \vdash \gamma}\ (L\&_1) \qquad \frac{\Gamma, \beta \vdash \gamma}{\Gamma, \alpha\&\beta \vdash \gamma}\ (L\&_2) \qquad \frac{\Gamma \vdash \alpha \quad \Gamma \vdash \beta}{\Gamma \vdash \alpha\&\beta}\ (R\&)$$

The $\top$ ("top") is the resource that all other resources can be mapped to, i.e. for every α, the implication $\alpha \multimap \top$ is a tautology. It is hence the neutral element with respect to additive conjunction.

$$\frac{}{\Gamma \vdash \top}\ (R\top)$$

External choice means that one out of two alternatives will occur, but we cannot choose which. External choice is an aspect of classical (and intuitionistic) disjunction. In linear logic, it is represented by the *additive disjunction* $\oplus$. The disjunction $\alpha \oplus \beta$ means that we will get either α or β. Analogous to classical disjunction, we therefore cannot judge $\alpha \oplus \beta \vdash \alpha$. However, a formula that is a consequence of both α and β can be optained: $!(\alpha \multimap \gamma), !(\beta \multimap \gamma), \alpha \oplus \beta \vdash \gamma$ is valid.

$$\frac{\Gamma, \alpha \vdash \gamma \quad \Gamma, \beta \vdash \gamma}{\Gamma, \alpha \oplus \beta \vdash \gamma}\ (L\oplus) \qquad \frac{\Gamma \vdash \alpha}{\Gamma \vdash \alpha \oplus \beta}\ (R\oplus_1) \qquad \frac{\Gamma \vdash \beta}{\Gamma \vdash \alpha \oplus \beta}\ (R\oplus_2)$$

Analogous to falsity in the classical sense, *absurdity* $\mathbf{0}$ is a constant that yields every other resource. It is the neutral element with respect to $\oplus$.

$$\frac{}{\mathbf{0} \vdash \alpha}\ (L\mathbf{0})$$

Example 3.2. We assume that, besides coffee, the cafeteria offers also pie (p) at the price of one euro per piece: $!(e \multimap p)$. We infer that for one euro, we have the choice between an arbitrary amount of coffee and a piece of pie: $!(e \multimap !c), !(e \multimap p) \vdash e \multimap (!c\&p)$. Let us furthermore assume that rather than with euros, we can also pay with dollars (d) at a $1:1$ ratio: $!(d \multimap !c), !(d \multimap p)$. We may infer that either one of *one dollar* or *one euro* buys us a choice between an arbitrary amount of coffee and one pie:

$$!(e \multimap !c), !(e \multimap p), !(d \multimap !c), !(d \multimap p) \vdash (e \oplus d) \multimap (!c\&p).$$

We can extend intuitionistic linear logic into a first-order system with the quantifiers $\exists$ and $\forall$. Their introduction rules are the same as in classical logic. In the following rules, t stands for an arbitrary term whereas a stands for a variable that is not free in Γ, α or β:

$$\frac{\Gamma, \alpha[x/t] \vdash \beta}{\Gamma, \forall x.\alpha \vdash \beta}\ (L\forall) \qquad \frac{\Gamma \vdash \beta[x/a]}{\Gamma \vdash \forall x.\beta}\ (R\forall)$$

$$\frac{\Gamma, \alpha[x/a] \vdash \beta}{\Gamma, \exists x.\alpha \vdash \beta}\ (L\exists) \qquad \frac{\Gamma \vdash \beta[x/t]}{\Gamma \vdash \exists x.\beta}\ (R\exists)$$

3.2 Properties of Intuitionistic Linear Logic

The resulting first-order system allows for a sound and complete embedding of intuitionistic first order logic. This is widely considered one of the most important features of linear

logic. The following translation from intuitionistic logic into intuitionistic linear logic is a variant of a translation proposed by Negri [1995]:

Definition 3.3. $(\cdot)^*$ is a translation from formulas of intuitionistic logic to formulas of intuitionistic linear logic, recursively defined by the following rules:

$$\begin{aligned}
p(\bar{t})^* &::= !p(\bar{t}) \\
(\bot)^* &::= \mathbf{0} \\
(\top)^* &::= \mathbf{1} \\
(A \wedge B)^* &::= A^* \otimes B^* \\
(A \vee B)^* &::= A^* \oplus B^* \\
(A \rightarrow B)^* &::= !(A^* \multimap B^*) \\
(\forall x.A)^* &::= !\forall x.(A^*) \\
(\exists x.A)^* &::= \exists x.(A^*)
\end{aligned}$$

$p(\bar{t})$ stands for an atomic proposition. The definition is extended to sets and multisets of formulas in the obvious manner. It has been proven in Negri [1995] that an intuitionistic sequent ($\Gamma \vdash_{IL} \alpha$) is valid *if and only if* ($\Gamma^* \vdash_{ILL} \alpha^*$) is valid in intuitionistic linear logic.

We distinguish two sorts of axioms in the sequent calculus. The (*Identity*) axiom and the constant axioms ($L\mathbf{1}$), ($R\mathbf{1}$), ($L\mathbf{0}$) and ($R\top$) constitute the *logical axioms* of intuitionistic linear logic. All axioms we add to the system on top of these are called *non-logical axioms* or *proper axioms*. We usually use the letter Σ to denote the set of proper axioms.

We express the fact that a judgement $\Gamma \vdash \alpha$ is provable using a non-empty set Σ of proper axioms by indexing the judgement relation with the set of proper axioms: $\vdash_\Sigma$.

Definition 3.4 *Linear-Logic Equivalence*. (1) We call two linear-logic formulas α, β *logically equivalent* if both $\alpha \vdash \beta$ and $\beta \vdash \alpha$ are provable. We write this as $\alpha \dashv\vdash \beta$.

(2) For any set of proper axioms Σ, we call two linear-logic formulas α, β *logically equivalent modulo* Σ if both $\alpha \vdash_\Sigma \beta$ and $\beta \vdash_\Sigma \alpha$ are provable. We write this as $\alpha \dashv\vdash_\Sigma \beta$.

As a well-behaved logical system, linear logic features a cut-elimination theorem [Girard 1987]:

Theorem 3.5 Cut Elimination Theorem. *(1) If a sequent $\Gamma \vdash \alpha$ has a proof π that does not contain any proper axioms, then it has a proof π' that contains neither proper axioms nor the (Cut) rule.*

(2) If a sequent $\Gamma \vdash_\Sigma \alpha$ has a proof π containing proper axioms, then it has a proof π' where the (Cut) rule is only used at the leaves such that one of its premises is an axiom.

A proof without any applications of (*Cut*) is called *cut-free*. A proof where (*Cut*) is only applied at the leaves is called *cut-reduced*.

An important consequence of cut elminiation is the *subformula property*. We quote a weak formulation of the property, which will suffice for our purpose: Every formula α in a cut-free proof of a sequent $\Gamma \vdash \beta$ is a subformula of either Γ or β, modulo variable renaming. In a cut-reduced proof of a sequent $\Gamma \vdash_\Sigma \beta$, every formula α is a subformula of Γ or β, modulo variable renaming, or there exists a proper axiom $(\Delta \vdash \gamma) \in \Sigma$ such that α is a subformula of Δ or γ, modulo variable renaming.

Another important feature of linear logic is the so-called *phase semantics* [Girard 1987]. It is a powerful tool to decide provability in linear logic and its subsegments (such as intuitionistic linear logic). It has been applied successfully to reason in LCC by Fages et al. [1997] and to CHR by Haemmerlé and Betz [2008].

The phase semantics interprets linear logic a an algebraic structure. Formulas of linear logic are mapped to subsets of a monoid and linear-logic connectives are mapped to operations on those sets. The judgement relation $\vdash$ finally maps to the inclusion relation $\subseteq$ between sets. While the phase semantics is essential for effective applications of the results published here, recalling it in its entirety is beyond the scope of this paper. We hence refer the interested reader to Girard [1987] for the phase semantics for full linear logic and to Fages et al. [1997] for a concise presentation of the phase semantics for intuitionistic linear logic.

4. A LINEAR-LOGIC SEMANTICS FOR CHR

In this section, we develop the linear-logic semantics for Constraint Handling Rules. We firstly recall the classical declarative semantics in Sect. 4.1. Then we motivate and present a linear-logic semantics based on proper axioms in Sect. 4.2. We will henceforth call this the *axiomatic* linear-logic semantics for CHR. Its soundness with respect to the operational semantics is shown in Sect. 4.3. We continue by introducing the notion of *state entailment* and using it to formulate and prove the completeness of our semantics in Sect. 4.4. We show an alternative linear-logic semantics that encodes programs and contsraint theories into linear logic in Sect. 4.5. We discuss our semantics in Sect. 4.6.

4.1 Analysis of the Classical Declarative Semantics

CHR is founded on a classical declarative semantics, which is reflected in its very syntax. In this section, we recall the classical declarative semantics and discuss its assets and limitations.

In the following, $\exists_{-\bar{x}}$ stands for existential quantification of all variables except those in $\bar{x}$, where $\bar{x}$ is a set of variables. The classical declarative semantics is given in the following table, where $(\cdot)^\dagger$ stands for translation to classical logic and $\bar{y}_r$ denotes the local variables of the respective rule:

States:	$\langle \mathbb{G}; \mathbb{V} \rangle^\dagger$	$::=$	$\exists_{-\mathbb{V}}.(\mathbb{G})$
Rules:	$(r \mathbin{@} H_1 \setminus H_2 \Leftrightarrow G \mid B)^\dagger$	$::=$	$\forall (G \rightarrow (H_1 \rightarrow (H_2 \leftrightarrow \exists \bar{y}_r.B)))$
Programs:	$\{R_1, ..., R_m\}^\dagger$	$::=$	$\{R_1^\dagger, ..., R_m^\dagger\}$

The following lemma – cited from Frühwirth and Abdennadher [2003] – establishes the relationship between the logical readings of programs, constraint theories and states.

Lemma 4.1 (Logical Equivalence of States). *Let $\mathbb{P}$ be a CHR program, CT be a constraint theory, and S be a state. Then for all computable states T_1 and T_2 of S, the following holds: $\mathbb{P}^\dagger, CT, ET \models \forall(T_1^\dagger \leftrightarrow T_2^\dagger)$.*

The declarative semantics of CHR must be distinguished from LP languages in the narrower sense, i.e. declarative languages applying backward reasoning on Horn clauses. Unlike these, CHR is not founded on the notion of *execution as proof search*. Declaratively, execution of a CHR program means stepwise transformation of the information contained in the state under logical equivalence as defined by the program's logical reading $\mathbb{P}^\dagger$ and the constraint theory CT. Founding CHR on such a declarative semantics is an obvious choice for several reasons:

Firstly, the notion of *execution as proof search* naturally implies a notion of *search*. This stands in contrast to the committed-choice execution of CHR. Furthermore, the forward-reasoning approach faithfully captures the one-sided variable matching between rule heads

and constraints in CHR, as opposed to unification. For example, a CHR state $\langle p(x); \top; \emptyset \rangle$ (where x is a variable) does not match with the rule head $(p(a) \Leftrightarrow \ldots)$ (where a is a constant) just as we cannot apply modus ponens on a fact $\exists x.p(x)$ and an implication $(p(a) \rightarrow \ldots)$. In contrast, an LP goal $p(x)$ would be unified with a rule head $(p(a) \leftarrow \ldots)$, accounting for the fact that application of the rule might lead to a proof of an instance of $p(x)$.

4.1.1 *Assets.* Let us have another look at the program discussed in Example 2.25:

$$\mathbb{P}_{leq} = \left\{ \begin{array}{lcl} r_I & @ & \mathbf{leq}(x,y) \wedge \mathbf{leq}(x,y) \Leftrightarrow \mathbf{leq}(x,y) \\ r_R & @ & \mathbf{leq}(x,x) \Leftrightarrow \top \\ r_S & @ & \mathbf{leq}(x,y) \wedge \mathbf{leq}(y,x) \Leftrightarrow x \doteq y \\ r_T & @ & \mathbf{leq}(x,y) \wedge \mathbf{leq}(y,z) \Rightarrow \mathbf{leq}(x,z) \end{array} \right\}$$

We claimed earlier that the rules of the program implement properties of the partial-order relation. We shall now substantiate this claim. Let us take a look at the logical reading of the program:

$$\mathbb{P}^{\dagger}_{leq} = \left\{ \begin{array}{lll} \forall x,y. & (\ \mathbf{leq}(x,y) \wedge \mathbf{leq}(x,y) \leftrightarrow \mathbf{leq}(x,y) \ ) \\ \forall x. & (\ \mathbf{leq}(x,x) \leftrightarrow \top \ ) \\ \forall x,y. & (\ \mathbf{leq}(x,y) \wedge \mathbf{leq}(y,x) \leftrightarrow x = y \ ) \\ \forall x,y,z. & (\ \mathbf{leq}(x,y) \wedge \mathbf{leq}(y,z) \rightarrow \mathbf{leq}(x,z) \ ) \end{array} \right\}$$

The translations of r_R, r_R and r_T logically express the properties of a partial-order relation. The translation of r_I is a logical tautology and thus redundant to the logical reading. By Lemma 4.1 it follows that $\mathbb{P}_{\leq}$ correctly implements the partial-order relation, i.e. every state in a computation controlled by $\mathbb{P}_{\leq}$ is a logical consequence of the initial state.

Consider furthermore the program from Example 2.27:

$$\mathbb{P}^{\dagger}_{birds} = \left\{ \begin{array}{lcl} \mathbf{bird} & \Leftrightarrow & \mathbf{albatross} \vee \mathbf{penguin} \\ \mathbf{penguin} \wedge \mathbf{flies} & \Leftrightarrow & \bot \end{array} \right\}$$

Its classical semantics looks as follows:

$$\mathbb{P}^{\dagger}_{birds} = \left\{ \begin{array}{lcl} (\ \mathbf{bird} & \leftrightarrow & \mathbf{albatross} \vee \mathbf{penguin} \ ) \\ (\ \mathbf{penguin} \wedge \mathbf{flies} & \leftrightarrow & \bot \ ) \end{array} \right\}$$

We observe that the logical reading nicely captures the theory implemented by $\mathbb{P}_{birds}$, including the meaning of disjunction.

4.1.2 *Limitations.* There are, however, several limitations to the classical declarative semantics of CHR, which shall be discussed in the following:

Directionality. One limitation lies in the fact that the classical declarative semantics does not capture the inherent directionality of CHR rules. Rather, all states within a computation are considered logically equivalent. Consider e.g. the minimal CHR program

$$\mathbb{P}_{ab} = \{\mathbf{a} \Leftrightarrow \mathbf{b}\}$$

In this program, we can compute a state $\langle \mathbf{b}; \top; \emptyset \rangle$ from a state $\langle \mathbf{a}; \top; \emptyset \rangle$ but not vice versa. This is not captured in its logical reading ($\mathbf{a} \leftrightarrow \mathbf{b}$) which e.g. implies ($\mathbf{b} \rightarrow \mathbf{a}$). The classical declarative semantics cannot be used e.g. to show that the state $\langle \mathbf{a}; \top; \emptyset \rangle$ is not computable from state $\langle \mathbf{b}; \top; \emptyset \rangle$.

Candidate Elimination. Any program that stepwisely approximates a result by eliminating candidates eludes the classical semantics. Consider the following program from Example 2.26:

$$\mathbb{P}_{min} = \left\{ r_M \ @ \ \mathbf{min}(x) \setminus \mathbf{min}(y) \ \Leftrightarrow \ x \leq y \mid \top \right\}$$

On a fixed-point execution, the program correctly computes the minimum of all arguments of *min* constraints found in the store at the beginning of the computation. Its logical reading is unhelpful at best:

$$\mathbb{P}^{\dagger}_{min} = \{\forall x, y.x \leq y \rightarrow \mathbf{min}(x) \rightarrow (\mathbf{min}(y) \leftrightarrow \top)\}$$

We encounter similar problems for programs simulating destructive updates.

Deliberate Non-Determinism. Any program that makes deliberate use of the inherent non-determinism of CHR has a misleading declarative semantics as well. Consider the following program, which simulates a coin throw in an appropriate probabilistic semantics of CHR (cf. Frühwirth et al. [2002]). (Note that *coin* is a variable, *head* and *tail* are constants.)

$$\mathbb{P}_{coin} = \left\{ \begin{array}{l} \mathbf{throw}(coin) \ \Leftrightarrow \ coin \doteq head \\ \mathbf{throw}(coin) \ \Leftrightarrow \ coin \doteq tail \end{array} \right\}$$

The logical reading of this program implies $\forall coin.(coin \doteq head \leftrightarrow coin \doteq tail)$. From this follows $head \doteq tail$ and – since *head* and *tail* are distinct constants – falsity $\bot$. The program's logical reading is thus inconsistent, trivially implying anything:

$$\mathbb{P}^{\dagger}_{coin} \models \forall x.\mathbf{pig}(x) \rightarrow \mathbf{flies}(x)$$

Multiplicities. While CHR faithfully keeps track of the multiplicities of constraints, this aspect eludes the classical semantics. Consider the idempotence rule from Example 2.25, which removes multiple occurrences of the same constraint:

$$r_I \ @ \ \mathbf{leq}(x, y) \wedge \mathbf{leq}(x, y) \ \Leftrightarrow \ \mathbf{leq}(x, y)$$

The logical reading of this rule is a tautology, falsely suggesting that the rule is redundant:

$$r_I^{\dagger} = \forall x, y.(\mathbf{leq}(x, y) \wedge \mathbf{leq}(x, y) \ \leftrightarrow \ \mathbf{leq}(x, y))$$

In conclusion, the classical declarative semantics is a powerful tool to prove the soundness and a certain notion of completeness of a program that directly implements a logical theory. This includes theories that involve disjunction. It is not adequate to capture directionality or updates, or to express the logic behind programs that make deliberate use of non-determinism or rely on the multiplicities of constraints. It is not suitable to prove safety conditions, i.e. to show that a certain intermediate or final state cannot be derived from a certain initial state.

4.2 The Axiomatic Linear-Logic Semantics for CHR

Our linear-logic semantics is based on two observations: Firstly, the difference in behaviour between built-in and user-defined constraints in CHR resembles the difference between linear and banged atoms in linear logic. Secondly, the application of simplification rules on user-defined constraints resembles the application of modus ponens in linear logic. Building on the first observation, we define an adequate representation of CHR constraints in

Atomic built-in constraints:	$c_b(\bar{t})^L ::= !c_b(\bar{t})$
Atomic user-defined constraints:	$c_u(\bar{t})^L ::= c_u(\bar{t})$
Falsity:	$\bot^L ::= \mathbf{0}$
Empty constraint/goal:	$\top^L ::= \mathbf{1}$
Constraints/goals:	$(\mathbb{G}_1 \wedge \mathbb{G}_2)^L ::= \mathbb{G}_1^L \otimes \mathbb{G}_2^L$
Disjunction within goals:	$(\mathbb{G}_1 \vee \mathbb{G}_2)^L ::= \mathbb{G}_1^L \oplus \mathbb{G}_2^L$
States:	$\langle \mathbb{G}; \mathbb{V} \rangle^L ::= \exists_{-\mathbb{V}}.\mathbb{G}^L$
Configurations:	$(\bar{S} \vee \bar{T})^L ::= \bar{S}^L \oplus \bar{T}^L$
Empty configuration:	$(\varepsilon)^L ::= \mathbf{0}$

Fig. 2. Translation of goals, states, configurations

linear logic. Translation to linear logic will be denoted as $(\cdot)^L$. The translation is summed up in Fig. 2.

Atomic user-defined and built-in constraints are mapped to atoms and banged atoms, respectively. Classical conjunction is mapped to multiplicative conjunction for both built-in and user-defined constraints. This mapping is motivated by the fact that multiplicative conjunction is aware of multiplicities and has no notion of weakening, thus capturing the multiset semantics of user-defined constraints. It is also adequate for built-in constraints as the bangs attached to the atomic built-in constraints restores the set semantics. We observe the mapping of built-in constraints is equal to the translation quoted in Definition 3.3.

We also follow Definition 3.3 by mapping the empty goal $\top$ to $\mathbf{1}$ and falsity $\bot$ to $\mathbf{0}$. Is is an obvious choice to map the split connective $\vee$ to multiplicative disjunction $\oplus$, as absurdity $\mathbf{0}$ is neutral with respect to $\oplus$ just as failed states are neutral to configurations. The translation of states is analogous to the classical case, and the translation of configurations follows from the mapping of $\vee$ to $\oplus$.

Proper axioms. The constraint theory CT, the explicit equality theory ET and programs are translated to proper axioms. Firstly, we define a set of proper axioms encoding the constraint theory as well as modelling the interaction between equality $\doteq$ and user-defined constraints.

Definition 4.2 *(Σ_{CT}).* For built-in constraints $\mathbb{B}, \mathbb{B}'$ and sets of variables $\bar{x}, \bar{x}'$ such that $CT \models \exists \bar{x}.\mathbb{B} \rightarrow \exists \bar{x}'.\mathbb{B}'$, the following is a proper axiom:

$$\exists \bar{x}.\mathbb{B}^L \vdash_{\Sigma_{CT}} \exists \bar{x}'.\mathbb{B}'^L$$

We denote the set of all such axioms as Σ_{CT}.

Definition 4.3 *($\Sigma_{\doteq}$).* For goals $\mathbb{G}, \mathbb{G}'$ and sets of variables $\bar{x}, \bar{x}'$ such that $ET \models \exists \bar{x}.\mathbb{G} \rightarrow \exists \bar{x}'.\mathbb{G}'$, the following is a proper axiom:

$$\exists \bar{x}.\mathbb{G}^L \vdash_{\Sigma_{\doteq}} \exists \bar{x}'.\mathbb{G}'^L$$

is a proper axiom. We denote the set of all such axioms as $\Sigma_{\doteq}$.

Definition 4.4 *($\Sigma_{\mathbb{P}}$).* If $r \ @ \ H_1 \setminus H_2 \ \Leftrightarrow \ G \mid B_b \wedge B_u$ is a variant of a rule with local variables $\bar{y}_r$, the sequent

$$H_1^L \otimes H_2^L \otimes G^L \vdash_{\Sigma_{\mathbb{P}}} H_1^L \otimes \exists \bar{y}_r.(B_b^L \otimes B_u^L \otimes G^L)$$

is a proper axiom. For a program $\mathbb{P}$, we denote the set of all axioms derived from its rules as $\Sigma_{\mathbb{P}}$.

$$\frac{CT \models \exists \bar{x}.\mathbb{B} \rightarrow \exists \bar{x}'.\mathbb{B}'}{\exists \bar{x}.\mathbb{B}^L \vdash_{\Sigma_{CT}} \exists \bar{x}'.\mathbb{B}'^L}\ (\Sigma_{CT}) \qquad \frac{ET \models \exists \bar{x}.\mathbb{G} \rightarrow \exists \bar{x}'.\mathbb{G}'}{\exists \bar{x}.\mathbb{G}^L \vdash_{\Sigma_{\doteq}} \exists \bar{x}'.\mathbb{G}'^L}\ (\Sigma_{\doteq})$$

$$\frac{(r\ @\ H_1 \setminus H_2 \Leftrightarrow G \mid B)\,[x/y] \in \mathbb{P}}{H_1^L \otimes H_2^L \otimes G^L \vdash_{\Sigma_{\mathbb{P}}} H_1^L \otimes \exists \bar{y}_r.(B^L \otimes G^L)}\ (\Sigma_{\mathbb{P}})$$

Fig. 3. The axiomatic linear-logic semantics for CHR$^\vee$

The existential quantification of the local variables $\bar{y}_r$ corresponds to the fact that these variables are by definition disjoint from $vars(H_1, H_2, \mathbb{U}, \mathbb{B}, \mathbb{V})$, assuring that fresh variables are introduced for the local variables of the rule. Fig. 3 sums up the three sets of proper axioms, represented as inference rules.

4.3 Soundness of the Axiomatic Semantics

In this section, we prove the soundness of the axiomatic linear-logic semantics for CHR with respect to the operational semantics. The proof can be found in the appendix.

Lemma 4.5 ($\equiv \Rightarrow \dashv\vdash_\Sigma$). *(1) Let CT be a constraint theory and $\Sigma = \Sigma_{CT} \cup \Sigma_{\doteq}$. For arbitrary CHR states S, T, we have:*

$$S \equiv_e T \Rightarrow S^L \dashv\vdash_\Sigma T^L$$

(2) For arbitrary configurations $\bar{S}, \bar{T}$, we have:

$$\bar{S} \equiv_\vee \bar{T} \Rightarrow \bar{S}^L \dashv\vdash_\Sigma \bar{T}^L$$

Theorem 4.6 states the soundness of our semantics. The proof can be found in the appendix.

Theorem 4.6 Soundness. *For any CHR$^\vee$ program $\mathbb{P}$, constraint theory CT and configurations $\bar{U}, \bar{V}$,*

$$[\bar{U}] \mapsto^* [\bar{V}] \quad \Rightarrow \quad \bar{U}^L \vdash_\Sigma \bar{V}^L$$

where $\Sigma = \Sigma_{\mathbb{P}} \cup \Sigma_{CT} \cup \Sigma_{\doteq}$.

To illustrate Theorem 4.6 presented in Sect. 4.3, we give an example of a CHR$^\vee$ derivation and show that it corresponds to a valid linear logic judgement:

Example 4.7. Let $\mathbb{P}$ be the partial-order constraint solver from Example 2.25 and let *CT* be a minimal constraint theory. We observe that under $\mathbb{P}$, we have:

$$[\langle \mathbf{leq}(3, a) \wedge a = 3; \emptyset\rangle] \mapsto^* [\langle \top; \emptyset\rangle]$$

This corresponds to the judgement $\langle 3 \leq a; a = 3; \emptyset\rangle^L \vdash_\Sigma \langle \top; \top; \emptyset\rangle^L$ or $\exists a.(3 \leq a \otimes\ !a = 3) \vdash_\Sigma \mathbf{1}$, respectively, where $\Sigma = \Sigma_{CT} \cup \Sigma_{\doteq} \cup \Sigma_{\mathbb{P}}$. The following is a proof of this judgement:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathbf{leq}(3,x) \otimes\ !x \doteq 3 \vdash_\Sigma \mathbf{leq}(x,x) \otimes\ !x \doteq 3}\ (\Sigma_{\doteq}) \quad \dfrac{\dfrac{\dfrac{}{\mathbf{leq}(x,x) \vdash_\Sigma \mathbf{1}}\ (\Sigma_{\mathbb{P}}) \quad \dfrac{}{!x \doteq 3 \vdash_\Sigma !x \doteq 3}\ (Identity)}{\mathbf{leq}(x,x), !x \doteq 3 \vdash_\Sigma \mathbf{1} \otimes\ !x \doteq 3}\ (R\otimes)}{\mathbf{leq}(x,x) \otimes\ !x \doteq 3 \vdash_\Sigma \mathbf{1} \otimes\ !x \doteq 3}\ (L\otimes)}{\mathbf{leq}(3,x) \otimes\ !x \doteq 3 \vdash_\Sigma \mathbf{1} \otimes\ !x \doteq 3}\ (Cut) \quad \dfrac{}{\mathbf{1} \otimes\ !x \doteq 3 \vdash_\Sigma \mathbf{1}}\ (\Sigma_{CT})}{\mathbf{leq}(3,x) \otimes\ !x \doteq 3 \vdash_\Sigma \mathbf{1}}\ (Cut)}{\exists a.(\mathbf{leq}(3,a) \otimes\ !a \doteq 3) \vdash_\Sigma \mathbf{1}}\ (L\exists)$$

The sequent $\mathbf{1} \otimes !x \doteq 3 \vdash \mathbf{1}$ is a tautology and as such could be derived without proper axioms, but it is also trivially included in Σ_{CT}.

While the soundness result for our semantics is straightforward, defining completeness is not quite as simple. Consider the following example:

Example 4.8. In the proof tree given in Example 4.7 we use the following proper axiom from Σ_{CT}:

$$\mathbf{1} \otimes !x \doteq 3 \vdash \mathbf{1}$$

This implies:

$$\langle x \doteq 3; \{x\}\rangle^L \vdash_\Sigma \langle \top; \emptyset\rangle^L$$

We observe, however, that $\langle x \doteq 3; \{x\}\rangle \mapsto^* \langle \top; \emptyset\rangle$ is untrue.

In the following section, we develop the notion of *state entailment* and apply this notion to specify a completeness result.

4.4 Completeness of the Axiomatic Semantics

In this section, we define the notion of *entailment* and apply it to formulate our theorem of completeness. We introduce it first for flat states and then extend it to configurations. We present it alongside various properties that follow from it and will be used in upcoming sections.

Definition 4.9 *Entailment of Flat States*. Entailment between flat states, written as $\cdot \rhd \cdot$, is the smallest partial-order relation over equivalence classes of flat states that satisfies the following conditions:

(1) *(Weakening of the Built-In Store)* For states $\langle \mathbb{U} \wedge \mathbb{B}; \mathbb{V}\rangle, \langle \mathbb{U} \wedge \mathbb{B}'; \mathbb{V}\rangle$ with local variables $\bar{s}, \bar{s}'$ such that $CT \models \forall(\exists \bar{s}.\mathbb{B} \rightarrow \exists \bar{s}'.\mathbb{B}')$, we have:

$$[\langle \mathbb{U} \wedge \mathbb{B}; \mathbb{V}\rangle] \rhd [\langle \mathbb{U} \wedge \mathbb{B}'; \mathbb{V}\rangle]$$

(2) *(Omission of Global Variables)*

$$[\langle \mathbb{U} \wedge \mathbb{B}; \{x\} \cup \mathbb{V}\rangle] \rhd [\langle \mathbb{U} \wedge \mathbb{B}; \mathbb{V}\rangle]$$

Analogously to state entailment, we define a notion of *configuration entailment*:

Definition 4.10 *Entailment of Configurations*. Entailment of configurations, denoted as $\cdot \blacktriangleright \cdot$, is the smallest reflexive-transitive relation over equivalence classes of configurations satisfying the following conditions:

(1) *Weakening:* For any state S and configuration $\bar{T}$:

$$[\bar{T}] \blacktriangleright [S \vee \bar{T}]$$

(2) *Redundance of Stronger States:* For any CHR$^\vee$ states S_1, S_2, T such that $S_1 \rhd S_2$:

$$[S_1 \vee S_2 \vee \bar{T}] \blacktriangleright [S_2 \vee \bar{T}]$$

The following property follows from the Definition 4.9 and Definition 4.10:

Property 4.11 ($\rhd \Rightarrow \blacktriangleright$). *For CHR$^\vee$ states S_1, S_2 such that $S_1 \rhd S_2$:*

$$[S_1 \vee \bar{T}] \blacktriangleright [S_2 \vee \bar{T}]$$

Proof. $[S_1 \vee \bar{T}] \blacktriangleright [S_2 \vee S_1 \vee \bar{T}] = [S_1 \vee S_2 \vee \bar{T}] \blacktriangleright [S_2 \vee \bar{T}]$ □

Theorem 4.12 gives a decidable criterion for state entailment. The criterion requires that the global variables of the entailed state are contained in the global variables of the entailing state. This is never a problem, as we may choose representatives of the respective equivalence classes that satisfy the condition. The proof can be found in the appendix.

Theorem 4.12 Criterion for $\rhd$. *Let $S = \langle \mathbb{U}; \mathbb{B}; \mathbb{V} \rangle, S' = \langle \mathbb{U}'; \mathbb{B}'; \mathbb{V}' \rangle$ be CHR states where the local variables $\bar{l}'$ of S' have been renamed apart from the local variables of S and where $\mathbb{V}' \subseteq \mathbb{V}$. Then we have:*

$$[S] \rhd [S'] \quad \Leftrightarrow \quad ET, CT \models \forall(\mathbb{B} \rightarrow \exists \bar{l}'.((\mathbb{U} \doteq \mathbb{U}') \wedge \mathbb{B}'))$$

Lemma A.5 establishes an important relationship between configuration entailment and state transition. The proof can be found in the appendix.

Lemma 4.13 Exchange of $\mapsto$ and $\blacktriangleright$. *Let $\bar{S}, \bar{U}, \bar{T}$ be configurations. If $\bar{S} \blacktriangleright \bar{U}$ and $\bar{U} \mapsto^r \bar{T}$ then there exists a configuration $\bar{V}$ such that $\bar{S} \mapsto^* \bar{V}$ and $\bar{V} \blacktriangleright \bar{T}$.*

The completeness of our semantics is formulated in Theorem 4.14. The proof can be found in the appendix.

Theorem 4.14 Completeness of the Semantics for CHR$^\vee$. *Let $\bar{S}, \bar{T}$ be configurations, let $\mathbb{P}$ be a program and CT be a constraint theory. If the sequent $\bar{S}^L \vdash \bar{T}^L$ is provable in a sequent calculus system with proper axioms $\Sigma = \Sigma_{CT} \cup \Sigma_{\doteq} \cup \Sigma_{\mathbb{P}}$ then there exists a configuration $\bar{U}$ such that $\bar{S} \mapsto^* \bar{U}$ and $\bar{U} \rhd \bar{T}$.*

The following example illustrates the completeness theorem:

Example 4.15. We consider the partial-order program $\mathbb{P}$ given in Example 2.25 and a minimal constraint theory CT. For $\Sigma = \Sigma_{\mathbb{P}} \cup \Sigma_{CT} \cup \Sigma_{\doteq}$, we have

$$a \leq b \otimes b \leq c \otimes c \leq a \vdash_\Sigma !a \doteq b$$

which equals:

$$\langle a \leq b \wedge b \leq c \wedge c \leq a; \top; \{a, b, c\} \rangle^L \vdash_\Sigma \langle \top; a \doteq b; \{a, b\} \rangle^L$$

This corresponds to:

$$\langle a \leq b \wedge b \leq c \wedge c \leq a; \top; \{a, b, c\} \rangle \mapsto^* \langle \top; a \doteq b \wedge a \doteq c; \{a, b, c\} \rangle \rhd \langle \top; a \doteq b; \{a, b\} \rangle$$

Lemma A.6 and Lemma 4.16 establish the relationship between entailment and logical judgement. The proofs can be found in the appendix.

Lemma 4.16 ($\blacktriangleright \Leftrightarrow \vdash$). *For configurations $\bar{S}, \bar{T}$, we have $[\bar{S}] \blacktriangleright [\bar{T}]$ if and only if $\bar{S}^L \vdash_\Sigma \bar{T}^L$ where $\Sigma = \Sigma_{CT} \cup \Sigma_{\doteq}$.*

The following example illustrates Lemma 4.16:

Example 4.17. In Example 4.8, we showed that the following judgement, which does not correspond to any transition in CHR, is provable in our sequent calculus system:

$$\langle x \doteq 3; \{x\} \rangle^L \vdash_\Sigma \langle \top; \emptyset \rangle^L$$

We observe that the two states are connected by the entailment relation:

$$\langle x \doteq 3; \{x\} \rangle \rhd \langle \top; \emptyset \rangle$$

In the following section, we will show that state entailment precisely covers the discrepance between transitions in a CHR program and judgements in its corresponding sequent calculus system as exemplified in Example 4.8

4.5 Encoding Programs and Constraint Theories

In the axiomatic linear-logic semantics presented in Sect. 4.2 to Sect. 4.4, only states are represented in logical judgements. Both programs and constraint theories disappear into the proper axioms of a sequent calculus system and hence are not objects of logical reasoning.

In this section, we show how to encode programs and constraint theories into logical judgements, enabling us to reason directly about them as well. In Sect. 5.3, we will use this encoding to decide operational equivalence of programs. As a further benefit, a complete encoding of programs and constraint theories assures the existence of cut-free proofs for the respective judgements and ensure compatibility with established methods for automated proof search methods relying on this property.

As usual, $(\cdot)^L$ stands for translation into linear logic.

Encoding of Constraint and Equality Theories. The constraint theory CT and the equality theory ET are encoded according to the translation quoted in Def. 3.3.

Definition 4.18 CT^L,ET^L. For a constraint theory CT and an equality theory ET their encodings are given as $CT^L ::= CT^*$ and $ET^L ::= ET^*$

Encoding of $\Sigma_{\mathbb{P}}$. The translation of CHR rules follows the same lines as the encoding of the CT axioms:

Definition 4.19 $(R^L, \mathbb{P}^L)$. (1) Let $R = r \ @ \ H_1 \setminus H_2 \Leftrightarrow G \mid B$ be a CHR rule with local variables $\bar{y}_r$. Then its linear-logic reading R^L is defined as:

$$R^L ::= !\forall(H_1^L \otimes H_2^L \otimes G^L \multimap H_1^L \otimes \exists \bar{y}_r.(B^L \otimes G^L))$$

(2) Let $\mathbb{P} = \{R_1, \ldots, R_n\}$ be a CHR program. Then its linear-logic reading $\mathbb{P}^L$ is defined as:

$$\mathbb{P}^L ::= \bigcup_{R \in \mathbb{P}} R^L$$

For the encoding semantics, the following soundness and completeness theorem holds:

Theorem 4.20 Soundness and Completeness. *Let $\bar{S}, \bar{T}$ be configurations. There exists a configuration $\bar{U}$ such that*

$$\bar{S} \mapsto^* \bar{U} \text{ and } \bar{U} \rhd \bar{T}$$

in a program $\mathbb{P}$ and a constraint theory CT if and only if

$$\mathbb{P}^L, ET^L, CT^L \vdash \forall(\bar{S}^L \multimap \bar{T}^L)$$

As the encoding semantics is logically equivalent to the one proposed in Betz [2007], Theorem 4.20 also proves the equivalence of the axiomatic linear-logic semantics with that earlier semantics.

4.6 Discussion

In this section, we shall revisit the limitations of the classical declarative semantics discussed in Section 4.1 and show how the linear-logic semantics overcomes these limitations.

Directionality. We showed that the classical declarative semantics does not capture the inherent directionality of CHR rules:

$$\mathbf{a} \Leftrightarrow \mathbf{b}$$

The classical declarative semantics cannot be used e.g. to show that the state $\langle \mathbf{a}; \top; \emptyset \rangle$ is not computable from state $\langle \mathbf{b}; \top; \emptyset \rangle$ in this program. On the contrary, in the linear-logic semantics we have

$$\mathbf{a} \vdash_\Sigma \mathbf{b} \qquad \text{but} \qquad \mathbf{b} \nvdash_\Sigma \mathbf{a}$$

Candidate Elimination. We also showed that the classical declarative semantics does not capture candidate generation or destructive updates.

$$\mathbb{P}_{min} = \left\{ r_M \ @ \ \mathbf{min}(x) \setminus \mathbf{min}(y) \ \Leftrightarrow \ x \leq y \mid \top \right\}$$

Its linear-logic declarative reading faithfully captures the strategy followed by the program:

$$\mathbb{P}^L_{min} = \{\forall x, y.\, \mathbf{min}(x) \otimes \mathbf{min}(y) \otimes (!x \leq y) \leftrightarrow \mathbf{min}(x) \otimes (!x \leq y)\}$$

Deliberate Non-Determinism. We furthermore showed that the classical declarative semantics is inadequate to capture deliberately non-deterministic programs:

$$\mathbb{P}_{coin} = \left\{ \begin{array}{l} \mathbf{throw}(coin) \ \Leftrightarrow \ coin \doteq head \\ \mathbf{throw}(coin) \ \Leftrightarrow \ coin \doteq tail \end{array} \right\}$$

In the classical declarative semantics, the logical reading for the above program is contradictory, i.e. it proves falsity. In the linear-logic semantics, this is not the case: $\mathbb{P}^L_{coin} \nvdash_\Sigma \mathbf{0}$. Rather, we can show that the non-deterministic choice is mapped to internal choice:

$$\mathbb{P}^L_{coin} \dashv\vdash \{\forall coin.\, \mathbf{throw}(coin) \multimap (!coin \doteq head) \& (!coin \doteq tail)\}$$

Multiplicities. We also showed that in the classical declarative semantics, the logical reading of the idempotence rule from Example 2.25 is a tautology:

$$r_I \ @ \ \mathbf{leq}(x, y) \wedge \mathbf{leq}(x, y) \ \Leftrightarrow \ \mathbf{leq}(x, y)$$

In linear logic, this is not the case. Instead, we get a faithful description of its operational behaviour:

$$r^L_I = \forall x, y.(\mathbf{leq}(x, y) \otimes \mathbf{leq}(x, y) \ \multimap \ \mathbf{leq}(x, y))$$

5. LINEAR-LOGIC REASONING IN CHR$^\vee$

In this section, we outline how our results can be applied to reason over programs and their respective observables. In Section 5.1, we discuss the conditions under which we can use linear logic to reason about CHR$^\vee$. In Section 5.2, we discuss the relationship between the linear-logic semantics and program observables. In Section 5.3, we show how we can compare the operational semantics of programs by means of their linear-logic semantics.

5.1 Congruence and Equivalence

In this section, we discuss the conditions under which logical equivalence and configuration equivalence coincide and identify a segment of CHR$^\vee$ where these conditions apply.

5.1.1 *Limitations of the Linear-Logic Semantics.* As we have established equivalence of states and configurations as a convenient low-level abstraction to reason about CHR$^\vee$ computations, we assume that this is also the granularity we desire when we apply linear logic to reason about CHR$^\vee$. On the other hand, the granularity of reasoning over a model in a logical system is naturally determined by logical equivalence.

Definition 5.1 *Congruence of Configurations*. Given a constraint theory CT, two configurations $\bar{S}, \bar{T}$ are considered *congruent* if $\bar{S} \blacktriangleright \bar{T}$ and $\bar{T} \blacktriangleright \bar{S}$. Congruence of $\bar{S}$ and $\bar{T}$ is denoted as $\bar{S} \blacktriangleleft\blacktriangleright \bar{T}$.

From Lemma 4.16, it follows that congruence of configurations coincides with logical equivalence over the respective linear-logic readings:

Property 5.2. *For arbitrary configurations $\bar{S}, \bar{T}$, we have $\bar{S} \blacktriangleleft\blacktriangleright \bar{T} \Leftrightarrow \bar{S} \dashv\vdash \bar{T}$.*

Hence, any reasoning over CHR$^\vee$ via the linear-logic semantics is necessarily modulo congruence. It shows, however, that congruence, does not necessarily coincide with equivelence:

Example 5.3. Consider the configurations $\bar{S} = \langle c_u(X)\rangle$ and $\bar{T} = \langle c_u(0)\rangle \vee \langle c_u(X)\rangle$. Since $\bar{S} \blacktriangleright \bar{T}$ and $\bar{T} \blacktriangleright \bar{S}$, we have $\bar{S} \blacktriangleleft\blacktriangleright \bar{T}$. However, the two are not equivalent: $\bar{S} \not\equiv_\vee \bar{T}$.

To emphasize that this is not merely a cosmetic problem, we show that congruence does not in general comply with rule applications:

Example 5.4 *Non-Compliance with Rule Application*. By compliance, we mean the property that for arbitrary configurations $\bar{S}, \bar{S}', \bar{T}$ such that $\bar{S} \equiv_\vee \bar{S}'$ and $\bar{S} \mapsto^* \bar{T}$, there exists a $\bar{T}'$ such that $\bar{S}' \mapsto \bar{T}'$ and $\bar{T}' \equiv_\vee \bar{T}$.

Let $\bar{S} = \langle c_u(X)\rangle$ and $\bar{T} = \langle c_u(0)\rangle \vee \langle c_u(X)\rangle$ be configurations. As $\langle c_u(0)\rangle \rhd \langle c_u(X)\rangle$, we have congruence: $\bar{S} \blacktriangleleft\blacktriangleright \bar{T}$. Now consider the following minimal CHR program:

$$r \mathbin{@} c_u(0) \Leftrightarrow d_u(0)$$

We observe that we have $\bar{T} \mapsto^r \langle d_u(0)\rangle \vee \langle c_u(X)\rangle$ whereas $\bar{S}$ is an answer configuration i.e. it does not allow any further transition. We thus observe that congruence of configurations is not in general compliant with rule application.

However, we can make a somewhat weaker statement about the relationship between congruence and rule application:

Property 5.5 Weak Compliance with Rule Application. *Let $\bar{S}, \bar{S}', \bar{T}$ be configurations such that $\bar{S} \blacktriangleleft\blacktriangleright \bar{S}'$. Then $\bar{S} \mapsto^* \bar{T}$ implies that there exists a $\bar{T}'$ such that $\bar{S}' \mapsto^* \bar{T}'$ and $\bar{T}' \blacktriangleright \bar{T}$.*

Proof. *$\bar{S} \blacktriangleleft\blacktriangleright \bar{S}'$ implies $\bar{S}' \blacktriangleright \bar{S}$. Furthermore, we have $\bar{S} \mapsto^* \bar{T}$. Hence, Lemma 4.13 proves $\bar{S}' \mapsto^* \bar{T}'$ and $\bar{T}' \blacktriangleright \bar{T}$.* □

In order to allow precise logical reasoning over CHR$^\vee$, we identify a segment of CHR$^\vee$ where congruence and equivalence of configurations coincide.

5.1.2 *Compactness and Analyticness.* Considering Example 5.4, we observe that we can construct similar examples for any configuration $\bar{S}$ of the form $\bar{S} = S_1 \vee S_2 \vee \bar{S}'$ where $S_1 \rhd S_2$ by postulating a rule r that fires for S_1 but not for S_2. Hence, our strategy is to explicitly exclude such configurations from consideration. We introduce the notion of *compactness*:

Definition 5.6 *Compactness*. A configuration $\bar{S}$ is called *compact* if it does not have a flat normal form $\bar{S}' = S_1 \vee S_2 \vee \ldots \vee \bar{S}_n$ where $S_1 \not\equiv S_\perp$ and $S_1 \rhd S_2$.

Compactness of a configuration $\bar{S}$ can straightforwardly be decided by transforming $\bar{S}$ into a flat normal form $\bar{S}_F$, removing all failed states from $\bar{S}_F$ and pairwisely verifying $(S_i \not\rhd S_j)$ for all remaining states where $i \neq j$.

We extend the compactness property to equivalence classes of configurations in the obvious manner. The following lemma states that compactness guarantees that congruence and equivalence coincide.

LEMMA 5.7. *Let $\bar{S}, \bar{T}$ be compact configurations such that $\bar{S} \blacktriangleleft\!\blacktriangleright \bar{T}$. Then $\bar{S} \equiv_\vee \bar{T}$.*

PROOF. *We assume that both $\bar{S}$ and $\bar{T}$ are in normal form: $\bar{S} \equiv_\vee S_1 \vee \ldots \vee S_n, \bar{T} \equiv_\vee T_1 \vee \ldots \vee T_m$. From Def. 4.10 follows that for every consistent S_i, we have a T_j such that $S_i \rhd T_j$, and for every consistent T_j there is an S_i such that $T_j \rhd S_i$. It follows that for every consistent S_i, we have T_j, S_k such that $S_i \rhd T_j \rhd S_k$. As $\bar{S}$ is compact, $S_i \rhd S_k$ implies $i = k$ and furthermore $S_i \equiv T_j$. As $\bar{T}$ is compact, there is exactly one T_j such that $S_i \equiv T_j$. Since every consistent S_i has a unique corresponding state T_j with $S_i \equiv T_j$ and vice versa, Def. 2.14 implies that $\bar{S} \equiv_\vee \bar{T}$.* □

We furthermore introduce a well-behavedness property for CHR$^\vee$ programs which guarantees compactness of derived configurations:

Definition 5.8 *Analyticness*. (1) A CHR$^\vee$ program is called *analytic* for a class of initial states $\mathbb{S}$ if for any flat state $S \in \mathbb{S}$ and configuration $\bar{T}$ such that $[S] \mapsto^* [\bar{T}]$, we have that $\bar{T}$ is compact.

(2) A CHR$^\vee$ program is called *generally analytic* if for any flat initial state S and configuration $\bar{T}$ such that $[S] \mapsto^* [\bar{T}]$, we have that $\bar{T}$ is compact.

It appears that a large number of CHR$^\vee$ programs satisfy this property for their intended class of initial states. This is due to the fact that an analyticness of a CHR$^\vee$ program means that it explores a search space non-redundantly. (This motivates the term *analytic*.)

Example 5.9 *Analyticness*. Consider the following program fragment:

$$\mathbb{P} = \left\{ \begin{array}{lcll} r_{src} & @ & \mathbf{search}(x,y) & \Leftrightarrow \ \mathbf{leq}(x,y) \vee \mathbf{geq}(x,y) \\ r_{leq} & @ & \mathbf{leq}(x,y) & \Leftrightarrow \ \mathbf{lower}(x,y) \vee x \doteq y \\ r_{geq} & @ & \mathbf{geq}(x,y) & \Leftrightarrow \ \mathbf{greater}(x,y) \vee x \doteq y \end{array} \right\}$$

Let $S_0 = \langle\, \mathbf{search}(x,y); \mathbb{V}\rangle$ and $\mathbb{V} = \{x, y\}$. We then have:

$$S_0 \mapsto^* \langle\, \mathbf{lower}(x,y); \mathbb{V}\rangle \vee \langle x \doteq y; \mathbb{V}\rangle \vee \langle x \doteq y; \mathbb{V}\rangle \vee \langle\, \mathbf{greater}(x,y); \mathbb{V}\rangle$$

Since $\langle x \doteq y; \mathbb{V}\rangle \rhd \langle x \doteq y; \mathbb{V}\rangle$, the final state is non-compact. Hence, $\mathbb{P}$ is not analytic for the initial state S_0.

In contrast, the following program is analytic for the class of initial states of the form $\langle\, \mathbf{search}(t, t'); \mathbb{V}\rangle$ where t, t' are arbitary terms. (This can be proved by building a state graph).

$$\mathbb{P}' = \left\{ \begin{array}{lcll} r_{src} & @ & \mathbf{search}(x,y) & \Leftrightarrow \ \mathbf{lower}(x,y) \vee \mathbf{geq}(x,y) \\ r_{leq} & @ & \mathbf{leq}(x,y) & \Leftrightarrow \ \mathbf{lower}(x,y) \vee x \doteq y \\ r_{geq} & @ & \mathbf{geq}(x,y) & \Leftrightarrow \ \mathbf{greater}(x,y) \vee x \doteq y \end{array} \right\}$$

However, the program is not generally analytic, as for example:

$$\begin{aligned}\langle \mathbf{search}(x,y) \wedge \mathbf{search}(x,y); \mathbb{V}\rangle \mapsto^* \; & \langle \mathbf{lower}(x,y) \wedge \mathbf{lower}(x,y); \mathbb{V}\rangle \vee \\ & \langle \mathbf{geq}(x,y) \wedge \mathbf{lower}(x,y); \mathbb{V}\rangle \vee \\ & \langle \mathbf{lower}(x,y) \wedge \mathbf{geq}(x,y); \mathbb{V}\rangle \vee \\ & \langle \mathbf{geq}(x,y) \wedge \mathbf{geq}(x,y); \mathbb{V}\rangle\end{aligned}$$

and $\langle \mathbf{geq}(x,y) \wedge \mathbf{lower}(x,y); \mathbb{V}\rangle \rhd \langle \mathbf{lower}(x,y) \wedge \mathbf{geq}(x,y); \mathbb{V}\rangle$.

We give a necessary and sufficient criterion for general analyticness of CHR$^\vee$ programs:

Lemma 5.10 Criterion for General Analyticness. *Let $\mathbb{P}$ be a CHR$^\vee$ program consisting of rules $R_1, \ldots, R_n$ where every rule R_i is of the form r @ $H_1 \setminus H_2 \Leftrightarrow G \mid (\mathbb{U}_1 \wedge \mathbb{B}_1) \vee \ldots \vee (\mathbb{U}_m \wedge \mathbb{B}_m)$. The program $\mathbb{P}$ is generally analytic* if and only if *$CT \not\models \exists(B_i \wedge B_j)$ for every $i, j \in \{1, \ldots, n\}$.*

Proof. *('⇒'): We assume w.l.o.g. $\mathbb{P}$ contains a rule of the form r @ $H_1 \setminus H_2 \Leftrightarrow G \mid (\mathbb{U}_1 \wedge \mathbb{B}_1) \vee (\mathbb{U}_2 \wedge \mathbb{B}_2)$ where $CT \models \exists(B_1 \wedge B_2)$. Then we have:*

$$\begin{aligned}& \langle H_1 \wedge H_2 \wedge H_2 \wedge G\rangle && (1)\\ \mapsto^r\; & \langle H_1 \wedge \mathbb{U}_1 \wedge \mathbb{B}_1 \wedge H_2 \wedge G\rangle \vee \langle H_1 \wedge \mathbb{U}_2 \wedge \mathbb{B}_2 \wedge H_2 \wedge G\rangle && (2)\\ \mapsto^r\; & \langle H_1 \wedge \mathbb{U}_1 \wedge \mathbb{B}_1 \wedge \mathbb{U}_1 \wedge \mathbb{B}_1 \wedge G\rangle \vee \langle H_1 \wedge \mathbb{U}_1 \wedge \mathbb{B}_1 \wedge \mathbb{U}_2 \wedge \mathbb{B}_2 \wedge H_2 \wedge G\rangle \vee \\ & \langle H_1 \wedge \mathbb{U}_2 \wedge \mathbb{B}_2 \wedge \mathbb{U}_1 \wedge \mathbb{B}_1 \wedge G\rangle \vee \langle H_1 \wedge \mathbb{U}_2 \wedge \mathbb{B}_2 \wedge \mathbb{U}_2 \wedge \mathbb{B}_2 \wedge G\rangle && (3)\end{aligned}$$

We observe, that the state we reach in step (3) *is not compact.*
('⇐'): We assume a single rule application $S \mapsto^r \bar{T}$ where the applied rule is of the form $R_i = r$ @ $H_1 \setminus H_2 \Leftrightarrow G \mid (\mathbb{U}_1 \wedge \mathbb{B}_1) \vee \ldots \vee (\mathbb{U}_m \wedge \mathbb{B}_m)$ such that $CT \not\models \exists(B_i \wedge B_j)$ for $i, j \in \{1, \ldots, n\}$.

It follows that for every $T_1 = \langle \mathbb{U}_1; \mathbb{B}_1; \mathbb{V}_1\rangle, T_2 = \langle \mathbb{U}_2; \mathbb{B}_2; \mathbb{V}_2\rangle$ such that $\bar{T} \equiv T_1 \vee T_2 \vee \bar{T}'$, we have $CT \not\models \exists(\mathbb{B}_1 \wedge \mathbb{B}_2)$. It follows by Lemma 4.12 that $T_1 \not\rhd T_2$.

As the built-in store grows monotonically stronger, correctness for the transitive closure of $\mapsto$ follows by induction. For the reflexive closure it follows from the fact that the state S is trivially a compact configuration. □

We observe that natural analyticsness holds for all CHR programs, i.e. CHR$^\vee$ programs without disjunction.

5.2 Reasoning About Observables

In this section, we show how to apply our results to reason about CHR$^\vee$ observables.

5.2.1 *Reasoning About Observables in Pure CHR.* We define two sets of observables based on the linear logic semantics, paralleling the observable sets of computable states and data-sufficient answers.

Definition 5.11. Let $\mathbb{P}$ be a pure CHR program, CT a constraint theory, and S an initial state. Assuming that $\Sigma = \Sigma_{\mathbb{P}} \cup \Sigma_{CT} \cup \Sigma_{\doteq}$, we distinguish two sets of observables based on the linear logic semantics:

$$\begin{aligned}\mathcal{L}^C_{\mathbb{P},CT}(S) &::= \{[\bar{T}] \mid S^L \vdash_\Sigma \bar{T}^L\}\\ \mathcal{L}^S_{\mathbb{P},CT}(S) &::= \{[\langle \bar{\mathbb{B}}; \mathbb{V}\rangle] \mid S^L \vdash_\Sigma \langle \bar{\mathbb{B}}; \mathbb{V}\rangle^L\}\end{aligned}$$

In the definition of $\mathcal{L}^S_{\mathbb{P},CT}(S)$, $\bar{\mathbb{B}}$ stands for a disjunction of built-in constraints. If the constraint theory CT is clear from the context or not important, we write the sets as $\mathcal{L}^C_{\mathbb{P}}(S), \mathcal{L}^S_{\mathbb{P}}(S)$.

The following definition and property establish the relationship between the logical observables $\mathcal{L}^C_{\mathbb{P}}$ and $\mathcal{L}^S_{\mathbb{P}}$ and the operational observables $C_{\mathbb{P}}$ and $\mathcal{S}_{\mathbb{P}}$

Definition 5.12 *Lower Closure of* $\blacktriangleright$. For any set $\mathbb{S}$ of equivalence classes of configurations:

$$\blacktriangledown\mathbb{S} ::= \{[\bar{T}] | \exists \bar{S} \in \mathbb{S}.[\bar{S}] \blacktriangleright [\bar{T}]\}$$

The following property establishes the relationship between the linear-logic observables and the ones based on the operational semantics:

Property 5.13 Relationship Between Observables. *For a pure CHR program $\mathbb{P}$, a constraint theory CT, and an initial state S, we have:*

$$\mathcal{L}^C_{\mathbb{P},CT}(S) = \blacktriangledown C_{\mathbb{P},CT}(S)$$
$$\mathcal{L}^S_{\mathbb{P},CT}(S) = \blacktriangledown \mathcal{S}_{\mathbb{P},CT}(S)$$

Proof. *By Theorem 4.6 and Theorem 4.14, we have that $S^L \vdash_\Sigma \bar{T}^L$* iff *there exists a configuration $\bar{U}$ such that $S^L \mapsto^* \bar{U}$ and $\bar{U} \blacktriangleright \bar{T}^L$. Hence $\mathcal{L}^C_{\mathbb{P},CT}(S) = \blacktriangledown C_{\mathbb{P},CT}(S)$. We recall that by Def. 2.19 $\mathcal{S}_{\mathbb{P},CT}(S)$ is the projection of $C_{\mathbb{P},CT}(S)$ to configurations with a representation of the form $\langle \bar{\mathbb{B}}; \mathbb{V} \rangle$ by Def. 5.11, $\mathcal{L}^S_{\mathbb{P},CT}(S)$ and is the projection of $\mathcal{L}^C_{\mathbb{P},CT}(S)$ to those configurations. Therefore, $\mathcal{L}^S_{\mathbb{P},CT}(S) = \blacktriangledown \mathcal{S}_{\mathbb{P},CT}(S)$.* □

From this relationship follow several properties that we can use to reason about the operational semantics. Firstly, in order to prove that a state S cannot develop into an empty configuration, it suffices to show that there exists any configuration $\bar{T}$, such that $[\bar{T}]$ is not contained in $C(S)$:

Property 5.14 Deciding Failure. *(1) Let $\mathbb{P}$ be a program $\mathbb{P}$, let CT be a constraint theory CT, and let S be a flat state. Then $\varepsilon \in C_{\mathbb{P},CT}(S)$* if and only if *$\varepsilon \in \mathcal{L}^C_{\mathbb{P},CT}(S)$.*

(2) If $\mathbb{P}$ is furthermore confluent, then a computation beginning with S will invariably fail if and only if *$\varepsilon \in \mathcal{L}^C_{\mathbb{P},CT}(S)$.*

Proof sketch. *Since $\varepsilon \blacktriangleright \bar{S}$ for any $\bar{S}$, we have that $\varepsilon \in \blacktriangledown C_{\mathbb{P},CT}(S)$* if and only if *$\varepsilon \in C_{\mathbb{P},CT}(S)$. The second property follows from Prop. 2.22.* □

Secondly, we can guarantee data-sufficient answers for a state S, if we can prove the empty resource $\mathbf{1}$ in linear logic. (Remember that $\mathbf{1}$ is the logical reading of the empty state $\langle \top; \emptyset \rangle$.)

Property 5.15 Deciding Existence of Data-Sufficient Answers. *(1) For a program $\mathbb{P}$, a constraint theory CT, and a flat state S, the state S has at least one data-sufficient answer* if and only if *$\langle \top; \emptyset \rangle \in \mathcal{L}^D_{\mathbb{P},CT}(S)$.*

(2) If $\mathbb{P}$ is furthermore confluent, S has exactly one data-sufficient answer if and only if *$\langle \top; \emptyset \rangle \in \mathcal{L}^D_{\mathbb{P},CT}(S)$.*

Proof sketch. *The first property follows from the fact that for any data-sufficient configuration $\langle \bar{\mathbb{B}}; \mathbb{V} \rangle$, we have $\langle \bar{\mathbb{B}}; \mathbb{V} \rangle \blacktriangleright \langle \top; \emptyset \rangle$. The second property follows from Prop. 2.22.* □

Finally, if a specific state does not follow in linear logic, it is guaranteed not to follow in the operational semantics:

Property 5.16 Safety Properties. *Consider a program $\mathbb{P}$, a constraint theory CT, a flat state S and a configuration $\bar{T}$. If $\bar{T} \notin \mathcal{L}^C_{\mathbb{P},CT}(S)$ then $\bar{T} \notin C_{\mathbb{P},CT}(S)$.*

Proof sketch. *This follows from the fact that $C_{\mathbb{P},CT}(S) \subset \mathcal{L}^C_{\mathbb{P},CT}(S)$.* □

Example 5.17. In this example, we implement a recursive descent parser for a context-free grammar. This is a typical CHR$^\vee$ program as it makes use of both don't-know non-determinism and multiple heads. Consider the following simple grammar whose start symbol and only non-terminal is S, and a, f are terminal symbols:

$$\begin{aligned} S &\rightarrow fSS \\ S &\rightarrow a \end{aligned}$$

For our recursive descent parser, we wrap the stack into a constraint **st** and the input into a constraint **inp**. The parser for our grammar looks as follows:

$$\mathbb{P}_{parse} = \left\{ \begin{array}{llll} r_S & @ & \mathbf{st}([S|rs]) & \Leftrightarrow \mathbf{st}([f,S,S|rs]) \vee \mathbf{st}([a|rs]) \\ r_{pop-a} & @ & \mathbf{st}([a|rs]) \wedge \mathbf{inp}([a|ri]) & \Leftrightarrow \mathbf{st}([rs]) \wedge \mathbf{inp}([ri]) \\ r_{pop-f} & @ & \mathbf{st}([f|rs]) \wedge \mathbf{inp}([f|ri]) & \Leftrightarrow \mathbf{st}([rs]) \wedge \mathbf{inp}([ri]) \\ r_{fail-f} & @ & \mathbf{st}([a|rs]) \wedge \mathbf{inp}([f|ri]) & \Leftrightarrow \bot \\ r_{fail-a} & @ & \mathbf{st}([f|rs]) \wedge \mathbf{inp}([a|ri]) & \Leftrightarrow \bot \\ r_{fail-i} & @ & \mathbf{st}([]) \wedge \mathbf{inp}([i|ri]) & \Leftrightarrow \bot \\ r_{fail-s} & @ & \mathbf{st}([s|rs]) \wedge \mathbf{inp}([]) & \Leftrightarrow \bot \\ r_{acc} & @ & \mathbf{st}([]) \wedge \mathbf{inp}([]) & \Leftrightarrow \mathbf{accept} \end{array} \right\}$$

If called with an initial state $\langle \mathbf{st}([S]), \mathbf{inp}(ri); \emptyset \rangle$ where ri is a word of the language defined by our grammar, it will derive a configuration containing at least one final state $\langle \mathbf{accept}; \emptyset \rangle$ in flat normal form. If ri is not a word of the grammar, it will invariably fail. (This is a very standard technique. Hence we do not need to prove it.)

We assume that we use the axiomatic semantics, i.e. we encode the rules, as well as the constraint and equality theories in a set of axioms Σ. In the following proof, we want to reason about arbitrary inputs. To this end, we add the symbol x to our terminal symbols and add the judgement

$$\mathbf{inp}([x|ri]) \vdash \mathbf{inp}([a|ri]) \oplus \mathbf{inp}([f|ri])$$

to Σ. We shall refer to it as the arbitrarity axiom. We furthermore assume, that for any natural number n and for any symbol σ of our grammar, σ^n is a list $[\sigma, \sigma, \ldots, \sigma]$ of length n and $\sigma^0 = []$ is an empty list. Hence,

$$\bigoplus_{n \in 2\mathbb{N}+1} \mathbf{st}(S^n)$$

is the additive disjunction of all **st** constraints whose argument is a list of $S's$ with odd length. We observe that by rule r_S, we have:

$$\bigoplus_{n \in 2\mathbb{N}+1} \mathbf{st}(S^n) \vdash_\Sigma \bigoplus_{n \in 2\mathbb{N}} \mathbf{st}([a|S^n]) \oplus \mathbf{st}([f|S^n])$$

By adding a multiplicative conjunction with $\mathbf{inp}([a|ri])$ and expanding on the right-hand side we get:

$$\mathbf{inp}([a|ri]) \otimes \bigoplus_{n\in 2\mathbb{N}+1} \mathbf{st}(S^n) \vdash_\Sigma \bigoplus_{n\in 2\mathbb{N}} (\mathbf{inp}([a|ri]) \otimes \mathbf{st}([a|S^n])) \oplus (\mathbf{inp}([a|ri]) \otimes \mathbf{st}([f|S^n]))$$

By rules r_{pop-a} and r_{fail-a} we then have:

$$\mathbf{inp}([a|ri]) \otimes \bigoplus_{n\in 2\mathbb{N}+1} \mathbf{st}(S^n) \vdash_\Sigma \bigoplus_{n\in 2\mathbb{N}} \mathbf{inp}(ri) \otimes \mathbf{st}(S^n)$$

Similarly, we can show for an input beginning with f:

$$\mathbf{inp}([f|ri]) \otimes \bigoplus_{n\in 2\mathbb{N}+1} \mathbf{st}(S^n) \vdash_\Sigma \bigoplus_{n\in 2\mathbb{N}} \mathbf{inp}(ri) \otimes \mathbf{st}([S, S|S^n])$$

Our arbitrarity axiom and factorising $\mathbf{inp}(ri)$ gives us:

$$\mathbf{inp}([x|ri]) \otimes \bigoplus_{n\in 2\mathbb{N}+1} \mathbf{st}(S^n) \vdash_\Sigma \mathbf{inp}(ri) \otimes \bigoplus_{n\in 2\mathbb{N}} (\mathbf{st}(S^n) \oplus \mathbf{st}([S, S|S^n]))$$

Due to the idempotence of $\oplus$, we can merge the disjoint stacks on the right-hand side and get:

$$\mathbf{inp}([x|ri]) \otimes \bigoplus_{n\in 2\mathbb{N}+1} \mathbf{st}(S^n) \vdash_\Sigma \mathbf{inp}(ri) \otimes \bigoplus_{n\in 2\mathbb{N}} \mathbf{st}(S^n)$$

Similarly (but with an additional application of r_{fail-i}) we show:

$$\mathbf{inp}([x|ri]) \otimes \bigoplus_{n\in 2\mathbb{N}} \mathbf{st}(S^n) \vdash_\Sigma \mathbf{inp}(ri) \otimes \bigoplus_{n\in 2\mathbb{N}+1} \mathbf{st}(S^n)$$

So we can finally conclude:

$$\mathbf{inp}(x^{m+2}) \otimes \bigoplus_{n\in 2\mathbb{N}+1} \mathbf{st}(S^n) \vdash_\Sigma \mathbf{inp}(x^m) \otimes \bigoplus_{n\in 2\mathbb{N}+1} \mathbf{st}(S^n)$$

And then:

$$\mathbf{inp}(x^m) \otimes \bigoplus_{n\in 2\mathbb{N}+1} \mathbf{st}(S^n) \vdash_\Sigma \otimes \bigoplus_{n\in 2\mathbb{N}+1} \left(\mathbf{inp}(x^{m \bmod 2}) \otimes \mathbf{st}(S^n)\right)$$

For $m \bmod 2 = 0$ and any $n > 0$, $\mathbf{inp}(x^{m \bmod 2}) \otimes \mathbf{st}(S^n)$ proves $\mathbf{0}$ by rule r_{fail-s}:

$$\mathbf{inp}(x^{m \bmod 2}) \otimes \bigoplus_{n\in 2\mathbb{N}+1} \mathbf{st}(S^n) \vdash_\Sigma \mathbf{0}$$

We now recall either inference rule ($R\oplus_1$) or ($R\oplus_2$) to remind us that:

$$\mathbf{st}([S]) \vdash_\Sigma \bigoplus_{n\in 2\mathbb{N}+1} \mathbf{st}(S^n)$$

Hence, for every even natural number m, we have:

$$\mathbf{inp}(x^m) \otimes \mathbf{st}([S]) \vdash_\Sigma \mathbf{0}$$

By Property 5.14, we then have for every even natural number m:

$$\langle \mathbf{inp}(x^m) \otimes \mathbf{st}([S]); \emptyset \rangle \mapsto^* \varepsilon$$

That is to say: Every input word whose length is an even number has a failed answer. As $\mathbb{P}_{parse}$ is furthermore confluent, it will invariably fail. Similarly, if we change the arbitrarity axiom to

$$\mathbf{inp}([x|ri]) \vdash \mathbf{inp}([a|ri])\&\mathbf{inp}([f|ri])$$

we can show that for every odd m there *exists* an input of length m for which the computation invariably fails.

5.3 Comparison of Programs

In this section, we concentrate on the comparison of CHR$^\vee$ with respect to program equivalence. We define three notions of operational equivalence, each one corresponding to one set of observables as introduced in Section 2.2.

Definition 5.18 *Operational Equivalence*. (1) Two CHR$^\vee$ programs $\mathbb{P}_1, \mathbb{P}_2$ are *operationally C-equivalent* under a given constraint theory CT if for any state S, we have $C_{\mathbb{P}_1,CT}(S) = C_{\mathbb{P}_2,CT}(S)$.

(2) Two CHR programs $\mathbb{P}_1, \mathbb{P}_2$ are *operationally $\mathcal{A}$-equivalent* under a given constraint theory CT if for any state S, we have $\mathcal{A}_{\mathbb{P}_1,CT}(S) = \mathcal{A}_{\mathbb{P}_2,CT}(S)$.

(3) Two CHR$^\vee$ programs $\mathbb{P}_1, \mathbb{P}_2$ are *operationally $\mathcal{S}$-equivalent* under a given constraint theory CT if for any state S, we have $\mathcal{S}_{\mathbb{P}_1,CT}(S) = \mathcal{S}_{\mathbb{P}_2,CT}(S)$.

(4) Two CHR$^\vee$ programs $\mathbb{P}_1, \mathbb{P}_2$ are *operationally $\mathcal{S}(\mathbb{S})$-equivalent* under a given constraint theory CT for a class of flat initial states $\mathbb{S}$ if for any state $S \in \mathbb{S}$, we have $\mathcal{S}_{\mathbb{P}_1,CT}(S) = \mathcal{S}_{\mathbb{P}_2,CT}(S)$.

We will mainly focus on C-eqivalence and $\mathcal{S}$-equivalence. What we call $\mathcal{A}$-equivalence has been researched extensively in the past (cf. Abdennadher et al. [1999]). It shows in this section that the linear-logic semantics is not adequate to reason about $\mathcal{A}$-equivalence.

Definition 5.19 *Logical Equivalence of Programs*. Two CHR programs $\mathbb{P}_1, \mathbb{P}_2$ are called *logically equivalent* under a given constraint theory CT if $ET^L, CT^L \vdash \bigotimes \mathbb{P}_1^L \multimap \bigotimes \mathbb{P}_2^L$, where the unary operator $\bigotimes$ stands for element-wise multiplicative conjunction and $\bigotimes \mathbb{P}_1^L \multimap \bigotimes \mathbb{P}_2^L$ is shorthand for $(\bigotimes \mathbb{P}_1^L \multimap \bigotimes \mathbb{P}_2^L)\&(\bigotimes \mathbb{P}_2^L \multimap \bigotimes \mathbb{P}_1^L)$.

The following proposition relates C- and $\mathcal{S}$-equivalence.

Proposition 5.20. *Operational $\mathcal{S}$-equivalence is a necessary but not a sufficient condition for C-equivalence.*

Proof. *To show that $\mathcal{S}$-equivalence is a necessary condition, we assume two C-equivalent programs $\mathbb{P}_1, \mathbb{P}_2$. For every state S, we have $C_{\mathbb{P}_1}(S) = C_{\mathbb{P}_2}(S)$. As each $\mathcal{S}_{\mathbb{P}_i}$ is the projection of $C_{\mathbb{P}_i}(S)$ to configurations with empty user-defined stores, we also have $\mathcal{S}_{\mathbb{P}_1}(S) = \mathcal{S}_{\mathbb{P}_2}(S)$.*

To show that $\mathcal{S}$-equivalence is not a sufficient condition, consider the following two programs:

$$\mathbb{P}_1 = \left\{ \begin{array}{lcl} \boldsymbol{a}(x) & \Leftrightarrow & \boldsymbol{b}(x) \\ \boldsymbol{b}(x) & \Leftrightarrow & x \doteq 0 \end{array} \right\} \qquad \mathbb{P}_2 = \left\{ \begin{array}{lcl} \boldsymbol{a}(x) & \Leftrightarrow & x \doteq 0 \\ \boldsymbol{b}(x) & \Leftrightarrow & x \doteq 0 \end{array} \right\}$$

Both programs ultimately map every $\boldsymbol{a}(x)$ and $\boldsymbol{b}(x)$ to $x \doteq 0$. Hence, they are $\mathcal{S}$-equivalent. For $S = \langle \boldsymbol{a}(x); \emptyset \rangle$ and $T = \langle \boldsymbol{b}(x); \emptyset \rangle$ we have $[T] \in C_{\mathbb{P}_1}(S)$ but $[T] \notin C_{\mathbb{P}_2}(S)$. Hence, the programs are not C-equivalent. □

We can show that operational C-equivalence implies logical equivalence of programs:

Proposition 5.21. *Let $\mathbb{P}_1, \mathbb{P}_2$ be two C-equivalent CHR$^\vee$ programs under CT. Then $CT^L \vdash \bigotimes \mathbb{P}_1 \multimap\!\!\circ \bigotimes \mathbb{P}_2$.*

Proof. *Since $\mathbb{P}_1$ and $\mathbb{P}_2$ are C-equivalent, we have that $C_{\mathbb{P}_1}(S) = C_{\mathbb{P}_2}(S)$ for all S. For every rule $R = (r \ @ \ H_1 \setminus H_2 \Leftrightarrow G \mid B) \in \mathbb{P}_2$, we have by Def. 2.18: $[\langle H_1 \wedge B \wedge G; \bar{x}\rangle] \in C_{\mathbb{P}_2}(\langle H_1 \wedge H_2 \wedge G; \bar{x}\rangle)$ where $\bar{x} = vars(H_1 \wedge H_2 \wedge G)$ and then by our hypothesis $[\langle H_1 \wedge B \wedge G; \bar{x}\rangle] \in C_{\mathbb{P}_1}(\langle H_1 \wedge H_2 \wedge G; \bar{x}\rangle)$. Therefore, we get $CT^L \vdash \bigotimes \mathbb{P}_1^L \multimap R^L$. Applying this to all rules $R \in \mathbb{P}_2$, we show $CT^L \vdash \bigotimes \mathbb{P}_1^L \multimap \bigotimes \mathbb{P}_2^L$. Analogously, we get $CT^L \vdash \bigotimes \mathbb{P}_2^L \multimap \bigotimes \mathbb{P}_1^L$.* □

The reverse direction does not hold in general as the following example shows:

Example 5.22. Let the constraint theory CT contain at least the theory of natural numbers. Compare the following two programs:

$$\mathbb{P}_1 = \left\{ \mathbf{c}(x) \ \Leftrightarrow \ x \geq 1 \right\} \qquad \mathbb{P}_2 = \left\{ \begin{array}{l} \mathbf{c}(x) \ \Leftrightarrow \ \top \\ \mathbf{c}(x) \ \Leftrightarrow \ x \geq 1 \end{array} \right\}$$

The greater-or-equal constraint $\geq$ is a built-in constraint. Hence, it is translated as $(x \geq 1)^L = !(x \geq 1)$. As $!(x \geq 1) \vdash 1$, we have $\bigotimes \mathbb{P}_1^L \dashv\vdash_\Sigma \bigotimes \mathbb{P}_2^L$. We observe that $\mathcal{S}_{\mathbb{P}_1}(\langle \mathbf{c}(x); x\rangle) = \{\langle x \geq 1; x\rangle\}$ and $\mathcal{S}_{\mathbb{P}_2}([\langle \mathbf{c}(x); \{x\}\rangle]) = \{[\langle x \geq 1; \{x\}\rangle], [\langle \top; \{x\}\rangle]\}$. As the sets are not equal, $\mathbb{P}_1$ and $\mathbb{P}_2$ are not operationally $\mathcal{S}$-equivalent and hence, by Prop. 5.20, not C-equivalent.

However, if we restrict ourselves to analytic, confluent programs, we can show that logical equivalence of programs implies operational $\mathcal{S}$-equivalence:

Proposition 5.23. *(1) Let $\mathbb{P}_1, \mathbb{P}_2$ be two confluent CHR$^\vee$ programs that are analytic for a class of initial states $\mathbb{S}$ and where $CT^L \vdash \bigotimes \mathbb{P}_1^L \multimap\!\!\circ \bigotimes \mathbb{P}_2^L$. Then $\mathbb{P}_1, \mathbb{P}_2$ are $\mathcal{S}(\mathbb{S})$-equivalent.*

(2) If $\mathbb{P}_1, \mathbb{P}_2$ are furthermore generally analytic, they are $\mathcal{S}$-equivalent.

Proof. *It will suffice to prove the first property as the second is its straightforward extension to arbitrary initial states: As both $\mathbb{P}_1$ and $\mathbb{P}_2$ are confluent, we have $|\mathcal{S}_{\mathbb{P}_i,CT}(S)| \in \{0, 1\}$ for any state S and $i \in \{1, 2\}$, where $|\cdot|$ denotes cardinality. If $|\mathcal{S}_{\mathbb{P}_i,CT}(S)| = 0$ then $|\blacktriangledown\mathcal{S}_{\mathbb{P}_i,CT}(S)| = 0$. Otherwise, $|\blacktriangledown\mathcal{S}_{\mathbb{P}_i,CT}| \geq 1$. In the former case, our proposition is trivially true since $\mathcal{S}_{\mathbb{P}_i,CT} = \emptyset$. In the following, we assume $|\mathcal{S}_{\mathbb{P}_i,CT}| = 1$.*

Logical equivalence implies that $\mathcal{L}^C_{\mathbb{P}_1,CT}(S) = \mathcal{L}^C_{\mathbb{P}_2,CT}(S)$ for all S. Since $\mathcal{L}^S$ is the projection of $\mathcal{L}^C$ to configurations with empty user-defined stores, we also have $\mathcal{L}^S_{\mathbb{P}_1,CT}(S) = \mathcal{L}^S_{\mathbb{P}_2,CT}(S)$ and hence $\triangledown\mathcal{S}_{\mathbb{P}_1,CT}(S) = \triangledown\mathcal{S}_{\mathbb{P}_2,CT}(S)$.

Since $|\mathcal{S}_{\mathbb{P}_i,CT}(S)| = 1$ for $i \in \{1, 2\}$, each lower closure $\triangledown\mathcal{S}_{\mathbb{P}_i,CT}(S)$ has a maximum $[\bar{M}_i] \in \triangledown\mathcal{S}_{\mathbb{P}_i,CT}(S)$ such that $\forall[\bar{S}] \in \triangledown\mathcal{S}_{\mathbb{P}_i,CT}(S).[\bar{M}_i] \rhd [\bar{S}]$ and $\mathcal{S}_{\mathbb{P}_i,CT}(S) = \{[M_i]\}$. As $\blacktriangledown\mathcal{S}_{\mathbb{P}_1,CT}(S) = \blacktriangledown\mathcal{S}_{\mathbb{P}_2,CT}(S)$, we have $\bar{M}_1 \blacktriangleleft\!\blacktriangleright \bar{M}_2$. As both programs are analytic in $\mathbb{S}$, we furthermore have that $\bar{M}_1, \bar{M}_2$ are compact. Hence, we have $\bar{M}_1 \equiv_\vee \bar{M}_2$ and therefore: $\mathcal{S}_{\mathbb{P}_1,CT}(S) = \mathcal{S}_{\mathbb{P}_2,CT}(S)$. □

The following example shows that logical equivalence does not imply operational $\mathcal{A}$-equivalence:

Example 5.24. We consider the program $\mathbb{P} = \{\mathbf{c}(x) \Leftrightarrow \mathbf{c}(x)\}$ and the empty program $\mathbb{P}_\emptyset = \emptyset$:

As the logical reading $\mathbb{P}^L = !\forall(\mathbf{c}(x) \multimap \mathbf{c}(x))$ of $\mathbb{P}$ is a logical tautology, it follows that $\mathbb{P}^L \dashv\vdash_\Sigma \mathbb{P}^L_\emptyset$ for any Σ. Yet, for $S = \langle \mathbf{c}(x); \top; \emptyset \rangle$, we have $\mathcal{A}_\mathbb{P}(S) = \emptyset$ whereas $\mathcal{A}_{\mathbb{P}_\emptyset}(S) = \{[S]\}$. Therefore $\mathcal{A}_\mathbb{P}(S) \neq \mathcal{A}_{\mathbb{P}_\emptyset}(S)$.

The following final example shows how we can apply the linear-logic semantics to compare Horn programs with committed-choice CHR programs.

Example 5.25. Automatic generation of CHR solvers from Horn programs is a topic of ongoing research [Abdennadher and Rigotti 2005; Sobhi et al. 2008]. In Sobhi et al. [2008], a generation method based on the classical declarative semantics was proposed.

As CHR$^\vee$ embeds both CHR and Horn programs, the linear logic semantics carries over to these formalisms as well. Hence, it should be a promising approach to investigate automatic generation of rule-based solver on the basis of this semantics. In this example, we show that the linear logic semantics allows us to compare the operational semantics of Horn programs and CHR programs on a very fine-grained level.

We begin with the following example [Abdennadher and Schütz 1998] of a Horn program embedded in CHR$^\vee$. It implements a ternary *append* predicate for lists, where the third argument is the concatenation of the first two:

$$\mathbb{H} = \left\{ \begin{array}{l} \mathbf{append}(x,y,z) \leftarrow x \doteq [\,] \wedge y \doteq z \\ \mathbf{append}(x,y,z) \leftarrow x \doteq [h|l_1] \wedge z \doteq [h|l_2] \wedge \mathbf{append}(l_1,y,l_2) \end{array} \right\}$$

We embed this program into CHR$^\vee$ according to Definition 2.23:

$$\mathbb{P}_1 = \left\{ \begin{array}{l} \mathbf{append}(x,y,z) \Leftrightarrow (x \doteq [\,] \wedge y \doteq z) \vee \\ \qquad\qquad\qquad\quad x \doteq [h|l_1] \wedge z \doteq [h|l_2] \wedge \mathbf{append}(l_1,y,l_2) \end{array} \right\}$$

The linear-logic reading of the embedded program looks as follows:

$$\mathbb{P}^L_1 = \left\{ \begin{array}{l} !\forall x,y,z.(\mathbf{append}(x,y,z) \multimap \exists l_1,l_2,h.(!x \doteq [\,] \otimes !y \doteq z) \oplus \\ \qquad\qquad\qquad\qquad\qquad\quad (!x \doteq [h|l_1] \otimes !z \doteq [h|l_2] \otimes \mathbf{append}(l_1,y,l_3))) \end{array} \right\}$$

Secondly, we write a program to implement the *append* predicate the way it would be expected in CHR:

$$\mathbb{P}_2 = \left\{ \begin{array}{lll} \mathbf{append}([\,],y,z) & \Leftrightarrow & y \doteq z \\ \mathbf{append}([h|l_1],y,z) & \Leftrightarrow & z \doteq [h|l_2] \wedge \mathbf{append}(l_1,y,l_2) \end{array} \right\}$$

The two programs are not *per se* $\mathcal{S}$-equivalent. Consider their behaviour in case the first argument of **append** is bound to anything else than a list. For $S_0 = \langle \mathbf{append}(3,x,y); \emptyset \rangle$, we have $\mathcal{S}_{\mathbb{P}_1}(S_0) = \{S_\perp\}$ but $\mathcal{S}_{\mathbb{P}_2}(S_0) = \emptyset$.

Now let us assume that the first argument is always bound to a list. We can model this by the following formula:

$$\varphi = \forall(\mathbf{append}(x,y,z) \multimap \mathbf{append}(x,y,z) \otimes (!x \doteq [\,] \oplus \exists h,l.!x \doteq [h|l]))$$

It shows that $CT^L, \varphi \vdash \bigotimes \mathbb{P}_1 \multimap\!\!\circ \bigotimes \mathbb{P}_2$. Hence, under the assumption that the first argument is always bound to a (non-empty or empty) list, the two programs are operationally $\mathcal{S}$-equivalent.

Moreover, we observe that φ is equivalent to the logical reading of the CHR$^\vee$ rule R_φ:

$$R_\varphi = (r \mathbin{@} \mathbf{append}(x,y,z) \Leftrightarrow \mathbf{append}(x,y,z) \wedge (x \doteq [\,] \vee x \doteq [h|l]))$$

Moreover $CT^L, \varphi \vdash \bigotimes \mathbb{P}_1 \multimap \bigotimes \mathbb{P}_2$ implies that $CT^L \vdash (\bigotimes \mathbb{P}_1 \otimes \varphi) \multimap (\bigotimes \mathbb{P}_2 \otimes \varphi)$. Hence, the programs $\mathbb{P}'_1 = \mathbb{P}_1 \cup R_\varphi$ and $\mathbb{P}'_2 = \mathbb{P}_2 \cup R_\varphi$ are operationally $\mathcal{S}$-equivalent (without any further assumptions).

6. RELATED WORK

From its advent in the 1980s, linear logic has been studied in relationship with programming languages. Common linear logic programming languages such as LO [Andreoli and Pareschi 1990], Lolli [Hodas and Miller 1991], LinLog [Andreoli 1992], and Lygon [James Harland and David J. Pym and Michael Winikoff 1996] rely on generalizations of backward-chaining backtracking resolution of Horn clauses.

The earliest approach at defining a linear-logic semantics for a committed-choice programming language that we are aware of has been proposed in Zlatuska [1993]. The corresponding language is indeed a fragment of pure CHR without multiple heads and with substantial restrictions on the use of built-in constraints. The first approach to a linear logic semantics for CHR was published in Betz [2004] and shortly after – though independently – in Bouissou [2005]. Both approaches correspond to the encoding semantics presented in Sect. 4.5, although the latter presents only translations for rules. An alternative approach has been investigated by Meister et al. [2007], using transaction logic and mapping CHR states to databases. Their approach is restricted to the range-restricted ground segment of CHR as non-ground states do not map naturally do databases.

The linear-logic programming language LolliMon, proposed in López et al. [2005], integrates backward-chaining proof search with committed-choice forward reasoning. It is an extension of the aforementioned language Lolli. The sequent calculus underlying Lolli extended by a set of dedicated inference rules. The corresponding connectives are syntactically detached from Lolli's own connectives and operationally they are processed within a monad. The actual committed-choice behaviour comes by the explicit statement in the operational semantics, that these inference are to be applied in a committed-choice manner during proof search. With respect to Lolli, committed choice thus comes at the cost of giving up the general notion of execution as proof search, although it is retained outside the monad.

The class LCC of linear logic concurrent constraint programming languages [Fages et al. 2001] has a close relationship with CHR, although the former is based on agents whereas the latter is based on rules. Similar to CHR, LCC languages are non-deterministic and execution is committed-choice. For the case of pure CHR, our linear-logic semantics can be considered a straightforward extension of LCC. Unlike $\text{CHR}^\vee$ however, LCC has no notion of disjunction.

Furthermore, Fages et al. [2001] have proposed the so-called *frontier semantics* for LCC, in which the committed-choice operator is interpreted analogously to the disjunction operator $\vee$ in $\text{CHR}^\vee$. In the linear-logic interpretation of the frontier semantics, it is correspondingly mapped to the multiplicative disjunction $\oplus$. However, the frontier semantics does not constitute a distinct programming language but is viewed as a tool to reason about properties of LCC programs. Hence, committed choice never co-exists with disjunction as in the linear logic semantics for $\text{CHR}^\vee$. Rather, the two are viewed as different interpretations of the same connective for different purposes.

More recently, Simmons and Pfenning [2008] proposed the linear logic-based committed-choice programming language *Linear Logical Algorithms* (LLA). LLA distin-

guishes between linear and persistent propositions. That distinction bears similarities to the distinction between user-defined and built-in constraints in CHR. However, LLA has no direct counterpart to the constraint theory *CT* in CHR. Rather, both types of constraints are handled by user-programs.

The segment of LLA without persistent propositions corresponds to the segment of CHR without built-in constraints and where only ground constraints occur. For this, rather restricted segment, programs can be directly translated between the two languages. In these cases, the linear-logic readings of rules in LLA coincides with the linear-logic semantics of CHR, i.e. their respective logical readings are logically equivalent.

7. CONCLUSION

In this article, we have presented a detailed analysis of the relationship between linear logic and CHR$^\vee$ (and thus also CHR) and we have shown its applications from reasoning about program observables to deciding operational equivalence of multi-paradigm CHR$^\vee$ programs.

Our first contribution is the definition of a sound and complete linear-logic semantics for CHR$^\vee$. This semantics maps the dualism between don't-care and don't-know nondeterminism in CHR$^\vee$ to the dualism of internal and external choice in linear logic. Furthermore, we have defined a notion of configuration entailment to characterize the discrepance between state transition and logical judgement. It is a key notion for the study and the application of our semantics.

We have shown that in the full segment of CHR$^\vee$ logical equivalence does not necessarily coincide with configuration equivalence. This makes linear-logic based reasoning over CHR$^\vee$ in general imprecise. However, we have presented a well-behavedness property for CHR$^\vee$ – analyticness – that allows us to identify a segment for which this restriction does not apply. This segment includes pure CHR, which does not contain disjunction.

As our second main contribution, we have shown how to apply our results to reason about CHR$^\vee$ programs. We have defined sets of linear-logic based observables that correspond with the usual program observables of computable state and data-sufficient answer by means of state entailment or confguration entailment, respectively. We have presented criteria to prove various program properties, foremost safety properties, which consist in the non-computability of a specific state from a certain initial state. Furthermore, we have given a criterion to prove operational equivalence with respect to data-sufficient answers for multi-paradigm programs.

As a further contribution, we have for the first time defined a formalization of the operational semantics of CHR$^\vee$ that is based on equivalence classes of configurations. It is an extension of the equivalence-based semantics for CHR, publiched in [Raiser et al. 2009].

Our results entail a wide range of possible future work. An obvious line of future work lies in the application of established methods for automated proof search in linear logic to reason about CHR$^\vee$ programs. As significant effort has been put in the current result on amending the discrepance between linear judgement and the semantics of CHR$^\vee$, it furthermore suggests itself to investigate whether a "purer" formalism to reason about CHR could be extracted from linear logic that avoids these discrepancies. As suggested by one of the anonymous reviewers of this paper, another important point for future work is a comprehensive investigation of the applicability of our results in the context of varying operational semantics.

ACKNOWLEDGMENTS

We are grateful to the reviewers of an earlier version of this paper for their helpful remarks. Hariolf Betz has been funded by Baden-Württemberg federal state grant LGFG #0518.

REFERENCES

Abdennadher, S. 1997. Operational Semantics and Confluence of Constraint Propagation Rules. In *CP*, G. Smolka, Ed. Lecture Notes in Computer Science, vol. 1330. Springer, 252–266.

Abdennadher, S., Fruhwirth, T., and Meuss, H. 1996. On Confluence of Constraint Handling Rules. In *CP'96, LNCS 1118*. Springer-Verlag, 1–15.

Abdennadher, S., Frühwirth, T. W., and Meuss, H. 1999. Confluence and Semantics of Constraint Simplification Rules. *Constraints 4,* 2, 133–165.

Abdennadher, S. and Rigotti, C. 2005. Automatic Generation of CHR Constraint Solvers. *THEORY AND PRACTICE OF LOGIC PROGRAMMING 5,* 4, 403–418.

Abdennadher, S. and Schütz, H. 1998. CHRv: A Flexible Query Language. In *FQAS*, T. Andreasen, H. Christiansen, and H. L. Larsen, Eds. Lecture Notes in Computer Science, vol. 1495. Springer, 1–14.

Andreoli, J.-M. 1992. Logic Programming with Focusing Proofs in Linear Logic. *Journal of Logic and Computation 2*, 297–347.

Andreoli, J.-M. and Pareschi, R. 1990. LO and Behold! Concurrent Structured Processes. In *OOPSLA/ECOOP*. 44–56.

Betz, H. 2004. A Linear Logic Semantics for Constraint Handling Rules. M.S. thesis, University of Ulm.

Betz, H. 2007. A Linear Logic Semantics for Constraint Handling Rules with Disjunction. In *Proceedings of the 4th Workshop on Constraint Handling Rules*. 17–31.

Betz, H. and Frühwirth, T. W. 2005. A Linear-Logic Semantics for Constraint Handling Rules. In *CP*, P. van Beek, Ed. Lecture Notes in Computer Science, vol. 3709. Springer, 137–151.

Betz, H., Raiser, F., and Frühwirth, T. W. 2010. A Complete and Terminating Execution Model for Constraint Handling Rules. *TPLP 10,* 4-6, 597–610.

Bouissou, O. 2005. A CHR Library for SiLCC. M.S. thesis, Technische Universität Berlin.

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 *ICLP*, B. Demoen and V. Lifschitz, Eds. Lecture Notes in Computer Science, vol. 3132. Springer, 90–104.

Fages, F., Ruet, P., and Soliman, S. 1997. Linear Concurrent Constraint Programming: Operational and Phase Semantics.

Fages, F., Ruet, P., and Soliman, S. 2001. Linear Concurrent Constraint Programming: Operational and Phase Semantics. *Inf. Comput. 165,* 1, 14–41.

Frühwirth, T. 2009. *Constraint Handling Rules*. Cambridge University Press.

Frühwirth, T. and Abdennadher, S. 2003. *Essentials of Constraint Programming*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Frühwirth, T. W. 1994. Constraint Handling Rules. In *Constraint Programming*, A. Podelski, Ed. Lecture Notes in Computer Science, vol. 910. Springer, 90–107.

Frühwirth, T. W. 1998. Theory and Practice of Constraint Handling Rules. *J. Log. Program. 37,* 1-3, 95–138.

Frühwirth, T. W., Pierro, A. D., and Wiklicky, H. 2002. Probabilistic Constraint Handling Rules. *Electr. Notes Theor. Comput. Sci. 76*.

Girard, J.-Y. 1987. Linear Logic. *Theor. Comput. Sci. 50*, 1–102.

Haemmerlé, R. and Betz, H. 2008. Verification of Constraint Handling Rules using Linear Logic Phase Semantics. In *Proceedings of the 5th Workshop on Constraint Handling Rules: CHR 2008*.

Hodas, J. S. and Miller, D. 1991. Logic Programming in a Fragment of Intuitionistic Linear Logic. In *LICS*. IEEE Computer Society, 32–42.

James Harland and David J. Pym and Michael Winikoff. 1996. Programming in lygon: An overview. In *AMAST*, M. Wirsing and M. Nivat, Eds. Lecture Notes in Computer Science, vol. 1101. Springer, 391–405.

López, P., Pfenning, F., Polakow, J., and Watkins, K. 2005. Monadic Concurrent Linear Logic Programming. In *PPDP*, P. Barahona and A. P. Felty, Eds. ACM, 35–46.

Meister, M., Djelloul, K., and Robin, J. 2007. A Unified Semantics for Constraint Handling Rules in Transaction Logic. In *LPNMR*, C. Baral, G. Brewka, and J. S. Schlipf, Eds. Lecture Notes in Computer Science, vol. 4483. Springer, 201–213.

Miller, D. 1992. The pi-Calculus as a Theory in Linear Logic: Preliminary Results. In *ELP*, E. Lamma and P. Mello, Eds. Lecture Notes in Computer Science, vol. 660. Springer, 242–264.

Negri, S. 1995. Semantical Observations on the Embedding of Intuitionistic Logic into Intuitionistic Linear Logic. *Mathematical Structures in Computer Science 5,* 1, 41–68.

Raiser, F., Betz, H., and Frühwirth, T. 2009. Equivalence of CHR States Revisited. In *6th International Workshop on Constraint Handling Rules (CHR)*, F. Raiser and J. Sneyers, Eds. 34–48.

Simmons, R. J. and Pfenning, F. 2008. Linear Logical Algorithms. In *ICALP (2)*, L. Aceto, I. Damgård, L. A. Goldberg, M. M. Halldórsson, A. Ingólfsdóttir, and I. Walukiewicz, Eds. Lecture Notes in Computer Science, vol. 5126. Springer, 336–347.

Sneyers, J., Schrijvers, T., and Demoen, B. 2009. The Computational Power and Complexity of Constraint Handling Rules. *ACM TOPLAS*.

Sobhi, I., Abdennadher, S., and Betz, H. 2008. Constructing Rule-Based Solvers for Intentionally-Defined Constraints. In *Constraint Handling Rules*, T. Schrijvers and T. W. Frühwirth, Eds. Lecture Notes in Computer Science, vol. 5388. Springer, 70–84.

Zlatuska, J. 1993. Committed-Choice Concurrent Logic Programming in Linear Logic. In *Kurt Gödel Colloquium*, G. Gottlob, A. Leitsch, and D. Mundici, Eds. Lecture Notes in Computer Science, vol. 713. Springer, 337–348.

A. LEMMATA AND PROOF DEVICES

In this appendix, we present definitions and lemmas that will be of lesser interest for the reader but are important for the proofs in Appendix B.

The notion of merging of states is an important tool for the proof of Theorem 4.14. We define it as follows:

Definition A.1 *($\cdot \diamond \cdot$).* Let $S = \langle \mathbb{G}; \mathbb{V} \rangle, S' = \langle \mathbb{G}'; \mathbb{V} \rangle$ be $\text{CHR}^\vee$ states that share the same set of global variables and whose local variables are renamed apart. Their *merging* is defined as:

$$S \diamond S' ::= \langle \mathbb{G} \wedge \mathbb{G}'; \mathbb{V} \rangle$$

The following property assures that we can without loss of generality assume the existence of $S \diamond T$ for any two states S, T:

Property A.2. *For any $CHR^\vee$ states S, T, there exist states S', T' where $S \equiv S', T \equiv T'$ such that $S' \diamond T'$ exists.*

Proof sketch. *Lemma 2.8.1 allows to rename the local variables apart, and Def. 2.6.4 allows the union of their respective sets of global variables.* □

Lemma A.3 states two properties of merging that will be used in upcoming proofs:

Lemma A.3 Properties of $\cdot \diamond \cdot$. *Let S, S', T be CHR states such that both $S \diamond T$ and $S' \diamond T$ exist. The following properties hold:*

(1) $S \rhd S' \quad \Rightarrow \quad S \diamond T \rhd S' \diamond T$

(2) $S \mapsto^r S' \quad \Rightarrow \quad S \diamond T \mapsto^r S' \diamond T$

Proof. *Lemma A.3.1: We assume w.l.o.g. that the states S, S', T share the same set of global variables. Let $S = \langle \mathbb{U} \wedge \mathbb{B}; \mathbb{V} \rangle, S' = \langle \mathbb{U}' \wedge \mathbb{B}'; \mathbb{V} \rangle, T = \langle \mathbb{U}_T \wedge \mathbb{B}_T; \mathbb{V} \rangle$ with local vars $\bar{l}, \bar{l}', \bar{l}_T$. From $S \rhd S'$ follows by Thm. 2.12: $CT \models \forall(\mathbb{B} \rightarrow \exists \bar{l}'.((\mathbb{U} \doteq \mathbb{U}') \wedge \mathbb{B}'))$. As $\mathbb{U}_T \doteq \mathbb{U}_T = \top$, we get $CT \models \forall(\mathbb{B} \wedge \mathbb{B}_T \rightarrow \exists \bar{l}'.\exists \bar{l}_T.((\mathbb{U} \doteq \mathbb{U}') \wedge (\mathbb{U}_T \doteq \mathbb{U}_T) \wedge \mathbb{B}' \wedge \mathbb{B}_T))$ which proves $S \diamond T \rhd S' \diamond T$.*

Lemma A.3.2: We assume w.l.o.g. that the states S, S', T share the same set of global variables. According to Def. 2.18, there exists a variant of a CHR rule r @ $H_1 \setminus H_2 \Leftrightarrow G \mid B_u \wedge B_b$, such that $S \equiv \langle H_1 \wedge H_2 \wedge \mathbb{U}; G \wedge \mathbb{B}; \mathbb{V} \rangle$ and $S' \equiv \langle H_1 \wedge B_c \wedge \mathbb{U} \wedge G \wedge B_b \wedge \mathbb{B}; \mathbb{V} \rangle$. By Prop. A.2, there exists a state $T' = \langle \mathbb{U}' \wedge \mathbb{B}'; \mathbb{V} \rangle$ such that $T' \equiv T$ whose local variables are renamed apart from those of S and T. By Def. 2.18, we get $S \diamond T \mapsto^r S' \diamond T$. □

Lemma A.4. *Let π be a cut-reduced proof of a sequent $\bar{S}^L \vdash \bar{T}^L$ where $\bar{S}, \bar{T}$ are arbitrary configurations. Any formula α in π is either of the form $\alpha = \bar{S}^L_\alpha$ or of the form $\alpha = c_b(\bar{t})$ where $\bar{S}_\alpha$ is a configuration and $c_b(\bar{t})$ is an atomic built-in constraint.*

Proof sketch. *We firstly observe that both the root of π and all proper axioms in Σ are of the form $\bar{U}^L_1 \vdash \bar{U}^L_2$ where $\bar{U}_1, \bar{U}_2$ are configurations. The subformula property hence guarantees that every formula α in π is a subformula of the logical reading $\bar{U}^L$ of some configuration $\bar{U}$.*

We secondly observe that an atomic subformula of $\bar{U}^L$ is either an atomic built-in constraint $c_b(\bar{t})$ or an atomic user-defined constraint $c_u(\bar{t})$. Both cases support our claim since $\langle c_u(\bar{t}); vars(\bar{t}) \rangle^L = c_u(\bar{t})$ and $\langle c_u(\bar{t}); vars(\bar{t}) \rangle$ is a singular configuration. We proof the lemma by induction over the depth of the fomula $\bar{U}^L$. □

It should be noted that the configuration $\bar{S}_\alpha$ is not necessarily unique, i.e. more than one configuration might map to a specific formula. For example, let formula $\alpha = c_u(\bar{t}) \oplus c_u(\bar{t})$. We then have $\langle c_u(\bar{t}) \vee c_u(\bar{t}); vars(c_u(\bar{t}))\rangle^L = (\langle c_u(\bar{t}); vars(c_u(\bar{t}))\rangle \vee \langle c_u(\bar{t}); vars(c_u(\bar{t}))\rangle)^L = \alpha$. However, we have by Def. 2.14.4 that $\bar{S}^L = \bar{T}^L \Rightarrow \bar{S} \blacktriangleright \bar{T}$.

Lemma A.5 establishes an important relationship between state transition and entailment of flat states. We will recur to it to proof Lemma 4.13.

Lemma A.5. *Let S, U, T be CHR states. If $S \rhd U$ and $U \mapsto^r T$ then there exists a state V such that $S \mapsto^r V$ and $V \rhd T$.*

Proof. *Let $S = \langle \mathbb{U}; \mathbb{B}; \mathbb{V}\rangle$ and let $\bar{y}_S, \bar{y}_U, \bar{y}_T$ be the local variables of S, U, T. By definition, $U \mapsto^r T$ implies that there is a variant of a CHR rule r @ $(H_1 \setminus H_2 \Leftrightarrow G \mid B_b \wedge B_u)$ such that $[U] = [\langle H_1 \wedge H_2 \wedge \hat{\mathbb{U}}; G \wedge \hat{\mathbb{B}}; \hat{\mathbb{V}}\rangle]$ and $[T] = [\langle H_1 \wedge B_u \wedge \hat{\mathbb{U}}; G \wedge B_b \wedge \hat{\mathbb{B}}; \hat{\mathbb{V}}\rangle]$.*

Now let $V = \langle H_1 \wedge B_u \wedge \hat{\mathbb{U}}; G \wedge B_b \wedge \hat{\mathbb{B}} \wedge (\mathbb{U} \doteq (H_1 \wedge H_2 \wedge \hat{\mathbb{U}}) \wedge \mathbb{B}; \hat{\mathbb{V}}\rangle$. From $[S] \rhd [U]$ follows by Thm. 4.12: $CT \models \forall(\mathbb{B} \rightarrow \exists \bar{y}_U.((\mathbb{U} \doteq (H_1 \wedge H_2 \wedge \hat{\mathbb{U}})) \wedge G \wedge \hat{\mathbb{B}}))$. Assuming w.l.o.g. that $\bar{y}_S \cap \bar{y}_U = \emptyset$, we can apply Def. 2.6.3 to get $S \equiv \langle \mathbb{U}; \mathbb{B} \wedge G \wedge \hat{\mathbb{B}} \wedge (\mathbb{U} \doteq (H_1 \wedge H_2 \wedge \hat{\mathbb{U}})); \mathbb{V}\rangle$ and then $S \equiv \langle (H_1 \wedge H_2 \wedge \hat{\mathbb{U}}; \mathbb{B} \wedge G \wedge \hat{\mathbb{B}} \wedge (\mathbb{U} \doteq (H_1 \wedge H_2 \wedge \hat{\mathbb{U}})); \mathbb{V}\rangle$. According to Def. 2.18, we have $S \mapsto_r V$. We apply Def. 4.9 to show that $V \rhd T$. □

Lemma A.6 will be used to prove Lemma 4.16:

Lemma A.6 ($\rhd \Rightarrow \vdash$). *For arbitrary singular configurations S, T, entailment $S \rhd T$ implies $S^L \vdash_\Sigma T^L$ for $\Sigma = \Sigma_{CT} \cup \Sigma_{\doteq}$.*

Proof of Lemma A.6. *('⇐') Follows from Thm. 4.14 by assuming an empty program $\mathbb{P} = \emptyset$. ('⇒') We proof that all conditions in Def. 4.9 comply with the judgement relation $\vdash$: For Def. 4.9.1, $CT \models \forall(\mathbb{B} \rightarrow \mathbb{B}')$ implies that Σ_{CT} contains an axiom $\mathbb{B} \vdash \mathbb{B}'$. Hence, we can prove $\exists_{\setminus \mathbb{V}}.\mathbb{U} \wedge \mathbb{B} \vdash \exists_{\setminus \mathbb{V}}.\mathbb{U} \wedge \mathbb{B}'$. For Def. 4.9.2, it is valid since $S^L \vdash \exists x.S^L$ holds for any S^L. Concerning the implicit conditions of a partial order relation, reflexivity and anti-symmetry hold for the judgement relation $\vdash$ as well and anti-symmetry is a natural consequence of Def. 3.4.* □

B. PROOFS

In this appendix, we present several proofs that have been removed from the main paper for more clarity.

Proof of Lemma 4.5. *(1).* We prove that state equivalence $S \equiv_e T$ implies linear judgement $S \vdash_\Sigma T$ by showing that every of the conditions given for $S \equiv_e T$ in Def. 2.6 implies $S \vdash T$:

Def. 2.6.1 implies linear judgement since $\otimes$ is associative, commutative, has the neutral element 1 and distributes over $\oplus$. For Def. 2.6.2, linear judgement is guaranteed, as $\Sigma_{\doteq}$ allow us to prove $\exists_{\setminus \mathbb{V}}.\mathbb{U} \otimes x \doteq t \otimes \mathbb{B} \vdash_\Sigma \exists_{\setminus \mathbb{V}}.\mathbb{U}\,[x/t] \otimes x \doteq t \otimes \mathbb{B}$. For Def. 2.6.3, it is similarly guaranteed by Σ_{CT}. Def. 2.6.4 implies linear judgement since the addition or removal of a global variable not occurring in a state does not change the logical reading of the state. W.r.t. Def. 2.6.5, linear judgement holds since $\varphi \otimes 0 \vdash \psi$ is valid for any φ, ψ. All the above arguments can be shown to apply in the reverse direction as well, thus proving compliance with the implicit symmetry of $\cdot \equiv \cdot$. The implicit reflexivity and transitivity of state equivalence comply with linear judgement due to the (*Identity*) and (*Cut*) rules.

By the symmatry of $\equiv_e$, it follows, that $S \equiv_e T$ also implies $T \vdash_\Sigma S$.

(2). We consider the properties given in Def. 2.14 – Def. 2.14.1: For all α, β, γ, we have $\alpha \oplus \beta \dashv\vdash \beta \oplus \alpha$ and $(\alpha \oplus \beta) \oplus \gamma \dashv\vdash \alpha \oplus (\beta \oplus \gamma)$. Def. 2.14.2: The property follows from (1). Def. 2.14.3: For all α, we have $0 \oplus \alpha \dashv\vdash \alpha$. Def. 2.14.4: For all $\alpha, \beta, \gamma, \mathbb{V}$, we have $(\exists_{\setminus \mathbb{V}}.\alpha \oplus \beta) \oplus \gamma \dashv\vdash (\exists_{\setminus \mathbb{V}}.\alpha) \oplus (\exists_{\setminus \mathbb{V}}.\beta) \oplus \gamma$.

□

Proof of Theorem 4.6. Let $\bar{U}, \bar{V}$ be configurations such that $\bar{U} \mapsto^r \bar{V}$. According to Def. 2.18, there exists a variant of a rule with fresh variables (r @ $H_1 \setminus H_2 \Leftrightarrow G \mid B$) and configurations $\bar{U}' = \langle H_1 \wedge H_2 \wedge G \wedge \mathbb{G}; \mathbb{V}\rangle \vee \bar{T}'$, $\bar{V}' = \langle B_u \wedge H_1 \wedge B_b \wedge G \wedge \mathbb{G}; \mathbb{V}\rangle \vee \bar{T}'$ such that $\bar{U}' \equiv \bar{U}$ and $\bar{V}' \equiv \bar{V}$. Consequently, $\Sigma_{\mathbb{P}}$ contains:

$$H_1^L \otimes H_2^L \otimes G^L \vdash_\Sigma H_1^L \otimes \exists \bar{y}_r.(B^L \otimes G^L)$$

From which we prove:

$$\exists_{\setminus \mathbb{V}}.H_1^L \otimes H_2^L \otimes G^L \otimes \mathbb{G} \vdash_\Sigma \exists_{\setminus \mathbb{V}}.H_1^L \otimes G^L \otimes B^L \otimes \mathbb{G}$$

The local variables $\bar{y}_r$ of r are by Def. 2.18 disjoint from $vars(H_1, H_2, \mathbb{U}, \mathbb{B}, \mathbb{V})$. Hence, we have:

$$(\exists_{\setminus \mathbb{V}}.H_1^L \otimes H_2^L \otimes G^L \otimes \mathbb{G}) \oplus \bar{T}^L \vdash_\Sigma (\exists_{\setminus \mathbb{V}}.H_1^L \otimes G^L \otimes B^L \otimes \mathbb{G}) \oplus \bar{T}^L$$

This corresponds to $\bar{U}'^L \vdash_\Sigma \bar{V}'^L$. Lemma 4.5 then proves that $\bar{U}^L \vdash_\Sigma \bar{V}^L$. As the judgement relation $\vdash_\Sigma$ is transitive and reflexive, the relationship can be generalized to the reflexive-transitive closure $\bar{U} \mapsto^* \bar{V}$. □

Proof of Theorem 4.12. '⇒': We show that the explicit axioms of entailment, as well as the implicit conditions reflexivity, anti-symmetry and transitivity comply with the criterion:

Def. 4.9.1. We assume w.l.o.g. that the strictly local variables of $\langle \mathbb{U}; \mathbb{B}; \mathbb{V}\rangle, \langle \mathbb{U}; \mathbb{B}'; \mathbb{V}\rangle$ are renamed apart. We observe that $(\mathbb{U} \doteq \mathbb{U}) = \top$ is a tautology for any $\mathbb{U}$. Hence, from $CT \models \forall(\exists \bar{s}.\mathbb{B} \rightarrow \exists \bar{s}.\mathbb{B}')$ follows $CT \models \forall(\exists \bar{s}.\mathbb{B} \rightarrow \exists \bar{l}'.(\mathbb{U} \doteq \mathbb{U}) \wedge \mathbb{B}')$, which proves: $CT \models \forall(\mathbb{B} \rightarrow \exists \bar{l}'.((\mathbb{U} \doteq \mathbb{U}) \wedge \mathbb{B}'))$

Def. 4.9.2. Let $\bar{l}$ be the local variables of $\langle \mathbb{U}; \mathbb{B}; \{x\} \cup \mathbb{V}\rangle$. For any x we have: $CT \models \forall(\mathbb{B} \rightarrow \exists x.\exists \bar{l}.((\mathbb{U} \doteq \mathbb{U}) \wedge \mathbb{B}))$

Reflexivity. Let $\langle \mathbb{U}; \mathbb{B}; \mathbb{V}\rangle, \langle \mathbb{U}'; \mathbb{B}'; \mathbb{V}'\rangle$ be CHR states such that $[\langle \mathbb{U}; \mathbb{B}; \mathbb{V}\rangle] = [\langle \mathbb{U}'; \mathbb{B}'; \mathbb{V}'\rangle]$, i.e. $\langle \mathbb{U}; \mathbb{B}; \mathbb{V}\rangle \equiv \langle \mathbb{U}'; \mathbb{B}'; \mathbb{V}'\rangle$. Assuming that the local variables $\bar{l}, \bar{l}'$ have been named apart, Thm. 2.12 implies $CT \models \forall(\mathbb{B} \rightarrow \exists \bar{y}'.((\mathbb{U} \doteq \mathbb{U}') \wedge \mathbb{B}'))$.

Anti-Symmetry. Let $\langle \mathbb{U}; \mathbb{B}; \mathbb{V}\rangle, \langle \mathbb{U}'; \mathbb{B}'; \mathbb{V}'\rangle$ be CHR states with local variables $\bar{l}, \bar{l}'$ such that $CT \models \forall(\mathbb{B} \rightarrow \exists \bar{l}'.((\mathbb{U} \doteq \mathbb{U}') \wedge \mathbb{B}'))$ and $CT \models \forall(\mathbb{B}' \rightarrow \exists \bar{l}.((\mathbb{U} \doteq \mathbb{U}') \wedge \mathbb{B}))$. By Thm. 2.12, we have that $\langle \mathbb{U}; \mathbb{B}; \mathbb{V}\rangle \equiv \langle \mathbb{U}'; \mathbb{B}'; \mathbb{V}'\rangle$ and hence $[\langle \mathbb{U}; \mathbb{B}; \mathbb{V}\rangle] = [\langle \mathbb{U}'; \mathbb{B}'; \mathbb{V}'\rangle]$.

Transitivity. Let $\langle \mathbb{U}; \mathbb{B}; \mathbb{V}\rangle, \langle \mathbb{U}'; \mathbb{B}'; \mathbb{V}'\rangle, \langle \mathbb{U}''; \mathbb{B}''; \mathbb{V}''\rangle$ be CHR states where the local variables $\bar{l}, \bar{l}', \bar{l}''$have been renamed apart and such that $CT \models \forall(\mathbb{B} \rightarrow \exists \bar{l}'.((\mathbb{U} \doteq \mathbb{U}') \wedge \mathbb{B}'))$ and $CT \models \forall(\mathbb{B}' \rightarrow \exists \bar{l}''.((\mathbb{U}' \doteq \mathbb{U}'') \wedge \mathbb{B}''))$. Therefore, $CT \models \forall(\mathbb{B} \rightarrow \exists \bar{y}'.((\mathbb{U} \doteq \mathbb{U}') \wedge \exists \bar{l}''.((\mathbb{U}' \doteq \mathbb{U}'') \wedge \mathbb{B}'')))$. As the sets of local variables are disjoint, we get $CT \models \forall(\mathbb{B} \rightarrow \exists \bar{l}'\bar{l}''.((\mathbb{U} \doteq \mathbb{U}') \wedge (\mathbb{U}' \doteq \mathbb{U}'') \wedge \mathbb{B}''))$ and finally

$$CT \models \forall(\mathbb{B} \rightarrow \exists \bar{l}''.((\mathbb{U} \doteq \mathbb{U}'') \wedge \mathbb{B}''))$$

'⇐': Let $S = \langle \mathbb{U}; \mathbb{B}; \mathbb{V}\rangle, S' = \langle \mathbb{U}'; \mathbb{B}'; \mathbb{V}'\rangle$ be CHR states with local variables $\bar{y}, \bar{y}'$ that have been renamed apart and such that $\mathbb{V}' \subseteq \mathbb{V}$ and $CT \models \forall(\mathbb{B} \rightarrow \exists \bar{y}'.((\mathbb{U} \doteq \mathbb{U}') \wedge \mathbb{B}'))$. We

apply Def. 4.9.1 to infer: $S \rhd \langle \mathbb{U}; (\mathbb{U} \doteq \mathbb{U}') \wedge \mathbb{B}'; \mathbb{V} \rangle$. By Def. 2.6.2 and Def. 2.6.3, we get $S \rhd \langle \mathbb{U}'; \mathbb{B}'; \mathbb{V} \rangle$. Since $\mathbb{V}' \subseteq \mathbb{V}$, several applications of Def. 4.9.2 give us $S \rhd \langle \mathbb{U}'; \mathbb{B}'; \mathbb{V}' \rangle = S'$. □

Proof of Lemma 4.13. Firstly, we consider hypothesis with respect to the axioms of configuration entailment (cf. Def. 4.10):

Def. 4.10.1. Assume that $[\bar{S}] \blacktriangleright [S \vee \bar{S}] \mapsto^r [\bar{T}]$. It follows that either (i) $[S] \mapsto^r [S']$ and $[\bar{T}] = [S' \vee \bar{S}]$ or (ii) $[\bar{S}] \mapsto^r [\bar{S}']$ and $[\bar{T}] = [S \vee \bar{S}']$. In case (i), we have $[\bar{V}] = [\bar{S}]$ and $[\bar{S}] \blacktriangleright [\bar{T}]$. In case (ii), we have $[\bar{V}] = [\bar{S}']$ and $[\bar{S}] \mapsto^r [\bar{S}'] \blacktriangleright [S \vee \bar{S}'] = [\bar{T}]$.

Def. 4.10.2. Assume that $[S_1 \vee S_2 \vee \bar{S}] \blacktriangleright [S_2 \vee \bar{S}] \mapsto^r [\bar{T}]$ where $[S_1] \rhd [S_2]$. It follows that either (i) $[S_2] \mapsto^r [S_2']$ and $[\bar{T}] = [S_2' \vee \bar{S}]$ or (ii) $[\bar{S}] \mapsto^r [\bar{S}']$ and $[\bar{T}] = [S_2 \vee \bar{S}']$. In case (i), Lemma A.5 proves that there exists an S_1' such that $[S_1] \mapsto^r [S_1']$ and $[S_1'] \rhd [S_2']$. Hence, we get $[\bar{V}] = [S_1' \vee S_2' \vee \bar{S}]$ and $[S_1 \vee S_2 \vee \bar{S}] \mapsto^r [S_1' \vee S_2 \vee \bar{S}] \mapsto^r [S_1' \vee S_2' \vee \bar{S}] \blacktriangleright [S_2' \vee \bar{S}] = [\bar{T}]$. In case (ii), we have $[\bar{V}] = [S_1 \vee S_2 \vee \bar{S}']$ and $[S_1 \vee S_2 \vee \bar{S}] \mapsto^r [S_1 \vee S_2 \vee \bar{S}'] \blacktriangleright [S_2 \vee \bar{S}] = [\bar{T}]$.

For the reflexive closure of these axioms, the hypothesis is true as $[\bar{S}] = [\bar{U}]$ implies $[\bar{V}] = [\bar{T}]$. For their transitive closure, it follows by induction. Hence, the hypothesis holds for configuration entailment in general. □

Proof of Theorem 4.14. To preserve clarity, we will omit the set of proper axioms from the judgement symbol. Furthermore, $\mathcal{D}(\bar{U}, \bar{V})$ denotes the fact that for configurations $\bar{U}, \bar{V}$, there exist configurations $\bar{U}_1, \ldots, \bar{U}_n$ for some n such that:

$$\bar{U} \mapsto_\vee \bar{U}_1 \mapsto_\vee \ldots \mapsto_\vee \bar{U}_n \blacktriangleright \bar{V}$$

Entailment $\bar{U} \blacktriangleright \bar{V}$ implies $\mathcal{D}(\bar{U}, \bar{V})$. We define $\cdot \diamond \cdot$ as in the proof of Thm. 4.14.

Secondly, we define an operator on formulas analogous to merging on states: For any two (possibly empty) sequences of variables $\bar{x}, \bar{y}$ and quantifier-free formulas α, β let $\exists \bar{x}.\alpha \diamond \exists \bar{y}.\beta ::= \exists \bar{x}.\exists \bar{y}.\alpha \otimes \beta$. We observe that for arbitrary CHR states U, V where $U \diamond V$ exists, we have $U^L \diamond V^L \dashv\vdash (U \diamond V)^L$. In the following, we assume w.l.o.g. that all existentially quantified variables in the antecedent of a sequent occuring in π are renamed apart. Hence, for every two formulas of the form U^L, V^L occurring in the antecedent of one sequent in π, both $U \diamond V$ and $U^L \diamond V^L$ exist.

We introduce a completion function η, defined by the following table, where $\bar{S}$ is a configuration, $c_b(\bar{t})$ is a built-in constraint and $\Gamma \vdash \alpha$ is a sequent:

$$\begin{array}{lll}
\eta(\bar{S}^L) & ::= & \bar{S}^L \\
\eta(c_b(\bar{t})) & ::= & !c_b(\bar{t}) \\
\eta(\alpha, \Gamma) & ::= & \eta(\alpha) \diamond \eta(\Gamma) \\
\eta(\Gamma \vdash \alpha) & ::= & \eta(\Gamma) \vdash \eta(\alpha) \quad \text{for non-empty } \Gamma \\
\eta(\vdash \alpha) & ::= & \mathbf{1} \vdash \eta(\alpha)
\end{array}$$

From Lemma A.4 follows that for every sequent $\Gamma \vdash \alpha$ in π, we have $\eta(\Gamma \vdash \alpha) = \bar{U}^L \vdash \bar{V}^L$ for some configurations $\bar{U}, \bar{V}$. We show by induction over the depth of π that for every such $\bar{U}^L \vdash \bar{V}^L$, we have $\mathcal{D}(\bar{U}, \bar{V})$.

Base case: In case the proof of $\bar{S}^L \vdash \bar{T}^L$ consists of a single leaf, it is either an instance of a (*Identity*), (*R***1**), or (*L***0**), or a proper axiom $(\Gamma \vdash \alpha) \in (\Sigma_{\doteq} \cup \Sigma_{CT} \cup \Sigma_{\mathbb{P}})$.

—(*Identity*), (*R***1**), (*L***0**):

$$\frac{}{\alpha \vdash \alpha}\ (\mathit{Identity}) \quad \frac{}{\vdash \mathbf{1}}\ (R\mathbf{1}) \quad \frac{}{\mathbf{0} \vdash \alpha}\ (L\mathbf{0})$$

In the case of (*Identity*), we have $\eta(\alpha \vdash \alpha) = \bar{U}^L \vdash \bar{U}^L$ for some configuration $\bar{U}^L$. In the case of (*R***1**), we have $\eta(\vdash \mathbf{1}) = \bar{U}^L \vdash \bar{U}^L$ for $\bar{U} = \langle \top; \mathbb{V} \rangle$. As the entailment relation is reflexive, we have $\mathcal{D}(\bar{U}, \bar{U})$. In the case of (*L***0**), we have $\eta(0 \vdash \alpha) = \bar{U}^L \vdash \bar{V}^L$ where $\bar{U} \equiv S_\perp$. By Def. 2.6, Def. 4.9, and Def. 4.10, we have that $\bar{U}^L \blacktriangleright \bar{V}^L$ and therefore $\mathcal{D}(U, V)$.

—For a proper axiom $(\Gamma \vdash \alpha) \in (\Sigma_{\doteq} \cup \Sigma_{CT})$ we have $\Gamma \vdash \alpha = \bar{U}^L \vdash \bar{V}^L$ where $\bar{U}, \bar{V}$ are singular configurations such that $\bar{U} \blacktriangleright \bar{V}$ and therefore $\mathcal{D}(\bar{U}, \bar{V})$.

—For a proper axiom $(\Gamma \vdash \alpha) \in \Sigma_{\mathbb{P}}$ we have $\Gamma \vdash \alpha = \bar{U}^L \vdash \bar{V}^L$ where $\bar{U}, \bar{V}$ are singular configurations such that $\bar{U} \mapsto \bar{V}$ and therefore $\mathcal{D}(\bar{U}, \bar{V})$.

Induction step: We distinguish thirteen cases according to which is the last inference rule applied in the proof. Cut reduction implies that it must be one of (*Cut*), (*L***1**), (*L*⊗), (*R*⊗), (*Weakening*), (*Dereliction*), (*Contraction*), (*R*!), (*L*∃), (*R*∃), (*L*⊕), (*R*⊕$_1$), and (*L*⊕$_2$):

—(*L*⊗), (*Dereliction*), (*R*!): For (*Dereliction*) and (*R*!), the banged formula must be an atomic built-in constraint $c_b(\bar{t})$:

$$\frac{\Gamma, \alpha, \beta \vdash \gamma}{\Gamma, \alpha \otimes \beta \vdash \gamma}\ (L\otimes) \quad \frac{\Gamma, c_b(\bar{t}) \vdash \beta}{\Gamma, !c_b(\bar{t}) \vdash \beta}\ (\mathit{Dereliction}) \quad \frac{!\Gamma \vdash c_b(\bar{t})}{!\Gamma \vdash !c_b(\bar{t})}\ (R!)$$

Since $\eta(\alpha,\beta) = \eta(\alpha \otimes \beta)$ and $\eta(!c_b(\bar{t})) = \eta(c_b(\bar{t}))$, each of these rule is invariant to the η-completion of the sequent, thus trivially satisfying the hypothesis.

—(*L***1**):

$$\frac{\Gamma \vdash \alpha}{\Gamma, \mathbf{1} \vdash \alpha}\ (L\mathbf{1})$$

We assume that $S_\Gamma = \langle \mathbb{G}_\Gamma, \mathbb{V}_\Gamma \rangle$ is a singular configuration and $\bar{S}_\alpha$ is a configuration such that $S_\Gamma^L = \eta(\Gamma)$, $\bar{S}_\alpha^L = \eta(\alpha)$, and $\mathcal{D}(S_\Gamma, \bar{S}_\alpha)$. Then by Def. 2.6.3, we have $\mathcal{D}(U_\Gamma, S_\alpha)$ where $U_\Gamma = \langle \mathbb{G}_\Gamma \wedge \top, \mathbb{V}_\Gamma \rangle$. As $U_\Gamma^L = \eta(\Gamma, \mathbf{1})$, this proves the hypothesis.

—(*Weakening*): By Lemma A.4, we have that the introduced formula is of the form $!c_b(\bar{t})$.

$$\frac{\Gamma \vdash \beta}{\Gamma, !c_b(\bar{t}) \vdash \beta}\ (\mathit{Weakening})$$

We assume that $S_\Gamma = \langle \mathbb{G}_\Gamma, \mathbb{V}_\Gamma \rangle$ is a singular configuration and $\bar{S}_\beta$ is a configuration such that $S_\Gamma^L = \eta(\Gamma)$, $\bar{S}_\beta^L = \eta(\beta)$ and $\mathcal{D}(S_\Gamma, \bar{S}_\beta)$. Furthermore, let $U = \langle \mathbb{G}_\Gamma \wedge c_b(\bar{t}); \mathbb{V}_\Gamma \rangle$. Since $U^L = \eta(\Gamma,\ !c_b(\bar{t}))$ and $U \rhd S_\Gamma$, Lemma A.5 proves the hypothesis.

—(*Contraction*): By the subformula property, we have that the contracted formula is of the form $!c_b(\bar{t})$.

$$\frac{\Gamma, !c_b(\bar{t}), !c_b(\bar{t}) \vdash \beta}{\Gamma, !c_b(\bar{t}) \vdash \beta}\ (\mathit{Contraction})$$

Since $\langle \mathbb{U}; \mathbb{B} \wedge c_b(\bar{t}); \mathbb{V} \rangle \rhd \langle \mathbb{U}; \mathbb{B} \wedge c_b(\bar{t}) \wedge c_b(\bar{t}); \mathbb{V} \rangle$ we prove the hypothesis analogously to (*Weakening*).

—($R\otimes$): The subformula property implies that the joined formulas must be singular configurations U^L and V^L without local variables:

$$\frac{\Gamma \vdash U^L \quad \Delta \vdash V^L}{\Gamma, \Delta \vdash U^L \otimes V^L}\ (R\otimes)$$

Let S_Γ, S_Δ be singular configurations such that $S_\Gamma^L = \eta(\Gamma)$, $S_\Delta^L = \eta(\Delta)$. The induction hypothesis gives us $\mathcal{D}(S_\Gamma, U)$ and $\mathcal{D}(S_\Delta, V)$. By Lemma A.3.1 and Lemma A.3.2 we have $\mathcal{D}(S_\Gamma \diamond S_\Delta, U \diamond S_\Delta)$ and $\mathcal{D}(U \diamond S_\Delta, U \diamond V)$. By Lemma A.5, we get $\mathcal{D}(S_\Gamma \diamond S_\Delta, U \diamond V)$.

—(Cut): Since π is a cut-reduced proof and all axioms are of the form $U_1^L \vdash U_2^L$, the eliminated formula must be the logical reading of a singular configuration U:

$$\frac{\Gamma \vdash U^L \quad U^L, \Delta \vdash \beta}{\Gamma, \Delta \vdash \beta}\ (Cut)$$

Let S_Γ, S_Δ be singular configurations and $\bar{S}_\beta$ a configuration such that $S_\Gamma^L = \eta(\Gamma)$, $S_\Delta^L = \eta(\Delta)$, and $\bar{S}_\beta^L = \eta(\beta)$. The induction hypothesis gives us $\mathcal{D}(S_\Gamma, U)$ and $\mathcal{D}(U \diamond S_\Delta, \bar{S}_\beta)$. Applying Lemma A.3, we get $\mathcal{D}(S_\Gamma \diamond S_\Delta, U \diamond S_\Delta)$. By Lemma A.5, we get $\mathcal{D}(S_\Gamma \diamond S_\Delta, \bar{S}_\beta)$ which proves the hypothesis.

—($L\exists$): In the preconditional sequent, the quantified variable x is by definition replaced by a fresh constant a that does not occur in Γ, α, or β:

$$\frac{\Gamma, \alpha\,[x/a] \vdash \beta}{\Gamma, \exists x.\alpha \vdash \beta}\ (L\exists)$$

Let $U = \langle \mathbb{G}\,[x/a]\,; \mathbb{V} \cup \{a\}\rangle$ be a singular configuration and $\bar{S}_\beta$ a configuration such that $U^L = \eta(\Gamma, \alpha\,[x/a])$, $\bar{S}_\beta^L = \eta(\beta)$, and $x \notin \mathbb{V}$. The definition of state equivalence gives us $U \equiv \langle \mathbb{G} \wedge x \doteq a; \mathbb{V} \cup \{a\}\rangle$. Furthermore, we have $\eta(\Gamma, \exists x.\alpha) = \langle \mathbb{G}, \mathbb{V}\rangle^L$. By the induction hypothesis, we have singular configurations $U_1, \ldots, U_n$ such that $U \mapsto^{r_1} U_1 \mapsto^{r_2} \ldots \mapsto^{r_n} U_n \rhd S_\beta$ where $U_i = \langle \mathbb{G}_i \wedge x \doteq a; \mathbb{V} \cup \{a\}\rangle$ for $i \in \{1, \ldots, n\}$. Neither the binding $x \doteq a$ nor the set of global variables affect rule applicability. Hence, we can construct an analogous derivation $\langle \mathbb{G}; \mathbb{V}\rangle \mapsto^{r_1} U_1' \mapsto^{r_2} \ldots \mapsto^{r_n} U_n'$ where $U_i' = \langle \mathbb{G}_i; \mathbb{V}\rangle$ for $i \in \{1, \ldots, n\}$. Since $U_n \blacktriangleright \bar{S}_\beta$ and a must not occur in β, we also have have $\langle \mathbb{G}_n; \mathbb{V}\rangle \blacktriangleright \bar{S}_\beta$. Therefore, we have $\mathcal{D}(\langle \mathbb{U}; \mathbb{B}; \mathbb{V}\rangle, \bar{S}_\beta)$. As $\eta(\Gamma, \exists x.\alpha) = \langle \mathbb{G}; \mathbb{V}\rangle^L$, this proves the hypothesis.

—($R\exists$): By definition, the quantified variable x substitutes an arbitrary term t.

$$\frac{\Gamma \vdash \beta[x/t]}{\Gamma \vdash \exists x.\beta}\ (R\exists)$$

Let $\bar{S}_\Gamma$ be a configuration and U, V be singular configurations such that $\bar{S}_\Gamma^L = \eta(\Gamma)$, $U^L = \eta(\beta\,[x/t])$, and $V^L = \eta(\exists x.\beta)$. By the induction hypothesis we have $\mathcal{D}(\bar{S}_\Gamma, U)$ for some n. Let $V = \langle \mathbb{G}; \mathbb{V}\rangle$ and $U = \langle \mathbb{G}\,[x/t]\,; \{x\} \cup \mathbb{V}\rangle$. We have $U \equiv \langle \mathbb{G} \wedge x \doteq t; \{x\} \cup \mathbb{V}\rangle \rhd \langle \mathbb{G} \wedge x \doteq t; \mathbb{V}\rangle \rhd \langle \mathbb{G}; \mathbb{V}\rangle \equiv V$, and therefore, $\mathcal{D}(\bar{S}_\Gamma, V)$.

—($L\oplus$):

$$\frac{\Gamma, \alpha \vdash \gamma \quad \Gamma, \beta \vdash \gamma}{\Gamma, \alpha \oplus \beta \vdash \gamma}\ (L\oplus)$$

Let $\mathbb{G}_\alpha, \mathbb{G}_\beta$ be goals, let $S_\Gamma = \langle \mathbb{G}; \mathbb{V}\rangle$ be a state and let $\bar{S}_\beta$ be a configuration such that $\mathbb{G}_\alpha^L = \eta(\alpha)$, $\mathbb{G}_\beta^L = \eta(\beta)$, $S_\Gamma^L = \eta(\Gamma)$ and $\bar{S}_\gamma^L = \eta(\gamma)$. Let furthermore $\bar{y}_\alpha = vars(\mathbb{G}_\alpha)$ and $\bar{y}_\beta = vars(\mathbb{G}_\beta)$. Hence, $\eta(\Gamma, \alpha) = \langle \mathbb{G} \wedge \mathbb{G}_\alpha; \mathbb{V} \cup \bar{y}_\alpha\rangle^L$, $\eta(\Gamma, \beta) = \langle \mathbb{G} \wedge \mathbb{G}_\beta; \mathbb{V} \cup \bar{y}_\beta\rangle^L$,

and $\eta(\Gamma, \alpha \oplus \beta) = \langle \mathbb{G} \wedge (\mathbb{G}_\alpha \vee \mathbb{G}_\beta); \mathbb{V} \cup \bar{y}_\alpha \cup \bar{y}_\beta \rangle^L$. The induction hypothesis gives us $\mathcal{D}(\langle \mathbb{G} \wedge \mathbb{G}_\alpha; \mathbb{V} \cup \bar{y}_\alpha \rangle, \bar{S}_\gamma)$ and $\mathcal{D}(\langle \mathbb{G} \wedge \mathbb{G}_\beta; \mathbb{V} \cup \bar{y}_\beta \rangle, \bar{S}_\gamma)$. By Def. 2.14.4 we have that $\eta(\Gamma, \alpha \oplus \beta) \equiv \langle \mathbb{G} \wedge \mathbb{G}_\alpha; \mathbb{V} \cup \bar{y}_\alpha \rangle \vee \langle \mathbb{G} \wedge \mathbb{G}_\beta; \mathbb{V} \cup \bar{y}_\beta \rangle$. Finally by Lemma 4.13, we get $\mathcal{D}(\langle \mathbb{G} \wedge (\mathbb{G}_\alpha \vee \mathbb{G}_\beta); \mathbb{V} \cup \bar{y}_\alpha \cup \bar{y}_\beta \rangle, \bar{S}_\gamma)$.

—$(R\oplus_1)$, $(R\oplus_2)$:

$$\frac{\Gamma \vdash \alpha}{\Gamma \vdash \alpha \oplus \beta}\ (R\oplus_1) \qquad \frac{\Gamma \vdash \beta}{\Gamma \vdash \alpha \oplus \beta}\ (R\oplus_2)$$

We consider $(R\oplus_1)$: By the subformula property, there exist configurations $\bar{S}_\Gamma, \bar{S}_\alpha, \bar{S}_\beta$, such that $\bar{S}_\Gamma^L = \eta(\Gamma)$, $\bar{S}_\alpha^L = \eta(\alpha)$, and $\bar{S}_\beta^L = \eta(\beta)$. By the induction hypothesis, we have $\mathcal{D}(\bar{S}_\Gamma, \bar{S}_\alpha)$. By Def. 4.10.1, we have $\bar{S}_\alpha \blacktriangleright (\bar{S}_\alpha \vee \bar{S}_\beta)$ and therefore $\mathcal{D}_n(\bar{S}_\Gamma, \bar{S}_\alpha \oplus \bar{S}_\beta)$. (The proof for $(R\oplus_2)$ works analogously.)

Finally, we have $\mathcal{D}(\bar{S}, \bar{T})$, i.e. there exist configurations $\bar{S}_1, \ldots \bar{S}_n$ such that:

$$\bar{S} \mapsto \bar{S}_1 \ldots \mapsto \bar{S}_n \rhd \bar{T}$$

It follows that for $\bar{U} = \bar{S}_n$, we have $\bar{S} \mapsto^* \bar{U}$ and $\bar{U} \rhd \bar{T}$. □

Proof of Lemma 4.16. ('⇐') Follows from Thm. 4.14 by assuming an empty program $\mathbb{P} = \emptyset$.
('⇒') We consider the axioms for configuration entailment in Def. 4.10: W.r.t. axiom (1), $[\bar{T}] \blacktriangleright [S \vee \bar{T}]$ implies $\bar{T}^L \vdash (S \vee \bar{T})^L$ since $\beta \vdash \alpha \oplus \beta$. For Def. 4.10.2, $[S_1] \rhd [S_2]$ implies $S_1^L \vdash_\Sigma S_2^L$ by Lemma A.6. From a proof of $S_1^L \vdash_\Sigma S_2^L$, we can construct a proof of $S_1^L \oplus S_2^L \oplus \bar{T}^L \vdash_\Sigma S_2^L \oplus \bar{T}^L$. As $\vdash_\Sigma$ is furthermore reflexive and transitive, the hypothesis is reduced to Lemma 4.5. □

Proof of Theorem 4.20. We prove Thm. 4.20 by showing that any proof tree in the axiomatic semantics can be transformed into a proof tree in the encoding semantics and vice versa. To ensure of clarity, we will omit the set of proper axioms from the judgement symbol.

Axiomatic to encoding:. We assume a proof π of a sequent $\bar{S}^L \vdash \bar{T}^L$ in the axiomatic semantics. We replace every axiom $\exists \bar{x}.\mathbb{B}^L \vdash \exists \bar{x}'.\mathbb{B}'^L$ in Σ_{CT} by a sub-tree proving $CT^L, \exists \bar{x}.\mathbb{B}^L \vdash \exists \bar{x}'.\mathbb{B}'^L$. Analogously, we replace every axiom $\exists \bar{x}.\mathbb{G}^L \vdash \exists \bar{x}'.\mathbb{G}'^L$ in $\Sigma_{\doteq}$ by a sub-tree proving $ET^L, \exists \bar{x}.\mathbb{B}^L \vdash \exists \bar{x}'.\mathbb{B}'^L$. Similarly, every axiom $H_1^L \otimes H_2^L \otimes G^L \vdash H_1^L \otimes \exists_{-\bar{y}_r}.(B^L \otimes G^L)$ in $\Sigma_{\mathbb{P}}$ is replaced with a sub-tree proving $\mathbb{P}^L, H_1^L \otimes H_2^L \otimes G^L \vdash H_1^L \otimes \exists_{-\bar{y}_r}.(B^L \otimes G^L)$. We propagate the thus introduced instances of CT^L and $\mathbb{P}^L$ throughout the proof tree, thus producing a proof π' of

$$CT^L, \ldots, CT^L, ET^L, \ldots, ET^L, \mathbb{P}^L, \ldots, \mathbb{P}^L, \bar{S}^L \vdash \bar{T}^L$$

We insert π' into:

$$\cfrac{\cfrac{\cfrac{\pi'}{CT^L, ET^L, \mathbb{P}^L, \bar{S}^L \vdash \bar{T}^L}\ (\mathit{Contraction})^*}{CT^L, ET^L, \mathbb{P}^L \vdash \bar{S}^L \multimap \bar{T}^L}\ (R\multimap)}{CT^L, ET^L, \mathbb{P}^L \vdash \forall(\bar{S}^L \multimap \bar{T}^L)}\ (R\forall)$$

Encoding to axiomatic:. Let $\bigotimes$ stand for element-wise multiplicative conjunction of a set and let π be a proof of a sequent $CT^L, \mathbb{P}^L \vdash \forall(\bar{S}^L \multimap \bar{T}^L)$ in the encoding semantics.

For every $!\forall(\exists\bar{x}.\mathbb{B}^L \multimap \exists\bar{x}'.\mathbb{B}'^L) \in CT^L$, we have $\vdash_\Sigma !\forall(\exists\bar{x}.\mathbb{B}^L \multimap \exists\bar{x}'.\mathbb{B}'^L)$ where $\Sigma = \Sigma_{CT}$. Hence, there exists a proof π_{CT} of $\vdash_\Sigma \bigotimes CT^L$. Similarly, there exist proofs π_{ET} of $\vdash_\Sigma \bigotimes ET^L$ and $\pi_{\mathbb{P}}$ of $\vdash_{\Sigma_{\mathbb{P}}} \bigotimes \mathbb{P}^L$.

$$\dfrac{\pi_{\mathbb{P}}\ \dfrac{\pi_{ET}\ \dfrac{\pi_{CT}\ \dfrac{\pi}{\bigotimes CT^L, \bigotimes ET^L, \bigotimes \mathbb{P}^L \vdash \forall(\bar{S}^L \multimap \bar{T}^L)}\,(L\otimes)^*}{\bigotimes ET^L, \bigotimes \mathbb{P}^L \vdash \forall(\bar{S}^L \multimap \bar{T}^L)}\,(Cut)}{\bigotimes \mathbb{P}^L \vdash \forall(\bar{S}^L \multimap \bar{T}^L)}\,(Cut)}{\vdash \forall(\bar{S}^L \multimap \bar{T}^L)} \qquad \dfrac{\dfrac{\dfrac{}{\bar{S}^L \vdash \bar{S}^L}\,(Identity) \quad \dfrac{}{\bar{T}^L \vdash \bar{T}^L}\,(Identity)}{\bar{S}^L, \bar{S}^L \multimap \bar{T}^L \vdash \bar{T}^L}\,(L\multimap)}{\forall(\bar{S}^L \multimap \bar{T}^L), \bar{S}^L \vdash \bar{T}^L}\,(L\forall)^*}{\bar{S}^L \vdash \bar{T}^L}\,(Cut)$$

As we can transform the respective proof tree from the axiomatic to the encoding semantics and vice versa, the two representations are equivalent. □