Constraints for Free in Concurrent Computation

Joachim Niehren, Martin Müller

Constraints for Free in Concurrent Computation*

Joachim Niehren Martin Müller

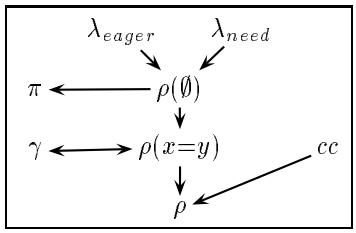
Programming Systems Lab
German Research Center for Artificial Intelligence (DFKI)
Stuhlsatzenhausweg 3, 66123 Saarbrücken, Germany
{niehren,mmueller}@dfki.uni-sb.de

Abstract. We investigate concurrency as unifying computational paradigm which integrates functional, constraint, and object-oriented programming. We propose the ρ-calculus as a uniform foundation of concurrent computation and formally relate it to other models: The ρ-calculus with equational constraints provides for logic variables and is bisimilar to the γ-calculus. The ρ-calculus without constraints is a proper subset of the π-calculus. We prove its Turing completeness by embedding the eager λ-calculus in continuation passing style. The ρ-calculus over an arbitrary constraint system is an extension of the standard cc-model with procedural abstraction.

1 Introduction

Concurrent computation allows the unification of many programming paradigms. This observation underlies Milner's π-calculus [14, 13], Saraswat's concurrent constraint (cc) model [21], and Smolka's γ-calculus [23]. It is also central to the actor model by Hewitt and Agha [1]. Concurrency is the key to the programming language Oz [24] which integrates functional [16], object-oriented [7] and constraint programming [9, 15].

In this paper we start to relate several computational calculi. An overview is given in the picture below. We formulate the relations by comparison with the ρ-calculus [19], a concurrent calculus with first-order constraints, higher-order procedural abstraction, and indeterminism via cells. Any constraint system determines an instance of the ρ-calculus. The ρ-calculus serves as a foundation of the concurrent constraint language Oz [24], is part of its language definition [25] and a basis for its implementation [11].



We prove bisimilarity of the γ-calculus [23] and the calculus $\rho(x{=}y)$: The γ-calculus has been designed to model concurrent objects, while $\rho(x{=}y)$ instanti-

* To appear in Proceedings of the Asian Computer Science Conference ACSC'95, 11–13 december, 1995, Pathumthani, Thailand. Springer LNCS.

ates the ρ-calculus with equational constraints to provide for logic variables. Our bisimulation allows to consider the ρ-calculus as an extension of the γ-calculus with constraints. To obtain this result, we simplified the original ρ-calculus [19]: Now, constraints actually "come for free" in ρ, in contrast to previous extensions of γ with constraints [22, 23, 19, 25].

The ρ-calculus over the trivial constraint system $\rho(\emptyset)$ is a proper subset of the asynchronous polyadic π-calculus [12, 3, 8]. This result is immediate from the identification of procedural abstractions with replicated input agents. Once-only input agents are not available in $\rho(\emptyset)$. Surprisingly, $\rho(\emptyset)$ is still Turing complete: Higher-orderness allows us to embed the eager λ-calculus. A continuation passing style [20] avoids logic variables which have been employed in an earlier embedding of the eager λ-calculus into γ [23, 16].[2] We prove the adequacy of our embedding based on a simulation and uniform confluence [16].[3]

The ρ-calculus is syntactically compositional: Constraints, applications, conditionals, and cells can be freely combined by composition, declaration, and abstraction. The reduction relation of ρ is defined up to a structural congruence, as familiar from recent presentations of π [13, 3, 8] and γ [23]. The central novelty in the version presented here is the distinction of logical conjunction ($\dot{\wedge}$) on constraints from composition ($\wedge$). In the standard cc-model [21, 5, 6], these distinctions hold implicitly due to a monolithic constraint store. In a compositional syntax, the separation of conjunction and composition is central since it yields simple normal forms.

On reduction, applications, cells, or conditionals interact with an arbitrary constraint in their environment, but only one of them. For instance, the conditional $\mathsf{if}\ x{=}y\ \mathsf{then}\ E\ \mathsf{else}\ F\ \mathsf{fi}$ is irreducible in the context of $x{=}1 \wedge y{=}1$, since none of $x{=}1$ or $y{=}1$ entails or disentails $x{=}y$, but reducible in the context of $x{=}1 \mathbin{\dot{\wedge}} y{=}1$ since $x{=}1 \mathbin{\dot{\wedge}} y{=}1 \models x{=}y$. Constraints must be combined explicitly by reduction:

$$\phi_1 \wedge \phi_2 \ \rightarrow\ \phi_1 \mathbin{\dot{\wedge}} \phi_2$$

This combination rule is the essential difference of the ρ-calculus in this paper to its predecessor [19]. It plays the role of elimination in γ where no conjunction is apparent:

$$\exists x \exists y (x{=}y \wedge E) \ \rightarrow\ \exists y E[y/x]$$

The separation of conjunction and composition leads to a transparently distributed constraint store. From this point of view, combination can be interpreted as unification which may or may not involve a network transfer.

Related Work: Most surprisingly, the lazy λ-calculus can be embedded into $\rho(\emptyset)$ with call-by-need complexity (see [17]). Alternatively, the call-by-need λ-calculus [2] could be directly embedded into $\rho(\emptyset)$. Both results are stronger than the analogous results for π [4], since both embeddings map into a uniformly

[2] We owe the idea to personal communication with Gert Smolka and Martin Odersky.

[3] Indeterminism via cells cannot arise in $\rho(\emptyset)$ and is not needed for functional computation.

confluent subset of $\rho(\emptyset)$ and π. Furthermore, Milner's embedding of the lazy λ-calculus into π [13] does not capture call-by-need complexity, and Smolka's call-by-need embedding of the lazy λ-calculus into γ [23] has been given without a correctness proof.

The relationship between the ρ-calculus and the standard cc model [21] is rather simple: The standard cc model can be obtained by dropping higher-order abstractions from ρ and replacing cells with indeterministic sums. A formal proof of this statement would have to deal with distinct notions of constraint systems.

Plan of the paper: In the Sections 2 and 3 we review γ and introduce ρ. Section 4 shows the coincidence of γ and $\rho(x{=}y)$. Section 5 is devoted to the discussion of uniformly confluent subcalculi of γ and ρ. In Section 6, we embed the eager λ-calculus into $\rho(\emptyset)$ and relate the latter with π.

2 The γ-Calculus

Our presentation of γ mainly follows [23] except that we omit names. The omission of names is argued below. We assume an infinite set of *variables* ranged over by x, y, z, u, v, and w. Sequences of variables are denoted by $\overline{x}$, $\overline{y}$, ... The set of elements of a sequences $\overline{x}$ is written as $\mathcal{V}(\overline{x})$. The γ-calculus is specified by *expressions*, a *structural congruence*, and a *reduction* relation. The *expressions* E, F and G are defined by the following abstract syntax:

$$E, F, G ::= x{:}\overline{y}/E \mid x\,\overline{y} \mid E \wedge F \mid \exists x\, E \mid x{=}y \mid \\ \mathsf{if}\ x{=}y\ \mathsf{then}\ E\ \mathsf{else}\ F\ \mathsf{fi} \mid x{:}\,y \mid \top$$

An *abstraction* $x{:}\overline{y}/E$ has a reference x, *formal arguments* $\overline{y}$ and *body* E. An *application* $x\,\overline{y}$ "calls" the abstraction x with *actual arguments* $\overline{y}$. *Composition* $E \wedge F$ puts the expressions E and F in parallel. *Declaration* $\exists x E$ introduces a new local variable x with scope E. *Equations* $x{=}y$ provide for logic variables. Note that conjunction ($\dot{\wedge}$) of equations is *not* apparent in γ. A *conditional* $\mathsf{if}\ x{=}y\ \mathsf{then}\ E\ \mathsf{else}\ F\ \mathsf{fi}$ consists of *guard* $x{=}y$ and *branches* E and F. A *cell expression* $x{:}\,y$ states that a cell referred to by x currently holds the content y. The *null expression* $\top$ is an expression which does nothing, but is useful for defining prenex normal forms.

Bound variables are introduced as formal arguments of abstractions and by declaration. Variables that are not bound are called *free*. $\mathcal{FV}(E)$ and $\mathcal{BV}(E)$ denote the sets of free respectively bound variables in E.

The *structural congruence* $\equiv$ is the least congruence on expressions satisfying the axioms in Figure 1. It provides for the usual properties of bound variables, composition, and declaration. The *reduction* $\rightarrow$ is the least binary relation on expressions satisfying the axioms and rules in Figure 2. We use a generalised replacement operator $[\overline{z}/\overline{y}]$ for simultaneous substitution. Its application implicitly requires that $\overline{z}$ and $\overline{y}$ have equal length and that $\overline{y}$ be linear; i.e., that all elements of $\overline{y}$ be pairwise distinct.

(α)	capture free renaming of bound variables	
(ACI)	$\wedge$ is associative and commutative and satisfies $E \wedge \top \equiv E$.	
(Exch)	$\exists x \exists y E \;\equiv\; \exists y \exists x E$	
(Scope)	$\exists x E \wedge F \;\equiv\; \exists x (E \wedge F)$	if $x \notin \mathcal{FV}(F)$

Fig. 1. Structural Congruence of the γ-Calculus

(A)	$x{:}\overline{y}/E \wedge x\,\overline{z} \;\to\; x{:}\overline{y}/E \wedge E[\overline{z}/\overline{y}]$	if $\mathcal{V}(\overline{z}) \cap \mathcal{BV}(E) = \emptyset$
(E)	$\exists x \exists y (x{=}y \wedge E) \;\to\; \exists y E[y/x]$	if $x \neq y$ and $y \notin \mathcal{BV}(E)$
(Cell)	$x{:}\,y \wedge x\,u\,v \;\to\; x{:}\,u \wedge v{=}y$	
(Then)	if $x{=}x$ then E else F fi $\;\to\; E$	

$$\frac{E \to F}{\exists x E \to \exists x F} \qquad \frac{E \to F}{E \wedge G \to F \wedge G} \qquad \frac{E_1 \equiv E_2 \quad E_2 \to F_2 \quad F_2 \equiv F_1}{E_1 \to F_1}$$

Fig. 2. Reduction of the γ-calculus

Application *(A)* executes procedure calls by passing actual parameters for formal ones. The side condition can always be met by α-renaming. Elimination *(E)* executes equations and may trigger all the other rules. Since elimination requires both variables to be declared, equations are symmetric: By *(α)* we get $\exists x \exists y (x{=}y \wedge E) \to \exists x E[x/y]$, if $x \neq y$ and $x \notin \mathcal{BV}(E)$. When a cell expression $x{:}\,y$ and an application $x\,u\,v$ meet, the value y held by x is replaced by u and v is bound to y *(Cell)*. A conditional reduces on equality of variables *(Then)*. Without names or more expressive constraints than equations, an *(Else)*-rule cannot be formulated. We shall discuss several versions of *(Else)*-rules in the sequel.

Elimination and Application. The simplest example is to apply the identity relation $id{:}\,Id \equiv id{:}x\,y/y{=}x$. Here we apply id to itself and the result to u.

$$\begin{aligned}
&\exists id(id{:}\,Id \wedge \exists z\,(id\,id\,z \wedge z\,u\,out)) \;\equiv\; \exists id \exists z\,(id{:}\,Id \wedge id\,id\,z \wedge z\,u\,out)\\
&\quad \to_A \exists id \exists z\,(id{:}\,Id \wedge z{=}id \wedge z\,u\,out) \;\equiv\; \exists id \exists z\,(z{=}id \wedge id{:}\,Id \wedge z\,u\,out)\\
&\quad \to_E \exists id(id{:}\,Id \wedge id\,u\,out) \to_A \exists id(id{:}\,Id \wedge out = u)
\end{aligned}$$

As done in this example, we shall freely annotate reduction arrows $\to_R$ with the name of the applied axiom R.

Cells. Cells introduce concurrently mutable state into γ by providing a persistent reference to a changeable value (i.e., a location). The indeterminism of cell reduction is needed to express concurrent objects [23]. Cells destroy confluence For example, consider the reduction of $E \equiv x{:}\,y \wedge x\,u_1\,v_1 \wedge x\,u_2\,v_2$:

$$\begin{aligned}
x{:}\,u_2 \wedge x\,u_1\,v_1 \wedge v_2{=}y \;{}_{Cell}{\leftarrow}\; E \to_{Cell} x{:}\,u_1 \wedge v_1{=}y \wedge x\,u_2\,v_2\\
x{:}\,u_1 \wedge v_1{=}u_1 \wedge v_2{=}y \;{}_{Cell}{\leftarrow} \quad \to_{Cell} x{:}\,u_2 \wedge v_1{=}y \wedge v_2{=}u_1
\end{aligned}$$

New Names. Compared to the original γ-calculus [23], our presentation lacks the dynamic creation of new and possibly private names. There, names provide a unique identity to all procedures and cells. The same mechanism conveniently provides a unique reference to concurrent objects. Names can also be used as primitive data structures with a built-in equality and for data encapsulation.

We can add the expressiveness of names orthogonally without affecting any of our results. It is not necessary to provide names with a scope of their own as in [23], but it suffices to let them inherit the scope of the variables. We introduce a new expression n_x and add the additional reduction axiom:

$$\textit{(Else)} \qquad \mathsf{n}_x \wedge \mathsf{n}_y \wedge \mathsf{if}\ x{=}y\ \mathsf{then}\ E\ \mathsf{else}\ F\ \mathsf{fi} \ \rightarrow\ \mathsf{n}_x \wedge \mathsf{n}_y \wedge F\ .$$

An expression $\mathsf{n}_y \wedge \mathsf{n}_y$ would be considered inconsistent since it violates the uniqueness assumption of names. This situation may arise dynamically:

$$\exists x \exists y (x{=}y \wedge \mathsf{n}_x \wedge \mathsf{n}_y) \rightarrow_E \exists y (\mathsf{n}_y \wedge \mathsf{n}_y \wedge E[y/x])$$

Inconsistencies destroy confluence by making $\mathsf{if}\ y{=}y\ \mathsf{then}\ E'\ \mathsf{else}\ F'\ \mathsf{fi}$ reducible via both *(Then)* and *(Else)*.

3 The ρ-Calculus

The ρ-calculus extends γ with constraints. Let Σ be a first-order signature declaring function and relation symbols with their respective arities. A *theory* is a set of closed first-order formulae (with equality) over Σ. A *constraint system* over Σ consists of a *theory* Δ and a set of first-order formulae called *constraints*. Constraints are ranged over by ϕ and ψ. We assume the theory Δ to be *consistent*, i.e. Δ to have a model. The set of constraints must be closed under *replacement* $\phi[y/x]$: for every constraint ϕ the formulae obtained from ϕ by replacing y for x is again a constraint.

We do not require constraints to be closed under *conjunction* (written as $\phi \mathbin{\dot\wedge} \psi$) nor *existential quantification* (denoted by $\dot\exists x \phi$). The formula $\dot\top$ stands for logical *truth*. If Φ_1 and Φ_2 are first-order formulae over Σ, then we write $\Phi_1 \models_\Delta \Phi_2$ iff $\Phi_1 \rightarrow \Phi_2$ is valid in all models of Δ. We write $\Phi_1 \models\!\!\!\models_\Delta \Phi_2$ iff $\Phi_1 \models_\Delta \Phi_2$ and $\Phi_2 \models_\Delta \Phi_1$. If Δ is empty, then we omit the subscript Δ and simply write $\Phi_1 \models \Phi_2$ and $\Phi_2 \models\!\!\!\models \Phi_1$.

The definition of ρ is parameterised by a constraint system. The expressions of ρ are those of γ, but with equations replaced by constraints of the underlying constraint system. The calculus $\rho(x{=}y)$ instantiates ρ with *equational constraints* over the empty theory:

$$\phi, \psi ::= x{=}y \mid \dot\top \mid \phi \mathbin{\dot\wedge} \psi$$

The calculus $\rho(\emptyset)$ instantiates ρ with the trivial constraint system, which is empty up to $\dot\top$. The structural congruence of ρ extends that of γ by the following axiom:

$$\textit{(Equ)} \quad \phi \ \equiv\ \psi \qquad \textit{if } \phi \models\!\!\!\models_\Delta \psi \textit{ and } \mathcal{FV}(\phi) = \mathcal{FV}(\psi)^4$$

Note that we distinguish existential quantification $\dot{\exists} x\,\phi$ from declaration $\exists x\,\phi$ and conjunction $\phi \mathbin{\dot{\wedge}} \psi$ from composition $\phi \wedge \psi$ and that structural congruence preserves this distinction. This is an important technical simplification over [19, 22, 25]. Reduction of ρ is defined by the axioms in Figure 3. As in γ, reduction is allowed in all position but in bodies of abstractions and branches of conditionals.

(Aρ)	$\phi \wedge x{:}\overline{y}/E \wedge x'\,\overline{z} \;\to\; \phi \wedge x{:}\overline{y}/E \wedge E[\overline{z}/\overline{y}]$	if $\phi \models_\Delta x{=}x'$, and $\mathcal{V}(\overline{z}) \cap \mathcal{BV}(E) = \emptyset$
(Cρ)	$\phi_1 \wedge \phi_2 \;\to\; \psi$	if $\phi_1 \mathbin{\dot{\wedge}} \phi_2 \models\!\!\models_\Delta \psi$
(Cellρ)	$\phi \wedge x{:}\,y \wedge x'\,u\,v \;\to\; \phi \wedge x{:}\,u \wedge v{=}y$	if $\phi \models_\Delta x{=}x'$
(Thenρ)	$\phi \wedge \mathsf{if}\ \psi\ \mathsf{then}\ E\ \mathsf{else}\ F\ \mathsf{fi} \;\to\; \phi \wedge E$	if $\phi \models_\Delta \psi$
(Elseρ)	$\phi \wedge \mathsf{if}\ \psi\ \mathsf{then}\ E\ \mathsf{else}\ F\ \mathsf{fi} \;\to\; \phi \wedge F$	if $\phi \models_\Delta \neg\psi$

Fig. 3. Axioms of Reduction of the ρ-Calculus

The axioms *(Aρ)*, *(Cellρ)*, and *(Thenρ)* are triggered by a single constraint of the context if it is sufficiently strong. Combination of constraints *(Cρ)* makes more information available in a single constraint. Compared to γ, *(Cρ)* replaces the elimination rule. By *(Elseρ)*, a conditional may reduce to its else branch if its guard is inconsistent with a constraint of the context. We continue with some examples illustrating computation in ρ.

Combining Constraints. To make a conditional reducible in a given context, an application of rule *(Cρ)* may be necessary. For example, consider:

$$E_1 \;\equiv\; \exists y(\; x{=}1 \wedge y{=}1 \wedge \mathsf{if}\ x{=}y\ \mathsf{then}\ F_1\ \mathsf{else}\ F_2\ \mathsf{fi}\;)$$

The conditional is irreducible because neither $x{=}1 \models x{=}y$, nor $y{=}1 \models x{=}y$ hold. However, E_1 reduces to E_2 by an application of *(Cρ)* :

$$E_2 \;\equiv\; \exists y(\; (x{=}1 \mathbin{\dot{\wedge}} y{=}1) \wedge \mathsf{if}\ x{=}y\ \mathsf{then}\ F_1\ \mathsf{else}\ F_2\ \mathsf{fi}\;)$$

Now, $x{=}1 \mathbin{\dot{\wedge}} y{=}1 \models x{=}y$ holds, such that the conditional can reduce to F_1.

Higher-Order Programming. The following example illustrates the higher-order nature of ρ. Consider a constraint system providing equations and integers with addition. Its signature Σ should contain the constants $\ldots, -1, 0, 1, \ldots$ and the binary function symbol $+$. We allow sugared notation $x\,\overline{n}$ for $\exists\overline{y}(x\,\overline{y} \wedge \overline{y}{=}\overline{n})$, where the $\overline{n}$ are integer symbols. Let us define three abstractions:

$$\begin{array}{ll} \mathit{applytwice}{:}f\,x\,y/\exists z\,(f\,x\,z \wedge f\,z\,y) & \quad \mathit{2times}{:}x\,y/\; y{=}x+x \\ \mathit{4times}{:}x\,y/\; \mathit{applytwice}\,\mathit{2times}\,x\,y & \end{array}$$

[4] Due to the side condition, structural congruence preserves closedness of expressions.

In the context of these abstractions, the expression $4times\,3\,u$ reduces as follows:

$$\begin{array}{l} 4times\,3\,u \;\to_{A\rho}\; applytwice\;2times\;3\,u \;\to_{A\rho}\; \exists z\,(2times\,3\,z \wedge 2times\,z\,u) \\ \quad \to_{A\rho}\; \exists z\,(z{=}3+3 \wedge 2times\,z\,u) \;\equiv\; \exists z\,(z{=}6 \wedge 2times\,z\,u) \\ \quad \to_{A\rho}\; \exists z\,(z{=}6 \wedge u{=}z+z) \;\to_{C\rho}\; \exists z\,(z{=}6 \mathbin{\dot\wedge} u{=}z+z) \\ \quad \equiv\; \exists z\,(z{=}6 \mathbin{\dot\wedge} u{=}12) \end{array}$$

The remaining declaration for z and the local binding $z{=}6$ could be dropped by an appropriate garbage collection rule (see also Example 2 in Section 6.).

New Names. The modelling of names as sketched for γ falls short in really mixing names and constraints. For example, the reduction step $\mathsf{n}_x \wedge \mathsf{n}_y \wedge$ if $f(x){=}f(y)$ then E else F fi $\to F$ is not justified by either of the conditional rules. Mixing names and constraints is important when constraints are used to model data structures holding higher-order data, in particular if they need to be compared for equality [24]. In this case, a closer integration of names and the constraint system is required. An elegant option is to extend the syntax by a declaration construct for names, and to axiomatically require all names to be distinct and different from all elements in the universe of Δ [19, 23].

An alternative approach would take n_x as a constraint for x in a new linear constraint system N, and require N to contain at least the axiom $\forall x \forall y(\mathsf{n}_x \mathbin{\dot\wedge} \mathsf{n}_y \leftrightarrow x \neq y)$. N is linear since $\mathsf{n}_x \mathbin{\dot\wedge} \mathsf{n}_x$ must not be equivalent to n_x. In addition, the entailment relation used in rules *(Aρ)*, *(Thenρ)*, *(Equ)*, etc. must be defined in terms of both first-order entailment and the linear entailment of N. We do not pursue this topic further, since names do not affect our results.

4 Relating the γ-Calculus and the ρ-Calculus

We show that γ can be identified with $\rho(x{=}y)$, when restricted to closed expressions. This statement holds with respect to termination and complexity, measured by the number of application steps.

We need some general notations about computational calculi such as γ, $\rho(x{=}y)$, π [13], or the eager λ-calculus, and lazy λ-calculus. Our notion of a calculus generalises abstract rewrite systems [10]: A *calculus* is a triple $(P, \equiv, \to)$ where P is a set, $\equiv$ an equivalence relation and $\to$ and a binary relation on P. We require a calculus to satisfy $(\equiv \circ \to \circ \equiv) \subseteq \to$, where $\circ$ stands for relational composition. The elements of P are called *expressions*, $\equiv$ *congruence* and $\to$ *reduction*. The least transitive relation containing $\to$ and $\equiv$ is denoted with $\to^*$.

A *derivation* of an expression E is a finite or infinite sequence of expressions $(E_i)_{i=0}^n$ or $(E_i)_{i=0}^\infty$ with $E_i \to E_{i+1}$ for all possible indices i and $E \equiv E_0$. A *computation* of E is a maximal derivation: That is, either an infinite derivation or a finite one whose last element is irreducible with respect to $\to$. Let R be a binary relation on expressions. The *number of R-steps* in a derivation $(E_i)_i$ is the number of indices i such that $(E_i,\, E_{i+1}) \in R$.

Theorem 1. *For every computation of a closed expression E in γ there exists a computation of $\dot\top \wedge E$ in $\rho(x{=}y)$ with the same number of application steps, and vice versa.*

Before we sketch the proof, let us make some additional comments: For every expression in γ and ρ, reduction without application terminates. Reduction with axioms other than *(A)* or *(Aρ)* decreases the number of constraints, applications or conditionals. Hence, Theorem 1 implies that termination of E in γ coincides with termination $\dot\top \wedge E$ in ρ. However, not all derivations of E need to have the same termination behaviour, due to cells which explicitly introduce indeterminism.[5]

The proof idea is to define a bisimulation similarly to [13] which establishes a bijection between computations of E in γ and $\dot\top \wedge E$ in $\rho(x{=}y)$. Our definition of an appropriate bisimulation is based on *normal forms*. These are α-standardised *prenex normal forms (PNFs)*. A PNF D is defined by the following grammar:

$B ::= x{:}\overline{y}/D \mid x\,\overline{y} \mid \mathsf{if}\ \phi\ \mathsf{then}\ D_1\ \mathsf{else}\ D_2\ \mathsf{fi} \mid x{:}\,y \mid \phi$	molecules
$C ::= \top \mid B \mid C \wedge C$	chemical solutions
$D ::= C \mid \exists x\, D$	PNFs

We say that D is a *normal form of* E if D is a normal form and $E \equiv D$.

Proposition 2. *Every expression E has some normalform D.*

Proof. Simple. By Axiom *(α)* any expression may be α-standardised, such that declarations can freely be moved outside via *(Scope)*.

Reduction can be decomposed into reduction on normal forms followed by normalisation, i.e., transformation of the result into normal form again. Following the notation in [16], we define $\equiv_2$ as smallest congruence satisfying the axioms *(ACI)* and *(Exch)*. Furthermore, we need a collection of relations for reduction on normal forms, each of which is specified by a single axiom:

$$\exists\overline{u}\,(x{:}\overline{y}/D \wedge x\,\overline{z} \wedge C) \ \bar{\to}_{A_t}\ \exists\overline{u}\,(x{:}\overline{y}/D \wedge D[\overline{z}/\overline{y}] \wedge C)$$
$$\exists\overline{u}\exists x\,(x{=}y \wedge C) \ \bar{\to}_{E_t}\ \exists\overline{u}\,C[y/x] \qquad \text{if } y \in \mathcal{V}(\overline{u})$$
$$\exists\overline{u}\,(x{:}\,y \wedge x\,u\,v \wedge C) \ \bar{\to}_{Cell_t}\ \exists\overline{u}\,(x{:}\,u \wedge v{=}y \wedge C)$$
$$\exists\overline{u}\,(\mathsf{if}\ x{=}x\ \mathsf{then}\ D_1\ \mathsf{else}\ D_2\ \mathsf{fi} \wedge C) \ \bar{\to}_{Then_t}\ \exists\overline{u}\,(D_1 \wedge C)$$

$$\exists\overline{u}\,(\phi \wedge x{:}\overline{y}/D \wedge x'\,\overline{z} \wedge C) \ \bar{\to}_{A\rho_t}\ \exists\overline{u}\,(\phi \wedge x{:}\overline{y}/D \wedge D[\overline{z}/\overline{y}] \wedge C) \quad \text{if } \phi \models_\Delta x{=}x'$$
$$\exists\overline{u}\,(\phi_1 \wedge \phi_2 \wedge C) \ \bar{\to}_{C\rho_t}\ \exists\overline{u}\,(\psi \wedge C) \quad \text{if } \phi_1 \dot\wedge \phi_2 \models\!\!\!\models_\Delta \psi$$
$$\exists\overline{u}\,(\phi \wedge x{:}\,y \wedge x'\,u\,v \wedge C) \ \bar{\to}_{Cell\rho_t}\ \exists\overline{u}\,(\phi \wedge x{:}\,u \wedge v{=}y \wedge C) \quad \text{if } \phi \models_\Delta x{=}x'$$
$$\exists\overline{u}\,(\phi \wedge \mathsf{if}\ \psi\ \mathsf{then}\ D_1\ \mathsf{else}\ D_2\ \mathsf{fi} \wedge C) \ \bar{\to}_{Then\rho_t}\ \exists\overline{u}\,(\phi \wedge D_1 \wedge C) \quad \text{if } \phi \models_\Delta \psi$$
$$\exists\overline{u}\,(\phi \wedge \mathsf{if}\ \psi\ \mathsf{then}\ D_1\ \mathsf{else}\ D_2\ \mathsf{fi} \wedge C) \ \bar{\to}_{Else\rho_t}\ \exists\overline{u}\,(\phi \wedge D_2 \wedge C) \quad \text{if } \phi \models_\Delta \neg\psi$$

[5] Other expressions should *not* incur indeterminism, or at least allow to detect indeterministic usage as a programming error (cf. admissibility in Section 5).

Proposition 3. *Let D be a normal form, E an expression, and R one of the axioms of reduction in γ or ρ. If $D \to_R E$ then $D\ (\equiv_2 \circ \bar{\to}_{R_t} \circ \equiv)\ E$.*

Proof. Hard technical work. A complete proof in the case of γ can be found in [16]. It adapts immediately to ρ. The most difficult part is the comparison of congruent normal forms. There we need that the congruences of γ and ρ are both rather simple and similar to each other. More complex congruences can be handled but they incur undue technical problems [19].

Definition 4 Bisimulation. A *bisimulation* for the embedding $E \mapsto \dot{\top} \wedge E$ is a relation S between closed expressions of γ and $\rho(x{=}y)$ satisfying the conditions:

B1. If E is a closed γ-expression, then $(E,\ \dot{\top} \wedge E) \in S$.

B2. If $(E,\ F) \in S$, R an axiom of $\{A, Cell, Then\}$, and $E \to_R E'$, then there exists F' such that $F \to_C^* \circ \to_{R\rho} F'$ and $(E',\ F') \in S$.

B3. If $(E,\ F) \in S$ and $E \to_E E'$, then exists F' with $F \to_{C\rho}^* F'$ and $(E',\ F') \in S$.

B4. If $(E,\ F) \in S$, R an axiom of $\{A, Cell, Then\}$ and $F \to_{R\rho} F'$, then there exists E' such that $E \to_E^* \circ \to_R E'$ and $(E',\ F') \in S$.

B5. If $(E,\ F) \in S$ and $F \to_{C\rho} F'$, then exists E' with $E \to_E^* E'$ and $(E,\ F') \in S$.

B6. If $(E,\ F) \in S$, then F is irreducible with respect to $\to_{Else\rho}$.

Definition 5. We define S^γ to be the set of all pairs $(E,\ F)$ of closed γ and $\rho(x{=}y)$ expressions which allow the following representation: There exists variables $\overline{x}\overline{y}$, natural numbers n, m, and k, equations $\phi_1 \ldots \phi_n$, equational constraints $\psi_1 \ldots \psi_m$, molecules $E_1 \ldots E_k, F_1 \ldots F_k$ neither of which is a constraint, and a substitution $\theta : \mathcal{V}(\overline{y}) \to \mathcal{V}(\overline{x})$ such that the following conditions hold:

$S^\gamma 1$. $\exists \overline{x}(\phi_1 \wedge \ldots \wedge \phi_n \wedge E_1 \wedge \ldots \wedge E_k)$ is a normal form of E.

$S^\gamma 2$. $\exists \overline{y} \exists \overline{x}(\psi_1 \wedge \ldots \wedge \psi_m \wedge F_1 \wedge \ldots \wedge F_k)$ is a normal form of F and $m \geq 1$.

$S^\gamma 3$. $\psi_1 \dot{\wedge} \ldots \dot{\wedge} \psi_m \models \theta$, where θ is considered as $\dot{\bigwedge}\{z{=}\theta(z) \mid z \in \mathcal{V}(\overline{y})\}$.

$S^\gamma 4$. $E_i = F_i\theta$ for all $i, 1 \leq i \leq k$,

$S^\gamma 5$. $\phi_1 \dot{\wedge} \ldots \dot{\wedge} \phi_n \models\!\!\!| \exists \overline{y}(\psi_1 \dot{\wedge} \ldots \dot{\wedge} \psi_m)$.

Lemma 6. *S^γ is a bisimulation for the embedding $E \mapsto \dot{\top} \wedge E$.*

Proof. The conditions of a bisimulation can be checked one by one. Condition $B1$ is a consequence of Proposition 2 which ensures the existence of a normal form for E. A normal form for $\dot{\top} \wedge E$ can be easily constructed from that of E.

We exemplarily prove one of the other cases, say $B3$ (all other cases are similar). Assume $(E,\ F) \in S^\gamma$ with the properties described in Definition 5 and E' such that $E \to_E E'$. Since $\exists \overline{x}(\phi_1 \wedge \ldots \wedge \phi_n \wedge E_1 \wedge \ldots \wedge E_k)$ is a normal form of

E, Proposition 3 applies. We can assume w.l.o.g. that ϕ_1 has been eliminated. Hence, there exist distinct u, v and $\overline{x'}$ with $\phi_1 \equiv u{=}v$, $\overline{x} = \overline{x'}u$, $v \in \mathcal{V}(\overline{x'})$, and:

$$E' \;\equiv\; \exists\overline{x'}(\phi_2[v/u] \wedge \ldots \wedge \phi_n[v/u] \wedge E_1[v/u] \wedge \ldots \wedge E_k[v/u]) \;.$$

We define $F' \equiv \exists\overline{y}u\exists\overline{z}v(\psi_1 \mathbin{\dot\wedge} \ldots \mathbin{\dot\wedge} \psi_m \wedge F_1 \wedge \ldots \wedge F_k)$. Obviously, $F \to^*_{C\rho} F'$, such that it is sufficient to prove $(E',\, F') \in S^\gamma$. Clearly, E' and F' have normal forms as required in $S^\gamma 1.$ and $S^\gamma 2.$ We define a substitution $\theta' : \mathcal{V}(\overline{y}u) \to \mathcal{V}(\overline{x'})$ by $\theta' = [v/u] \circ \theta$ (first θ then [v/u]). For short, we write $\psi = \psi_1 \mathbin{\dot\wedge} \ldots \mathbin{\dot\wedge} \psi_m$.

S^γ**3.** Assumptions $\psi \models \theta$ and $\exists\overline{y}\psi \models\!\models \dot\bigwedge_{i=1}^n \phi_i$ imply $\psi \models \phi_1 \wedge \theta \models\!\models u{=}v \wedge \theta \models\!\models \theta'$.

S^γ**4.** $E_i[v/u] = (F_i\theta)[v/u] = F_i([v/u] \circ \theta) = F_i\theta'$ for $1 \le i \le k$.

S^γ**5.** $(\dot\bigwedge_{i=2}^n \phi_i)[v/u] \models\!\models \exists u(u{=}v \mathbin{\dot\wedge} \dot\bigwedge_{i=2}^n \phi_i) \models\!\models \exists u(\dot\bigwedge_{i=1}^n \phi_i) \models\!\models \exists\overline{y}u\psi$

Proof of Theorem 1. The theorem follows from the existence of a bisimulation (Lemma 6). If $(E_i)_{i=0}^n$ is a finite derivation of E and $(E,\, F) \in S^\gamma$, then we can inductively construct a derivation $(F_i)_{i=0}^n$ with $(E_n,\, F_n) \in S^\gamma$ such that both sequences have the same number of application steps. We even get the same result for reduction with cells and conditionals. If $(E_i)_{i=0}^n$ is maximal, then $(F_i)_{i=0}^n$ must be maximal. Otherwise, we could contradict maximality of $(E_i)_{i=0}^n$ by applying our bisimulation the other way around. □

5 Uniformly Concurrent Subcalculi

Functional programming is a special form of concurrent programming [17]. Result, termination, and complexity of functional programs are independent of the execution order. For eager functional programming this is reflected by the eager λ-calculus, and for lazy functional programming by the call-by-need λ-calculus [2]. Concurrent computation satisfying the above three independence properties is called *uniformly concurrent* in [16]. We consider complexity and uniformity, since these notions allow for simple adequacy proofs of calculi embeddings.

A major advantage of γ is the existence of a uniformly concurrent subcalculus which can be easily distinguished. This subcalculus is called δ and has been introduced and investigated in [16]. By Theorem 1, we can carry over most properties from γ to $\rho(x{=}y)$ and conversely. Termination and complexity in terms of application steps correspond exactly. But our bisimulation is *not* strong enough for carrying over confluence or relating the numbers of elimination and combination steps. In this section, we show how to distinguish a uniform and confluent part of ρ over an arbitrary constraint system. This can be done in analogy to the extraction of δ out of γ.

We need some general properties of computational calculi. All proofs are feasible with standard methods and can be found in [16]. A calculus $(P,\, \equiv,\, \to)$ is called *uniformly confluent*, if it satisfies the following condition:

$$(\leftarrow \circ \to) \quad\subseteq\quad (\equiv \;\cup\; (\to \circ \leftarrow))$$

A uniformly confluent calculus is confluent and uniform with respect to termination and complexity: all computations of the same expression have the same length.

Uniform confluence is a compositional property. If $(P, \equiv, \rightarrow_1)$ and $(P, \equiv, \rightarrow_2)$ are two calculi with commuting reductions, i.e. $({}_1\!\leftarrow \circ \rightarrow_2) \subseteq (\rightarrow_1 \circ {}_2\!\leftarrow)$, then the union $(P, \equiv, \rightarrow_1 \cup \rightarrow_2)$ is uniformly confluent.

We now restrict ρ to a uniformly concurrent calculus. We call an expression E *inconsistent*, if E contains a constraint equivalent to $-$ or a subexpression as:

$$\phi \wedge x{:}\overline{y}/E \wedge x'{:}\overline{z}/F \qquad \text{such that } \phi \models_\Delta x{=}x'.$$

E is called *consistent* iff it is not inconsistent. We call an expression E *admissible*, if all expressions F with $E \rightarrow^* F$ are consistent. It is clear that admissility is an undecidable property. But we can characterise admissible subsets by type systems [16]. An appropriate type system for eager functional programming is given in Section 6.

Theorem 7. *The restriction of ρ to admissible expressions without cells is uniformly confluent.*

Proof. It is sufficient to establish the uniform confluence for all relations $\rightarrow_{R\rho}$ where R is in $\{A, C, Then, Else\}$, such as their pairwise commutation. These problems can be reduced to a collection of critical pairs of normal forms.

We exemplify the proof of the uniform confluence of $\rightarrow_{C\rho}$. If $F_1 \; {}_{C\rho}\!\leftarrow E \rightarrow_{C\rho} F_2$ then we can apply Proposition 3. This yields normal forms E_1 and E_2 of E with $F_1 \; {}_{C\rho_t}\!\overset{-}{\leftarrow} E_1 \equiv_2 E_2 \overset{-}{\rightarrow}_{C\rho_t} F_2$. By definition of $\overset{-}{\rightarrow}_{C\rho_t}$ there are $\overline{u}$, ϕ_1, ϕ_2, ψ_1, ψ_2, C_1, and C_2 such that:

$$E_1 \equiv_2 \exists\overline{u}(\phi_1 \wedge \psi_1 \wedge C_1) \overset{-}{\rightarrow}_{C\rho_t} \exists\overline{u}(\phi_1 \dot{\wedge} \psi_1 \wedge C_1) \equiv F_1$$
$$E_2 \equiv_2 \exists\overline{u}(\phi_2 \wedge \psi_2 \wedge C_2) \overset{-}{\rightarrow}_{C\rho_t} \exists\overline{u}(\phi_2 \dot{\wedge} \psi_2 \wedge C_2) \equiv F_2$$

Lets first treat the case $\phi_1 = \phi_2$. If furthermore $\psi_1 = \psi_2$, then $F_1 \equiv F_2$. If $\psi_1 \neq \psi_2$ then there exists C' such that $C_1 \equiv_2 \psi_2 \wedge C'$ and $C_2 \equiv_2 \psi_1 \wedge C'$. Hence:

$$F_1 \rightarrow_{C\rho} \exists\overline{u}(\phi_1 \dot{\wedge} \psi_1 \dot{\wedge} \psi_2 \wedge C') \; {}_{C\rho}\!\leftarrow F_2$$

Case $\phi_1{\neq}\phi_2$ and $\psi_1{=}\psi_2$ is symmetric and case $\phi_1{\neq}\phi_2$ and $\psi_1{\neq}\psi_2$ is similar.

6 The Eager λ-Calculus, ρ-Calculus, and π-Calculus

The ρ-calculus over the trivial constraint system $\rho(\emptyset)$ is Turing complete. We prove this statement by embedding the eager λ-calculus such that termination and complexity are preserved. This embedding employs a continuation passing style (CPS) [20]. An even simpler embedding can be found in [17]. Both embeddings lift into π, since $\rho(\emptyset)$ can be embedded into the asynchronous, polyadic

π-calculus [12, 3, 8]. It suffices to observe the close syntactic similarity of both calculi:

$$\begin{array}{l} P,Q ::= P|Q \quad \mid (\nu x)\, P \mid \overline{x}\langle \overline{y} \rangle \mid\, !x(\overline{y})\,.P \mid \mathsf{0} \\ E,F ::= E \wedge F \mid \quad \exists x\, E \quad \mid \; x\,\overline{y} \quad \mid \quad x{:}\overline{y}/E \quad \mid \top \mid \dot{\top} \end{array}$$

Application in $\rho(\emptyset)$ corresponds to communication and replication in π. Cells can be ommitted, since $(C\rho)$ is not applicable in $\rho(\emptyset)$ as it refers to equations.

The expressions of the *eager λ-calculus* λ_e are defined by the abstract syntax:

$$V ::= \lambda x.M \qquad\qquad M,N ::= x \mid V \mid MN$$

A variable x is bound in $\lambda x.M$, and there is no other variable binder. Free and bound variables of an expression M ($\mathcal{FV}(M)$ and $\mathcal{BV}(M)$, resp.) are defined as usual. The congruence $\equiv$ of λ_e identifies λ-expressions up to α-renaming. Its reduction $\to_{\beta_e}$ is defined as follows:

$$(\lambda x.M)\, V \;\to_{\beta_e}\; M[V/x] \qquad \text{if } \mathcal{FV}(V) \cap \mathcal{BV}(M) = \emptyset$$

$$\frac{M \;\to_{\beta_e}\; M'}{MN \;\to_{\beta_e}\; M'N} \qquad \frac{N \;\to_{\beta_e}\; N'}{MN \;\to_{\beta_e}\; MN'} \qquad \frac{M \equiv M' \quad M' \to_{\beta_e} N' \quad N' \equiv N}{M \;\to_{\beta_e}\; N}$$

In Figure 4, we define our CPS-embedding introducing formulae $u(\!|M|\!)$ as abbreviation for $\rho(\emptyset)$-expressions. These reads as: "u is the continuation applied to the eventual result of evaluating M".

$$\begin{array}{lll} u(\!|x|\!) & \equiv u\,x & \\ u(\!|\lambda x.M|\!) & \equiv \exists y\,(u\,y \wedge y{:}x\,v/v(\!|M|\!)) & uy \text{ linear} \\ u(\!|MN|\!) & \equiv \exists v \exists w\,(v(\!|M|\!) \wedge w(\!|N|\!) \wedge v{:}y/w{:}z/y\,z\,u) & uvw \text{ linear} \end{array}$$

Fig. 4. CPS Embedding

Example 1. We show the reduction of the expression $u(\!|II|\!)$, where $I \equiv \lambda x.x$ is the λ-identity. For brevity, we allow some notation for anonymous ρ-abstractions: For instance, we write $Id = x\,v/v\,x$ for the anonymous CPS-identity and $y{:}Id$ for the CPS-identity named by y, i.e. for the ρ-expression $y{:}x\,v/v\,x$.

We shall additionally use the anonymous abstractions $V = y/w{:}W$ and $W = z/y\,z\,u$. Then we have the following computation in the context of $\dot{\top}$:

$$\begin{array}{rcl} u(\!|II|\!) & \equiv & \exists v \exists w\,(\exists y(\,v\,y\, \wedge y{:}Id) \wedge w(\!|I|\!) \wedge v{:}y/w{:}z/y\,z\,u) \\ & \to_{A\rho} & \exists y \exists w\,(y{:}Id \wedge w(\!|I|\!) \wedge w{:}z/y\,z\,u) \wedge \exists v\,(v{:}V) \\ & \equiv & \exists y \exists w\,(y{:}Id \wedge \exists z\,(\,w\,z\, \wedge z{:}Id) \wedge w{:}z/y\,z\,u) \wedge \exists v\,(v{:}V) \\ & \to_{A\rho} & \exists y \exists z\,(y{:}Id \wedge z{:}Id \wedge \,y\,z\,u\,) \wedge \exists v \exists w\,(v{:}V \wedge w{:}W) \\ & \to_{A\rho} & \exists z\,(u\,z\, \wedge z{:}Id) \wedge \exists v \exists w \exists y\,(v{:}V \wedge w{:}W \wedge y{:}Id) \\ & \equiv & u(\!|I|\!) \wedge \exists v \exists w \exists y\,(v{:}V \wedge w{:}W \wedge y{:}Id) \end{array}$$

Theorem 8. *Eager reduction of a closed λ-expression M terminates if and only if reduction of $\dot{\top} \wedge u(\!|M|\!)$ terminates in $\rho(\emptyset)$. The number of β_e steps in computations of M equals 3 times the number of $(A\rho)$ steps in those of $\dot{\top} \wedge u(\!|M|\!)$.*

The proof is based on a simulation plus uniformity instead of a bisimulation. Given a computation of M in λ_e, we construct a corresponding computation of $\dot{\top} \wedge u(\!|M|\!)$ in $\rho(\emptyset)$ using a simulation. The converse follows from uniformity, since all computations of an admissible ρ-expression have the same termination and complexity behaviour.

In the rest of this section we prove Theorem 8 in two steps: First, we define a appropriate notion of simulation for an embedding $M \mapsto \dot{\top} \wedge u(\!|M|\!)$ and show the existence of such a simulation. Second, we prove admissibility of embedded expressions $\dot{\top} \wedge u(\!|M|\!)$. This is sufficient for combining the proof as sketched above, because we can apply Theorem 7.

Note that the bisimulation technique of Section 4 is not powerful enough for proving Theorem 8, since our CPS-embedding does not establish a bijection between computations in the λ_e and $\rho(\emptyset)$. For instance, the expression $I\,(II)$ has a unique computation in λ_e, while $\dot{\top} \wedge u(\!|I\,(II)|\!)$ has two distinct computations.

To get rid of the context $\dot{\top}$, we reason in the part of γ bisimilar to $\rho(\emptyset)$, i.e. we use the slightly simpler axiom (A) instead of $(A\rho)$.

Definition 9 Simulation. A *simulation* for an embedding $M \mapsto u(\!|M|\!)$ is a relation $S_u^{\lambda_e}$ between closed λ-expressions and γ-expressions satisfying:

$S_u^{\lambda_e}1$. If M is closed, then $(M,\, u(\!|M|\!)) \in S_u^{\lambda_e}$.

$S_u^{\lambda_e}2$. If $M \to_{\beta_e} M'$ and $(M,\, E) \in S_u^{\lambda_e}$, then there exists E' such that $E \to_A^3 E'$ and $(M',\, E') \in S_u^{\lambda_e}$, where $\to_A^3 = \to_A \circ \to_A \circ \to_A$.

$S_u^{\lambda_e}3$. If M is irreducible in λ_e and $(M,\, E) \in S_u^{\lambda_e}$, then E is irreducible in $\to_A$.

Proposition 10. *For all u there is a simulation for the embedding $M \mapsto u(\!|M|\!)$.*

We omit a detailed proof. Instead, we illustrate the exact correspondence between computations of M and those of $u(\!|M|\!)$ by an example. A precise formalisation of this correspondence would yield the simulation. Such a description can be based on explicit substitutions for λ_e [13] and contexts for γ [16].

Example 2. We consider $Copy = \lambda x.x\,x$ and reduce $Copy\,(I\,I)$. To compare reductions in γ and λ_e, we need to cope with the fact that β_e may copy or annul abstractions. These effects are hidden by our meta notation which makes substitutions explicit.

$$\begin{aligned} Copy\,(I\,I) &\to_{\beta_e} (Copy\,v_1)\ [I/u_1][I/v_1] \\ &\to_{\beta_e} (v_1\,v_1)\ [Copy/u_2][I/u_1][I/v_1] \\ &\to_{\beta_e} v_1\ [Copy/u_2][I/u_1][I/v_1] \end{aligned}$$

In γ, contexts play the rôle of explicit substitutions in λ_e. A *context* is an expression with a hole $\bullet$ which acts as a placeholder: Replacement of an expression E or context T_2 for the hole in a context T_1 is denoted by $T_1[E]$ or $T_1[T_2]$.

In order to get rid of unary (continuation) abstractions introduced by $u(MN)$, we extend reduction with garbage collection (G):

$$\exists \overline{y}(\overline{x}{:}\overline{V}) \ \to_G \ \top \qquad \text{if } \mathcal{V}(\overline{x}) \subseteq \mathcal{V}(\overline{y})$$

As proved in [16], the number of application steps in computations is independent of garbage collection. To ease reading, we use the notation $\langle vwu\rangle$ instead of $v{:}y/w{:}z/y\,z\,u$. The eager computation above corresponds to the computation:

$$\begin{array}{rl} u(Copy\ (I\ I)) \to_A^3 \circ \to_G & T_1[u(Copy\, v_1)] \\ & T_1 = \exists u_1 \exists v_1 (\bullet \wedge u_1{:}I \wedge v_1{:}I) \\ \to_A^3 \circ \to_G & T_1[T_2[u(v_1\ v_1)]] \\ & T_2 = \exists u_2 (\bullet \wedge u_2{:}Copy) \\ \to_A^3 \circ \to_G & T_1[T_2[u(v_1)]] \end{array}$$

We verify the first reduction sequence in detail: Observe that $u(Copy\ (I\ I))$ equals $\exists u_0 \exists v_0\, (u_0(Copy) \wedge v_0(I\ I) \wedge \langle u_0 v_0 u\rangle)$. We reduce the subexpression $v_0(I\ I)$:

$$\begin{array}{rl} v_0(I\ I) \equiv & \exists u_1' \exists v_1' (u_1'(Id) \wedge v_1'(Id) \wedge \langle u_1' v_1' v_0\rangle) \\ \to_A^2 & \exists u_1 (\exists v_1 (u_1{:}Id \wedge v_1{:}Id \wedge u_1\, v_1\, v_0) \wedge \exists u_1' \exists v_1' (\langle u_1' v_1' v_0\rangle \wedge v_1{:}z/u_1\, z\, v_0)) \\ \to_A & \exists u_1 (\exists v_1 (u_1{:}Id \wedge v_1{:}Id \wedge v_0(v_1)) \wedge \exists u_1' \exists v_1' (\langle u_1' v_1' v_0\rangle \wedge v_1{:}z/u_1\, z\, v_0)) \\ \to_G & \exists u_1 \exists v_1 (u_1{:}Id \wedge v_1{:}Id \wedge v_0(v_1)) \\ \equiv & T_1[v_0(v_1)] \end{array}$$

We now obtain the expected derivation:

$$\begin{array}{rl} u(Copy\ (I\ I)) \equiv & \exists u_0 \exists v_0\, (u_0(Copy) \wedge v_0(I\ I) \wedge \langle u_0 v_0 u\rangle) \\ \to_A^3 \circ \to_G & \exists u_0 \exists v_0\, (u_0(Copy) \wedge T_1[v_0(v_1)] \wedge \langle u_0 v_0 u\rangle) \\ \equiv & T_1[\exists u_0 \exists v_0\, (u_0(Copy) \wedge v_0(v_1) \wedge \langle u_0 v_0 u\rangle)] \\ \equiv & T_1[u(Copy\, v_1)] \end{array}$$

Proposition 11. *Every expression $u(M)$ is admissible.*

The proof of this proposition can be done with a linear type system excluding multiple assignment statically. It can be found the in report version of this paper [18].

Proof of Theorem 8. Let $(M_i)_{i=0}^n$ be a computation in λ_e such that M_0 is closed. By Proposition 10 there exists a simulation $S_u^{\lambda_e}$ for $M \mapsto u(M)$. The properties $S_u^{\lambda_e}1$ and $S_u^{\lambda_e}2$ allow the construction of a derivation

$$u(M_0) \to_A^3 E_1 \to_A^3 \ldots \to_A^3 E_n$$

such that $(M_n, E_n) \in S_u^{\lambda_e}$. Since M_n is irreducible, $S^{\lambda_e}3$ implies that E_n is irreducible. Lemma 6 establishes the following computation in $\rho(\emptyset)$:

$$\dot{\top} \wedge u(M_0) \to_{A\rho}^3 \dot{\top} \wedge E_1 \to_{A\rho}^3 \ldots \to_{A\rho}^3 \dot{\top} \wedge E_n$$

We use that combination $\rightarrow_{C\rho}$ is not applicable to expressions containing one constraint only.

A similar construction applies to infinite computations $(M_i)_{i=0}^{\infty}$. Let us consider the converse statements. Suppose d is a computation of $u(M)$ and d' an arbitrary computation of M. As shown above there exists a computation d'' of $u(M)$ 3 times longer than d'. The expression $u(M)$ is admissible (Proposition 11) such that we can apply Theorem 7. This yields that the lengths of d'' and d coincide, and that the length of d is 3 times the length of d'. □

7 Conclusion

We have simplified the ρ-calculus, a syntactically compositional cc-model with procedural abstraction. This allowed us to relate various models for concurrent computation in a single formal framework: The ρ-calculus with equational constraints and the γ-calculus are proved bisimilar. We have embedded the eager λ calculus into ρ without constraints, which is a proper subset of the π-calculus. We have extracted a uniformly concurrent kernel in ρ over an arbitrary constraint system. In particular, this distinguishes a uniformly concurrent part of π which is Turing complete.

Acknowledgments. Motivated by Martin Odersky, the CPS style embedding of the eager λ-calculus grew out of discussions with Gert Smolka. The research reported in this paper has been supported by the Bundesminister für Forschung und Technologie (FTZ-ITW-9105), the Esprit Project ACCLAIM (PE 7195), the Esprit Working Group CCL (EP 6028), and the DFG-Graduiertenkolleg Kognition at the Universität des Saarlandes for the second author.

References

1. Gul Agha. *ACTORS: A Model of Concurrent Computation in Distributed Systems.* The MIT Press, Cambridge, MA, 1986.
2. Z. M. Ariola, M. Felleisen, J. Maraist, M. Odersky, and P. Wadler. A call-by-need lambda calculus. In *Proc. POPL*, pp. 233–246. ACM Press, 1995.
3. G. Boudol. Asynchrony and the π-calculus (note). Rapport de Recherche 1702, INRIA, Sophia Antipolis, France, May 1992.
4. S. Brook and G. Ostheimer. Process semantics of graph reduction. In *Proc. CONCUR*, pp. 238–252, August 1995.
5. F. S. de Boer and C. Palamidessi. A process algebra of concurrent constraint programming. In Krzysztof Apt, ed., *Proc. JICSLP*, pp. 463–477, Cambridge, Massachusetts, 1992. The MIT Press.
6. M. Falaschi, M. Gabbrielli, and C. Palamidessi. Compositional analysis for concurrent constraint programming. In *Proc. LICS*, pp. 210–220. IEEE Computer Society Press, June 1993.

7. M. Henz, G. Smolka, and J. Würtz. Object-Oriented Concurrent Constraint Programming in Oz. In V. Saraswat and P. Van Hentenryck, eds., *Principles and Practice of Constraint Programming*, chapter 2, pp. 27–48. The MIT Press, Cambridge, MA, Cambridge, MA, 1995.
8. K. Honda and N. Yoshida. On Reduction-Based Semantics. In R. K. Shyamasundar, ed., Proc. FST-TCS, Bombay, India, December 1993.
9. Sverker Janson. *AKL - A Multiparadigm Programming Language*. PhD thesis, SICS Swedish Institute of Computer Science, SICS Box 1263, S-164 28 Kista, Sweden, 1994. SICS Dissertation Series 14.
10. J. W. Klop. Term Rewriting Systems. In S. Abramsky, D. M. Gabbay, and T. S. M. Maibaum, eds., *Handbook of Logic in Computer Science*, volume 2, chapter 2, pp. 2–116. Oxford University Press, 1992.
11. M. Mehl, R. Scheidhauer, and C. Schulte. An Abstract Machine for Oz. In *Proc. PLILP*, LNCS. Utrecht, NL, 9/20–22/95. Springer, Berlin, Germany. To appear.
12. R. Milner. The polyadic π-calculus: A tutorial. In F. L. Bauer, W. Brauer, and H. Schwichtenberg, eds., *Proc. 1991 Marktoberndorf Summer School on Logic and Algebra of Specification*. NATO ASI Series, Springer, Berlin, Germany, 1993.
13. R. Milner. Functions as Processes. *Mathematical Structures in Computer Science*, 2(2):119–141, 1992.
14. R. Milner, J. Parrow, and D. Walker. A Calculus of Mobile Processes, I and II. *Information and Computation*, 100(1):1–40 and 41–77, September 1992.
15. T. Müller, Konstantin Popow, C. Schulte, and J. Würtz. Constraint programming in Oz. DFKI Oz documentation series, DFKI Saarbrücken, Germany, 1994. Documentation and System: http://ps-www.dfki.uni-sb.de.
16. J. Niehren. *Funktionale Berechnung in einem uniform nebenläufigen Kalkül mit logischen Variablen*. Doctoral Dissertation. Universität des Saarlandes, Technische Fakultät, 66041 Saarbrücken, Germany, December 1994. In German.
17. J. Niehren. Functional computation as concurrent computation, 1995. Submitted, http://ps-www.dfki.uni-sb.de/~niehren.
18. Joachim Niehren and Martin Müller. Constraints for Free in Concurrent Computation. Research Report, German Research Center for Artificial Intelligence (DFKI), Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany, September 1995.
19. J. Niehren and G. Smolka. A confluent relational calculus for higher-order programming with constraints. In J.-P. Jouannaud, ed., *Proc. CCL*, LNCS 845, pp. 89–104, Germany, 1994.
20. G. D. Plotkin. Call-by-name, Call-by-value and the λ-Calculus. *Theoretical Computer Science*, 1:125–159, 1975.
21. V. A. Saraswat, M. Rinard, and P. Panangaden. Semantic foundations of concurrent constraint programming. In *Proc. POPL*, pp. 333–352. ACM Press, 1991.
22. G. Smolka. A Calculus for Higher-Order Concurrent Constraint Programming with Deep Guards. Research Report RR-94-03, DFKI, Saarbrücken, Germany, 1994.
23. G. Smolka. A Foundation for Concurrent Constraint Programming. In J.-P. Jouannaud, ed., *Proc. CCL*, LNCS 845, pp. 50–72, Germany, 1994.
24. G. Smolka. An Oz primer. DFKI Oz documentation series, DFKI, Saarbrücken, Germany, 1995. Documentation and System: http://ps-www.dfki.uni-sb.de.
25. G. Smolka. The Definition of Kernel Oz. In Andreas Podelski, ed., *Constraints: Basics and Trends*, LNCS 910, pp. 251–292. Springer, Berlin, Germany, 1995.