Proofs as concurrent processes

A Logical Interpretation of Concurrent Constraint Programming*

Patrick Lincoln
SRI International

Vijay Saraswat
Penn State University

April 2003
DRAFT

Abstract

Following the formula-as-state and proof-as-action analogy due to Girard, and using ideas of process testing [dNH84,Hen88], we present the *proof-as-concurrent-computation* paradigm.

Assume given a programming language $\mathcal{P}$ with a class of agents A, tests T, and a "transition" relation $\longrightarrow \subseteq (A \times T) \times (A \times T + \{\mathtt{true}\})$ wihch captures the operational semantics of the language. We say that A passes the test T iff $\langle A, T\rangle \longrightarrow^* \mathtt{true}$. We shall say that a logic $\mathcal{L}$ (with an entailment relation $\vdash_{\mathcal{L}}$) *corresponds* to $\mathcal{P}$ if it is possible to read A and T as formulas $[\![A]\!]$ and $[\![T]\!]$ of $\mathcal{L}$ such that A passes the test T iff $[\![A]\!] \vdash_{\mathcal{L}} [\![T]\!]$. Such a correspondence thus establishes a (concurrent) computational interpretation of the logic, and a logical interpretation of the programming language. Examples of such correspondences are: LL($\otimes$) and Petri-nets [Asp90,Gun85], LL($\otimes, \oplus$) and nondeterministic counter machines [LMSS90], "cancellative linear logic" and financial games [MOM91], a fragment of (classical) logic and logic programming. Thus, we propose that a sequent $\Gamma \vdash \Delta$ be read as "agent Γ satisfies the test Δ".

The main body of this paper establishes the computational content of rich fragments of (first-order) intuitionistic logic by relating them to the paradigm of concurrent constraint programming [Sar93]. Conversely, this establishes a simple logical interpretation for the cc languages complementing their well-understood operational and denotational semantics [SRP91].

1 Introduction

This paper is about connections between operational semantics of concurrent programming languages and proof-theory.

Our basic idea is that the state of the concurrent program may be described as a logical formula, with conjunction treated as parallel composition. The notion of *testing* arises naturally in such a context as the probing of the "environment" (the other agents running in parallel) with different kinds of tests. The simplest test is whether the environment is capable of producing some pieces of partial information (constraints). This is the idea of the "ask" operation of concurrent constraint programming. However, much richer notions of testing are also possible, such as:

- Disjunctive testing: The system passes the test if it passes either test.
- Conjunctive testing: The system passes the test if it passes both tests.

*An earlier draft of this paper with the same title was circulated in November 1991 as a technical report from the Xerox Palo Alto Research Center (PARC). Portions of this work were done by the first author while he was at Stanford University, and by the second author while he was at Xerox PARC.

- Reactive (or hypothetical) testing: The test is in the form of a pair: an agent A and a test T. The test is passed if the environment augmented by the agent A can pass the test T. However, when the test is passed, the environment remains unchanged. Thus such a test is checking whether the environment would pass the test T *were it to be augmented* with the agent A.
- Generic testing: This checks if the environment can pass the test generically, that is, for all possible values of a specified variable.
- Recursive testing: The test unfolds recursively; the system of agents passes the test if it passes each of the generated tests.

(See Section 5 for more discussions.)

Given that the state of the computation is described by a formula, it is natural to think of a test as also specified by a formula. The above notions of testing lead to intuitively natural logical interpretations. A disjunctive test is of the form $T_1 \vee T_2$, a conjunctive test is of the form $T_1 \wedge T_2$, a reactive test of the form $A \supset T$, a generic test of the form $\forall X.T$, and a recursive test is of the form g where the underlying theory may be considered to be enriched with formulas $(\forall)(G \supset g)$.

Given that both agents and tests may be thought of as formulas in the logic, it then becomes natural to ask whether the entailment relation of the logic $\vdash$ captures precisely the operational idea of the agent passing the test. That is, suppose that we have an operational semantics specified in the form of a binary transition relation $\longrightarrow$ on pairs $A : G$ of agents and tests, together with the notion that a configuration transitions to the special configuration `true` if the agent A should be considered to pass the test G. The central question that arises, then, is whether $A : G \longrightarrow^*$ `true` iff $A \vdash G$. That is, operational execution equals logical derivation.

Note that Horn clause logic programming may be thought of as the degenerate case which has a vacuous agent A and recursive tests. Computation now establishes whether some value for the free variables of the test which allows them to be satisfied by the vacuous agent. A similar interpretation may be provided to Petri nets [Asp90,Gun85], nondeterministic counter machines [LMSS90] and financial games [MOM91].

In this paper we take the first step towards such a computational interpretation of intuitionistic logic by studying concurrent constraint programming. We establish completeness for primtive ask tests. We leave as open for future work whether this kind of correspondence can be carried over to the richer tests described above.

1.1 Concurrent Constraint programming

The concurrent constraint (cc) programming paradigm is based on a small set of principles [Sar93] gleaned from the study of concurrent logic programming and constraint logic programming. In *constraint-based* computation, the usual store of values underlying standard imperative programming language operational semantics is replaced by a store of constraints. That is, the store contains partial information which specifies admissible values for the variables of a program. Computation proceeds by introducing more and more refined constraints in a monotonic accumulation of constraints in the store. *Concurrency* arises naturally by the notion of multiple threads of control interacting with a shared store of constraints. *Synchronization*, for example, is achieved through the blocking of certain conditional expressions until (if ever) there is enough information in the store to entail a given constraint. Other processes may unblock such a waiting process by adding the requisite information to the store.

The basic test here is just: has the store been upgraded to the above a given constraint?

The basic idea. The constraint system is assumed to be specified by means of sequents of a particular form. Similarly, program definitions are given by axioms of another form.

Programs are described by the syntactic category P, agents by A and goals (or tests) by G.

$$\begin{array}{lcl} P & ::= & a \rightarrow A \mid \forall X.P \mid P \wedge P \\ A & ::= & c \mid c \ \rightarrow \ A \mid a \mid A \wedge A \mid \exists X.A \\ G & ::= & c \end{array}$$

Here a ranges over atomic formulas, called *atoms* that can be thought of as procedure calls. c ranges over a given set C of atomic formulas called *primitive constraints*. The set of atoms is assumed to be distinct from the set of constraints. C comes equipped with a collection of *constraint sequents* of the form:

$$\Gamma, c_1, \ldots, c_k \vdash_{\mathcal{C}} c$$

The basic idea is that A corresponds to the syntactic category of "agents", P allows for the (recursive) definition of agents, and queries check whether an agent satisfies a primitive constraint. (The universal quantification in G enables variables to be shared between the agent and the constraint being checked to hold of the agent.)

Example 1.1 (Nrev) Consider the example of naive reverse:

```
append([], X, Y) :: X =Y.
append([A|B], X, Y) :: R^(Y=[A|R], append(B, X, R)).
nrev([], X) :: X=[].
nrev([A|X], Y) :: L^(nrev(X, L), append(L, [A], Y)).
```

On the presentation of an agent `nrev([1,2,3], X)` computation progresses by recursively spawning two agents (with a new variable to connect them). Eventually an `nrev` agent will produce a list, thus triggering an `append` agent. The output of the `append` will trigger another `nrev` etc. □

Thus, cc computations correspond to logic programming computations – but with a "non-standard" notion of program and query. Programs are "inverted" definite clauses: instead of providing sufficient conditions for a literal (that is, being of the form $\phi \rightarrow A$), they provide necessary conditions, that is, are of the form $A \rightarrow \phi$. Correspondingly, queries are universally quantified implications instead of being existentially quantified conjunctions (as in Prolog).

This paper provides a logical foundation for cc. We choose intuitionistic logic for its inherently constructive (implementable) nature. In the future we hope to find another logical foundation for the extended *linear* concurrent constraint programming framework building on Girard's linear logic. We feel that such a linear logical foundation will guide the extension of cc languages to resource-conscious computations.

In this paper we show that non-deterministic cc programs can be regarded as formulas in a certain fragment of intuitionistic logic which enjoys a rich set of logical permutabilities.

2 Operational semantics for CCP

The syntax of agents and tests is as before. A configuration consists of a multiset of agents and a test, G. The multiset of agents is also called a *vat*.

$$\begin{array}{rcl} \text{Agents } A, B & ::= & c \mid A \wedge A \mid \exists x.A \mid G \rightarrow A \mid a \\ \text{Tests } G, H & ::= & c \\ \text{Config } s & ::= & u : G \\ \text{Vat } u & ::= & u_1, \ldots, u_k \mid A \end{array}$$

The transition relation $\longrightarrow \subseteq s \times s \cup \texttt{true}$ is defined using the rules below. Intuitively, the multiset constructor corresponds to the parallel execution of agents

$$u, A_1 \wedge A_2 : G \longrightarrow u, A_1, A_2 : G$$
$$u, \exists x.A : G \longrightarrow u, A : G \qquad (x \text{ not free in (u,G)})$$

$$u, a : G \longrightarrow u, A : G \quad (a \vdash A \text{ program axiom})$$
$$u, c \rightarrow A : G \longrightarrow u, A : G \qquad (u \vdash c)$$
$$u : G \longrightarrow \texttt{true} \qquad (u \vdash G)$$

Let $\longrightarrow^\star$ be the reflexive, transitive closure of $\longrightarrow$. In [SRP91], in slightly different setup, we have shown that the transition relation is confluent under fair reduction sequences, and that there is an associated denotational semantics which treats agents as closure operators over the underlying constraint system.

The formulation of the operational semantics here has been chosen to make the connection with deductions in intuitionistic logic transparent.

3 The LCC calculus

Various forms of proof normalization have been proposed for various sequent calculii in the past; for example, Gentzen's cut-free proofs (the product of his "Haupstatz" theorem) [Kle52a]. More recently, Miller [Mil89] has described a normal form of proofs in sequent logic. Informally, a proof is *uniform* if the right hand side is always reduced to atomic formulas before any left rules are applied. More formally, a cut-free proof is uniform if every sequent with non-atomic succedent is the conclusion of some right-introduction rule.

In this paper, we investigate a different normal form for proofs we call *simple* proofs. A cut-free proof is simple if productions on the left side are only used when the requisite information is already available in the context. That is, the left hypothesis of the left-implication rule (called $\supset\vdash$ below) is trivial. Formally, a cut-free proof is simple if in every application of the ($\supset\vdash$) rule, the left hypothesis is proven with axioms (logical or nonlogical) only.

3.1 The LJ calculus for Intuitionistic Logic

The logical framework for this paper is intuitionistic logic. We use (a minor variant of) Gentzen's sequent formulation of this logic, LJ [Sza69]. An intuitionistic sequent is of the form $\Gamma \vdash \Delta$, where Γ and Δ are multisets of formulas. In LJ it is the case that the right hand side, or succedent, never contains more than one formula. That is, $|\Delta| \leq 1$. The full set of proof rules for LJ are given in Figure 1. The quantifier rules have some side-conditions:

- in rules for quantifiers of universal force (the rules $(\vdash \forall)$ and $(\exists \vdash)$) y must not occur free in Γ, Δ
- in rules for quantifiers of existential force (the rules $(\vdash \exists)$ and $(\forall \vdash)$), the notation $A[t/x]$ stands for the substitution of t for all free occurrences of x in A, where bound variables of A are renamed apart from t.

In sequent calculii, a *proof* is a tree, usually presented with the root at the bottom and the leaves at the top. Each branch of the proof is a sequence of applications of the proof rules given in Figure 1. Some of these rules (e.g. $(\vdash \wedge)$) represent branching points in the proof tree. Some (e.g. $(\vee \vdash)$) extend the length of a branch. Some (e.g. (I)) terminate a branch. The leaves embody the assumptions and the root the conclusion. Such a structure is said to be a *proof* of the conclusions from the assumptions. If the set of assumptions is empty, such a structure represents a *proof* of the conclusion. We write $\Gamma \vdash_{\mathrm{LJ}} \Delta$ when there is a proof of this sequent in LJ.

In some sequent calculus presentations, the structual rules of weakening, contraction and exchange are presented as explicit proof rules. Here we have "pushed weakening to the leaves" to mimic the monotonic behavior of the operational semantics we have in mind. Also, we have encoded contraction implicitly in each rule. Since the two sides of the sequent are taken to be multisets, exchange is implied. The actual removal of the structural rules from classical logic (as opposed to making them implicit) leads to linear logic [Gir87].

As Gentzen proved, applications of the (Cut) rule may be removed from any LJ proof.

Theorem 3.1 (Cut-Elimination for LJ) $\Gamma \vdash_{\mathrm{LJ}} \Delta$ *iff* $\Gamma \vdash_{\mathrm{LJ}-\mathit{Cut}} \Delta$

$$\frac{}{\Gamma, A \vdash A}\ (I) \qquad \frac{\Gamma \vdash C \quad \Gamma, C \vdash \Delta}{\Gamma \vdash \Delta}\ (Cut)$$

$$\frac{\Gamma \vdash A}{\Gamma, \neg A \vdash \Delta}\ (\neg \vdash) \qquad \frac{\Gamma, A \vdash \neg A}{\Gamma \vdash \neg A}\ (\vdash \neg)$$

$$\frac{\Gamma, A \supset B \vdash A \quad \Gamma, A \supset B, B \vdash \Delta}{\Gamma, A \supset B \vdash \Delta}\ (\supset \vdash) \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \supset B}\ (\vdash \supset)$$

$$\frac{\Gamma, A \vee B, A \vdash \Delta \quad \Gamma, A \vee B, B \vdash \Delta}{\Gamma, A \vee B \vdash \Delta}\ (\vee \vdash) \qquad \frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\ (\vdash \vee_1) \quad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\ (\vdash \vee_2)$$

$$\frac{\Gamma, A \wedge B, A, B \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta}\ (\wedge \vdash) \qquad \frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\ (\vdash \wedge)$$

$$\frac{\Gamma, \forall x.A, A[t/x] \vdash \Delta}{\Gamma, \forall x.A \vdash \Delta}\ (\forall \vdash) \qquad \frac{\Gamma \vdash A[y/x]}{\Gamma \vdash \forall x.A}\ (\vdash \forall)$$

$$\frac{\Gamma, \exists x.A, A[y/x] \vdash \Delta}{\Gamma, \exists x.A \vdash \Delta}\ (\exists \vdash) \qquad \frac{\Gamma \vdash A[t/x]}{\Gamma \vdash \exists x.A}\ (\vdash \exists)$$

Figure 1: Rules for LJ

For a proof see [Kle52a]. This theorem immediately yields many important corollaries. Chief among them is the *subformula* property. Every formula appearing anywhere in a proof of $\Gamma \vdash \Delta$ is a subformula of Γ or Δ. The notion of subformula is defined as follows: The formula A is a subformula of itself. If A is a subformula of B, then A is also a subformula of $C \wedge B, B \wedge C, C \vee B, B \vee C, C \supset B, B \supset C, \neg B, (\forall x : B)$ and $(\exists x : B)$. Also, for any term t, the formula $A[t/x]$ is considered to be a subformula of both $\forall x : A$ and $\exists x : A$.

Corollary 3.2 *If A is a subformula of any formula in any sequent in any cut-free proof of $\Gamma \vdash_{\mathrm{LJ}} \Delta$ from assumptions Σ, then A is also a subformula of some formula in $\Gamma, Delta$.*

The proof is by induction on the depth of the proof and case analysis on all the non-cut rules.

Another immediate corollary is consistency of the logic:

Corollary 3.3 (LJ **Consistency)** *$\vdash A$ is not provable in* LJ *for any propositional symbol A.*

Proof 3.3 If a proof exists, then there must be a cut-free proof. But no rule other than Cut can possibly have $\vdash A$ as a conclusion. □

Now one may see the implication of the intuitionistic restriction to one formula in the succedent. In classical logic, $\vdash A \vee \neg A$ is provable, but not in LJ. To see this, considerable possible cut-free proofs. The cut-free proof must end in $(\vdash \vee)$. But then neither of its premisses is provable in LJ, by Corollary 3.3.

3.1.1 LJ+: Intuitionistic Logic with Constraints

We extend LJ with nonlogical axioms (called *constraint axioms*) of the form $\Gamma, c_1, \ldots, c_n \vdash c$ where Γ is the unspecified remainder of the context, and $c_1, \ldots, c_n, c$ are atomic formulas. We will study proofs with assumptions of the form $\Gamma, c_1, \ldots, c_n \vdash c$. We assume that the set of constraint axioms is closed under entailment. That is, if $\Gamma, c_1, \ldots, c_n \vdash c$ and $\Delta, d_1, \ldots, d_k, c \vdash d$ are constraint axioms, then so is $\Gamma, \Delta, c_1, \ldots, c_n, d_1, \ldots, d_k \vdash d$. One may think of this condition as saying that the set of constraint axioms admits cut elimination, since any deduction involving a cut on a constraint between two constraint axioms can be replaced by a deduction involving a single constraint axiom and no application of cut.

Theorem 3.4 (LJ+ **Cut Normalization)** *There is a proof of $\Gamma \vdash_{\mathrm{LJ+}} \Delta$ from assumptions Σ if and only if there is a proof of $\Gamma \vdash_{\mathrm{LJ+}} \Delta$ from assumptions Σ, where every application of (Cut) has a sequent in Σ as at least one hypothesis.*

Proof 3.4 To simplify presentation, we just give the modifications to the well-known cut-elimination procedures for LJ [Kle52a]. The only new cases are due to nonlogical axioms of the form $\Gamma, c_1, \ldots, c_k \vdash c$, and assumptions in Σ. We define the *principal formula* for the new axioms as any of the c_i and c, that is the only principal formulas of this rule are atomic constraints. We also define all formulas appearing in sequents in Σ as principal in their introduction. For the other rules of LJ+ the definition of principal formula is the usual one (the formula introduced by the rule). (Note that atomic constraints are not the principal formula of any other rule.) Thus a cut between two nonlogical axioms be reduced to one nonlogical axiom, the existence of which is guaranteed as discussed above. Otherwise, we leave the cut-elimination procedure unmodified, leaving no cut-reductions for some cases when one hypothesis is from Σ. Thus cuts will be eliminated or have one hypothesis from Σ. □

One may read this theorem as saying that cuts may be permuted so that cuts are applied only to nonlogical axioms or assumptions. We will call any proof normalized by this theorem as *cut-normal*.

Corollary 3.5 (LJ+ **subformula property)** *If A is a subformula of any formula in any sequent in any cut-normal proof of $\Gamma \vdash_{\mathrm{LJ+}} \Delta$ from assumptions Σ, then A is also a subformula of some formula in Γ, Δ, Σ.*

Proof 3.5 By induction on the depth of proof and case analysis on all the non-cut rules. □

In the table below, an entry $-$ is read as saying that the subterm (specified by the column) has polarity opposite to that of the term (specified by the row). An entry $+$ specifies the polarity is the same.

	A	B
$\neg A$	-	
$\exists x : A$	+	
$\forall x : A$	+	
$A \vee B$	+	+
$A \wedge B$	+	+
$A \supset B$	-	+

Figure 2: Polarity of formulas

Polarity. We introduce the definition of polarity as in Figure 2. following definition. We claim that polarity is preserved throughout cut-free proofs.

Lemma 3.6 (Polarity Preservation) *If a formula A has polarity p in an occurrence in a sequent in a cut-normal proof of $\Gamma \vdash_{\mathrm{LJ}} \Delta$ from assumptions Σ, then A has polarity p in $\Gamma \vdash \Delta$ or in Σ.*

This lemma may be proven by induction on the size of the proof.

3.2 The LCC calculus

We define the logic LCC to be LJ without the proof rules $(\vdash \neg)$, $(\neg \vdash)$ and $(\vdash \exists)$. We define LCC+ to be LCC augmented with constraint axioms, as above for LJ+, and further augmented with axioms representing a program P.

For any formula A, and set of sequents Ξ, we define $A.\Xi$ to stand for the result of adding A to the antecedent of each element of Ξ.

Due to our choice of presentation of the sequent calculi, with implicit weakening at the leaves, the following property is not trivial, though it is not surprising. It is used in the proof of the Permutability Theorem (which follows).

Lemma 3.7 (Weakening Lemma) *For the systems* LJ, LJ+, LCC, LCC+, *if there is a proof of $\Gamma \vdash \Delta$ from assumptions Σ, then there is a proof of $\Gamma, A \vdash \Delta$ from assumptions $A.\Sigma$.*

This may be proven by induction on the depth of the proof.

The rest of this section is devoted to proofs of sequents arising from the operational semantics of cc languages. Figure 3 defines the logical reading of cc programs, agents, and configurations.

Theorem 3.8 *For any cc program P, agent A and test T, there is a proof of $[\![P]\!], [\![A]\!] \vdash_{\mathrm{LJ+}} [\![T]\!]$ from assumptions Σ iff there is a proof of $[\![A]\!] \vdash_{\mathrm{LCC+}} [\![T]\!]$ from assumptions Σ and program $[\![P]\!]$.*

Proof 3.8 The only difference between these systems are the omitted rules: $(\vdash \neg)$, $(\neg \vdash)$ and $(\vdash \exists)$, and the treatment of P as a set of axioms. Thus if there is a proof in LCC+, by Lemma 3.7, all the formulas in $[\![P]\!]$ may be added in. The resulting proof is already a proof Θ in LJ+. If there is any proof in LJ+ then we normalize cuts in Θ as in Theorem 3.4 to obtain the proof Θ'. By the subformula property, Corollary 3.5, we immediately see that $(\vdash \neg)$ and $(\neg \vdash)$ cannot apply anywhere in Θ'. Further if $\vdash \exists$ applies anywhere in the proof, then by Polarity preservation, there must be some formula $(\exists x : A)$ with positive polarity in the conclusion $P, A \vdash_{\mathrm{LJ}} T$ of Θ'. However, by inspection of the construction of P, A, and T, no such formula is allowed. □

Agents A and programs P are translated into formulas $[\![A]\!]$ and $[\![P]\!]$ thus:

$$\begin{array}{rcl@{\qquad}rcl}
[\![c]\!] & \stackrel{def}{=} & c & & & \\
[\![c \rightarrow A]\!] & \stackrel{def}{=} & [\![c]\!] \supset [\![A]\!] & [\![H \rightarrow A]\!] & \stackrel{def}{=} & H \supset [\![A]\!] \\
[\![H]\!] & \stackrel{def}{=} & H & [\![P_1 \wedge P_2]\!] & \stackrel{def}{=} & [\![P_1]\!] \wedge [\![P_2]\!] \\
[\![A_1 \wedge A_2]\!] & \stackrel{def}{=} & [\![A_1]\!] \wedge [\![A_2]\!] & [\![\forall x.P]\!] & \stackrel{def}{=} & \forall x : [\![P]\!] \\
[\![\exists x.A]\!] & \stackrel{def}{=} & \exists x : [\![A]\!] & & &
\end{array}$$

A program P is represented as an axiom $\vdash [\![P]\!]$. Goals or tests can be read as atomic formulas directly, so no explicit translation is necessary. A configuration $u : G$ is read as the sequent $[\![u]\!] \vdash [\![G]\!]$, where $[\![u_1, \ldots, u_k]\!] = [\![u_1]\!], \ldots, [\![u_k]\!]$ ($[\![A]\!]$ is already defined above), and $[\![c]\!] = c$.

Figure 3: Translation of cc programs, agents and configurations.

Theorem 3.9 (LCC+ **Cut Normalization)** *There is a proof of* $\Gamma \vdash_{\mathrm{LCC+}} \Delta$ *from assumptions* Σ *if and only if there is a proof of* $\Gamma \vdash$ LCC+Δ *from assumptions* Σ*, where every application of (Cut) has a sequent in* Σ *as at least one hypothesis.*

Proof 3.9 Analogous to the proof cut-normalization for LJ+.

□

Permutability theorem. The following permutability theorem shows an important property of the process of proof search in intuitionistic logic, at least for the class of formulas of interest here.

Theorem 3.10 (Permutability Theorem) *Propositional (non-quantifier) inferences in cut-normal* LCC+ *proofs of sequents* $[\![P]\!], [\![A]\!] \vdash [\![T]\!]$ *may be permuted as long as the permutation would not violate the subformula property or the side conditions on quantifier instantiation, and does not involve* $(\supset\vdash)$ *moving below* $(\vdash\supset)$*,* $(\supset\vdash)$ *moving below* $(\vee\vdash)$*, or* $(\vdash\vee)$ *moving below* $(\vee\vdash)$*.*

Proof 3.10 (Sketch) One may refer to [Kle52b] for a list of all the intuitionistic impermutabilities. Most propositional impermutabilities do not occur in the proofs in question, since some inference rules of LJ are not present in LCC+. The only remaining cases are explicilty excluded by the proposition. □

Example 3.1 Any $(\vee\vdash)$ inference may be permuted below any $(\vdash\supset)$. In the following diagram, $\Theta_1, \ldots, \Theta_5$ stand for elided proofs. Proofs Θ_4 and Θ_5 may be constructed from proof Θ_1 using Lemma 3.7

$$\dfrac{\begin{array}{c}\Theta_1\\ \vdots \\ \Gamma, C \vee D \vdash A\end{array} \qquad \dfrac{\begin{array}{c}\Theta_2\\ \vdots \\ \Gamma, A \supset B, B, C \vee D, C \vdash \Delta\end{array} \qquad \begin{array}{c}\Theta_3\\ \vdots \\ \Gamma, A \supset B, B, C \vee D, D \vdash \Delta\end{array}}{\Gamma, A \supset B, B, C \vee D \vdash \Delta}\,(\vee\vdash)}{\Gamma, A \supset B, C \vee D \vdash \Delta}\,(\supset\vdash)$$

This proof tree may be transformed as follows:

$$\dfrac{\dfrac{\begin{array}{c}\Theta_4\\ \vdots \\ \Gamma, C \vee D, D \vdash A\end{array} \qquad \dfrac{\begin{array}{c}\Theta_3\\ \vdots \\ \Gamma, A \supset B, B, C \vee D, D \vdash \Delta\end{array}}{\Gamma, A \supset B, C \vee D, D \vdash \Delta}\,(\supset\vdash)}{\Gamma, A \supset B, C \vee D \vdash \Delta} \qquad \dfrac{\dfrac{\begin{array}{c}\Theta_5\\ \vdots \\ \Gamma, C \vee D, C \vdash A\end{array} \qquad \dfrac{\begin{array}{c}\Theta_2\\ \vdots \\ \Gamma, A \supset B, B, C \vee D, C \vdash \Delta\end{array}}{\Gamma, A \supset B, C \vee D, C \vdash \Delta}\,(\supset\vdash)}{\Gamma, A \supset B, C \vee D \vdash \Delta}}{\Gamma, A \supset B, C \vee D \vdash \Delta}\,(\vee$$

However the opposite transformation is not always possible. Consider the sequent

$$A \vee B, A \supset B \vdash B$$

A cut-free proof of this formula must begin with an application of $\vee\vdash$. Thus this theorem explicitly excludes this case. □

Simple proofs. We recall the earlier definition of simple proofs. A cut-free proof is simple if in every application of the $(\supset\vdash)$ rule, the left hypothesis is proven with axioms only.

Theorem 3.11 *If there is any proof of* $[\![P]\!], [\![A]\!] \vdash_{\mathrm{LCC+}} [\![T]\!]$ *then there is a simple proof.*

Proof 3.11 (Sketch) First, we normalize cuts (Theorem 3.9) Then by repeated application of the Permutability Theorem, we may move inferences in the left hypothesis of any proof ending in $(\supset\vdash)$ that does not violate the subformula property. Since the only formulas $A \supset B$ of negative polarity which appear in $[\![P]\!]$ or in $[\![A]\!] \vdash [\![T]\!]$ are of the form $c \supset A$ or $a \supset A$, by the Polarity Lemma we know that the principal formula of every $(\supset\vdash)$ rule is either of the form $c \supset A$ or $a \supset A$. Since both c and a are atomic formulas, they can only be principal in axioms. Thus we may permute every inference except one axiom (or assumptions) on the left branch below the application of $(\supset\vdash)$. □

For the next lemma we focus on a special form of goals. We also recall the definition of uniform proofs. A cut-free proof is uniform if every sequent with nonatomic succedent is the conclusion of some right-introduction rule.

Lemma 3.12 *If there is any proof of* $[\![P]\!], [\![A]\!] \vdash_{\mathrm{LCC+}} [\![T]\!]$ *then there is a simple uniform proof.*

The proof follows from the previous theorem, and by observing that the the only formulas that appear on the right hand side of sequents are atomic.

3.3 The LK calculus for classical logic

The rules for LK are provided in Figure . As for LJ, the quantifier rules of LK have the same side conditions.

As above, we consider the extension LK+ of LK, with constraint axioms. (Note that the constraint system only proves queries with a single constraint on the right hand side.) Cut-elimination, cut-normalization and the subformula property also hold for LK. In LK+ there are no impermutabilities:

Theorem 3.13 *Inferences in cut-free proofs of* LK+ *formulas may be permuted as long as the permutation would not violate the subformula property, nor the side conditions on quantifier rules.*

We now show that for restricted classes of goals, classical and intuitionistic logic coincide.

Theorem 3.14 $[\![P]\!], [\![A]\!] \vdash_{\mathrm{LJ+}} [\![T]\!]$ *iff* $[\![P]\!], [\![A]\!] \vdash_{\mathrm{LK+}} [\![T]\!]$ *for atomic* T *specified by* $T{:=}c \mid h$.

Proof 3.14 (Sketch) If there is a proof in LJ+ then there is a proof in LK+, since the LJ+ rules are derivable in LK+.

For the other direction, we assume a cut-free LK+ proof, and permute all applications of $(\supset\vdash)$ up the proof as far as they will go. Now we show that an LJ+ proof may be contructed, by induction on the depth of the LK+ proof.

First, note that $[\![T]\!]$ contains a single formula, so if the LK+ proof consists of (I), the proof is an LJ+ proof.

By the subformula property, $(\neg\vdash), (\vdash\neg), (\vdash\vee), (\vdash\forall), (\vdash\exists), (\vdash\supset), (\vdash\wedge)$ do not appear in the LK+ proof.

Since cut has been eliminated, by inspection we see that the only remaining rule which does not preserve the number of formulas in the right hand side of the sequent is $(\supset\vdash)$. In an application of this rule, the right hand side

$$\frac{}{\Gamma, A \vdash A, \Delta}\ (I) \qquad \frac{\Gamma \vdash C, \Sigma \quad \Gamma, C \vdash \Delta}{\Gamma \vdash \Delta, \Sigma}\ (Cut)$$

$$\frac{\Gamma \vdash A, \Delta}{\Gamma, \neg A \vdash \Delta}\ (\neg \vdash) \qquad \frac{\Gamma, A \vdash \neg A, \Delta}{\Gamma \vdash \neg A, \Delta}\ (\vdash \neg)$$

$$\frac{\Gamma, A \supset B \vdash A, \Delta \quad \Gamma, A \supset B, B \vdash \Delta}{\Gamma, A \supset B \vdash \Delta}\ (\supset \vdash) \qquad \frac{\Gamma, A \vdash B, \Delta}{\Gamma \vdash A \supset B, \Delta}\ (\vdash \supset)$$

$$\frac{\Gamma, A \vee B, A \vdash \Delta \quad \Gamma, A \vee B, B \vdash \Delta}{\Gamma, A \vee B \vdash \Delta}\ (\vee \vdash) \qquad \frac{\Gamma \vdash A, B, \Delta}{\Gamma \vdash A \vee B, \Delta}\ (\vdash \vee)$$

$$\frac{\Gamma, A \wedge B, A, B \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta}\ (\wedge \vdash) \qquad \frac{\Gamma \vdash A, \Delta \quad \Gamma \vdash B, \Delta}{\Gamma \vdash A \wedge B, \Delta}\ (\vdash \wedge)$$

$$\frac{\Gamma, \forall x : A, A[t/x] \vdash \Delta}{\Gamma, \forall x : A \vdash \Delta}\ (\forall \vdash) \qquad \frac{\Gamma \vdash A[y/x], \forall x : A, \Delta}{\Gamma \vdash \forall x : A, \Delta}\ (\vdash \forall)$$

$$\frac{\Gamma, \exists x : A, A[y/x] \vdash \Delta}{\Gamma, \exists x : A \vdash \Delta}\ (\exists \vdash) \qquad \frac{\Gamma \vdash A[t/x], \exists x : A, \Delta}{\Gamma \vdash \exists x : A, \Delta}\ (\vdash \exists)$$

Figure 4: Rules for LK

branch is $\Gamma \vdash c, [\![G]\!]$ or $\Gamma \vdash h, [\![G]\!]$ because the only formulas $S \supset B$ of negative polarity have $S = c$ or $S = h$. Since (a) $[\![G]\!]$ is atomic in this fragment, (b) there are no axioms with multiple principal succedent formulas, and (c) we have permuted $(\supset\vdash)$ up the proof as far as it will go, if there is a proof of $\Gamma \vdash c, [\![G]\!]$ then there is also a proof of either $\Gamma \vdash c$ or $\Gamma \vdash [\![G]\!]$ in LK+. In the former case the application of $(\supset\vdash)$ is already intuitionistically valid. In the latter case, by the Weakening Lemma, we may construct a proof of $\Gamma, c \supset [\![A]\!] \vdash [\![G]\!]$ which never includes two or more formulas on the right. □

Theorem 3.15 $[\![P]\!], [\![A]\!] \vdash_{\mathrm{LJ+}} [\![T]\!]$ *iff* $[\![P]\!], [\![A]\!] \vdash_{\mathrm{LK+}} [\![T]\!]$ *for tests* T *specified by* $T{:=}c \mid T \wedge T \mid \forall x : T$.

Proof 3.15 (Sketch) Similar to the above, noting that conjunctive and universally quantified goals do not interfere with the more general induction. □

4 Connection with operational semantics

We now turn our attention to the main topic of the paper, which is showing the tight connection between computation and (simple) proofs in LCC+. We will show that there is a computation from some configuration $u : G$ to another configuration $u' : G$ iff there is a proof of u from the assumptions embodied in u'. Also, we will show that if u' entails c in the underlying constraint system, then $[\![u]\!] \vdash_{\mathrm{LJ+}} c$.

Lemma 4.1 $s \rightarrow s'$ *only if there is a proof of* $[\![s]\!]$ *from the assumption* $[\![s']\!]$.

Proof 4.1 By induction on the size of the $\longrightarrow$ derivation. Inside the induction we procede by case analysis on the definition of $\longrightarrow$.

Suppose $s \equiv u : G \rightarrow \mathtt{true}$. Then the side condition establishes the following derivation of s:

$$\frac{}{u \vdash G}$$

Suppose $s \equiv u, A_1 \wedge A_2 : G \longrightarrow s'$. Then $s' \equiv u, A_1, A_2 : G$. By inductive hypothesis, there is a derivation of s'. Use $(\wedge\vdash)$ to get a derivation of s.

Similarly, use $(\exists\vdash)$ for the case in which the selected agent is an existential, and $(\supset\vdash)$ for the case in which the selected agent is an ask.

In the case where the selected agent is a procedure call a, and is replaced by the agent A, we must have $a :: A$ is in the program. Hence we can construct the following proof:

$$\frac{\dfrac{}{u, a \vdash a}\,(I) \quad u, [\![A]\!] \vdash G}{u, a \supset [\![A]\!], a \vdash G}\,(\supset\vdash)$$

□

Lemma 4.2 $A : T \longrightarrow^* \mathtt{true}$ *only if* $[\![P]\!], [\![A]\!] \vdash_{\mathrm{LJ+}} [\![T]\!]$

Proof 4.2 By using the previous lemma, and induction on the length of the sequence. □

Note that the derivation of $[\![P]\!], [\![A]\!] \vdash_{\mathrm{LJ+}} [\![T]\!]$ constructed in the proof of the previous theorem is a simple proof.

Theorem 4.3 $A : T \longrightarrow^* \mathtt{true}$ *iff* $[\![P]\!], [\![A]\!] \vdash_{\mathrm{LJ+}} [\![T]\!]$

Proof 4.3 (Sketch) Given the previous lemma, all that remains to be shown is that from a simple cut-normal proof Θ of $[\![P]\!], [\![A]\!] \vdash_{\mathrm{LJ+}} [\![T]\!]$ one can establish $A : T \longrightarrow^* \texttt{true}$. This can be established by induction on the size of Θ. Since the RHS is a simple constraint, the proof will not contain any applications of the $(\vdash \star)$ rules, for $\star$ a logical connective. Instead, it will be built up from applications of $(\supset\vdash), (\exists \vdash), (\wedge \vdash)$. In each case the corresponding rule can be chosen from the operational semantics.

In essence, every simple cut-normal proof of $[\![P]\!], [\![A]\!] \vdash_{\mathrm{LJ+}} [\![T]\!]$ has a long "spine" which involves successive applications of $(\supset\vdash)$, interspersed with simplification of conjunctions and existentials. □

5 Conclusion and Future Work

The cc languages can be extended in a natural fashion – from an operational point of view – to a much richer fragment of intuitionistic logic:

$$\begin{array}{rcl} P & ::= & a \rightarrow A \mid G \rightarrow g \mid \forall X.P \mid P \wedge P \\ A & ::= & c \mid c \ \rightarrow \ A \mid a \mid A \wedge A \mid \exists X.A \mid A \vee A \\ G & ::= & c \mid g \mid G \wedge G \mid G \vee G \mid A \rightarrow G \mid \forall X.G \end{array}$$

Here, one thinks of $A \vee A$ as a *disjunctive* agent, which allows for non-deterministic exploration of multiple branches [JSS91].

The richer notion of tests has a very simple computational interpretation. Atomic formulas g allow for recursive definitions of tests, as in Prolog. These are fairly easy to implement. One needs to add test definitions to the program: that corresponds to adding the production $G \rightarrow g$.

The test $G \wedge G$ allows for *conjunctive* tests – both the tests must be satisfied. Similarly, the test $G \vee G$ allows for disjunctive tests.

The test $A \rightarrow G$ allows for *reactive testing*. This may be thought of thus. This test succeeds if the environment (that is, the collection of other agents running in parallel) has enough information to allow A to pass the test G. This is a very powerful construct, similar to the notion of "deep guards" in concurrent logic programming languages. Such a test may be operationalized by creating a new "subvat", copying the contents of the entire environment into it, and adding A to it. Now if this subcomputation can pass the test G, then the test is considered to have passed. Note that subvats can be nested – this allows an agent to test its environment to arbitrarily deep levels.

Finally, $\forall X.G$ allows for information to be communicated between the agent in a reactive test and the condition, e.g. one may formulate the test: $\forall \mathtt{Y}.\mathtt{nrev}(\mathtt{X}, \mathtt{Y}) \rightarrow \mathtt{Y} = []$. Such a test will succeed only if the environment can establish that $X = []$.

It is currently an open question whether the natural operational semantics for these constructs leads to completeness results of the form shown in this paper for simple constraint tests. Also currently open is the question of an appropriate denotational semantics for such a richer language. The simple closure operator semantics will need to be enriched to take into account recursive reactive tests.

Another direction of extension is the computational interpretation of fragments of linear logic, following the same idea of the left hand side of the sequent specifying the computational system, and the right hand side a "test".

More recently, Dale Miller and colleagues have developed similar ideas through the notion of representing arbitrary transition systems as sequent systems (see e.g. [MMP96]). Here we are not concerned with arbitrary transition systems but rather with a specific transition system, that for concurrent constraint programming. The connection with that work should be carefully examined.

References

[Asp90] A. Asperti. *Categorical topics in Computer Science*. PhD thesis, Dipartimento di Informatica, Universita di Pisa, March 1990.

[dNH84] R. de Nicola and M. Hennessyd. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.

[Gir87] J.Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[Gun85] C. Gunter. *Profinite solutions for recursive domain equations*. PhD thesis, Department of Mathematics, University of Wisconsin-Madison, 1985.

[Hen88] M. Hennessy. *An algebraic theory of processes*. MIT Press, 1988.

[JSS91] R. Jagadeesan, V. Shanbhogue, and V. Saraswat. Angelic non-determinism in concurrent constraint programming. Technical report, System Sciences Lab, Xerox PARC, January 1991.

[Kle52a] S.C. Kleene. *Introduction to Metamathematics*. North-Holland, 1952.

[Kle52b] S.C. Kleene. Permutability of inferences in gentzen's calculi lk and lj. *Memoirs of the American Mathematical Society*, 10, 1952.

[LMSS90] P. Lincoln, J. Mitchell, A. Scedrov, and N. Shankar. Decision problems for propositional linear logic. In *Proceedings of the 31st IEEE Symposium on Foundations of Computer Science*, pages 662–671, 1990.

[Mil89] D. Miller. A logical analysis of modules in logic programming. *Journal of Logic Programming*, 6:79–108, 1989.

[MMP96] R. McDowell, D. Miller, and C. Palamidessi. Encoding transition systems in sequent calculus. In *Proceedings of the 1996 Workshop on Linear Logic*, 1996.

[MOM91] N. Marti-Oliet and J. Meseguer. From petri nets to linear logic through categories: A survey. Technical report, SRI International, April 1991.

[Sar93] Vijay A. Saraswat. *Concurrent Constraint Programming*. Doctoral Dissertation Award and Logic Programming. MIT Press, 1993.

[SRP91] V. Saraswat, M. Rinard, and P. Panangaden. Semantic foundations of concurrent constraint programming. In *Proceedings of the Eighteenth ACM Symposium on Principles of Programming Languages*, 1991.

[Sza69] M.E. Szabo. *The collected papers of Gerhard Gentzen*. North-Holland, 1969.