Processes in Space

Luca Cardelli[1] and Philippa Gardner[2]

[1]Microsoft Research Cambridge, [2]Imperial College London
[1]luca@microsoft.com, [2]pg@doc.ic.ac.uk

Abstract. We introduce a geometric process algebra based on affine geometry, with the aim of describing the concurrent evolution of geometric structures in 3D space. We prove a relativity theorem stating that algebraic equations are invariant under rigid body transformations.

Keywords: Process Algebra, Affine Geometry

1 Introduction

Process algebras are used to study fundamental primitives of interaction between distributed, concurrent processes. They have been used successfully as a basis of distributed programming languages, and for modeling molecular interaction in biological organisms. In this paper, we introduce a geometric process algebra, called 3π, that combines the interaction primitives of the π-calculus with geometric transformations, in order to extend the applicability of process algebras to geometrical systems.

One of the key motivating examples of the π-calculus is the handover protocol of mobile phones [10]: a mobile phone is connected to a fixed tower, and through the connection receives a new frequency to connect to a different tower. In actuality, the handover is based on the relative distance (relative signal power) between the mobile device and the fixed towers, and hence the protocol depends on geometry. More generally, one can find many examples of interacting processes that are continually moving, like interacting transmitters attached to animals or robots, and interacting cells in biological organisms. Biological systems, in particular, provide complex examples of process interaction that depend on geometry, such as the growth of tissues, the diffusion of signaling molecules, and the overlapping of chemical gradients. During biological development, tissues expand, split and twist, and there is no fixed coordinate system that one can coherently apply. To capture examples such as these, it is natural to turn to *affine geometry*, which is the geometry of properties that are invariant under linear transformations and translations.

Beyond choosing the particular kind of geometry, we must choose how the geometric space relates to the processes that are living within it. How should the position of a process be represented? How should a process move from one position to another? How should processes at different positions interact? In 3π, processes have access to the standard affine basis consists of the origin $\dot{+}$ and the orthogonal unit vectors $\updownarrow_x,\updownarrow_y,\updownarrow_z$. However, all geometric data is interpreted with respect to a global frame $\mathcal{A}$, which is an affine map. In particular, what a process believes to be the origin, $\dot{+}$, is actually $\mathcal{A}(\dot{+})$, and this is seen as the *location* of the process in the global frame. The

true size and orientation of the basis vectors is also influenced by $\mathcal{A}$, as they are interpreted as $\mathcal{A}(\hat{1}_x),\mathcal{A}(\hat{1}_y),\mathcal{A}(\hat{1}_z)$. The global frame $\mathcal{A}$ is inaccessible to processes, although they can carry out comparisons between computed values that may reveal some information about $\mathcal{A}$.

Processes can change position via a *frame shift* operation M[Q], where the local frame M denotes an affine map $\mathcal{B}$: given process M[Q] in a global frame $\mathcal{A}$, then process Q is interpreted in the shifted global frame $\mathcal{A}\circ\mathcal{B}$. The process M[Q] | N[R] therefore indicates that processes Q and R are in different frames, with Q shifted by M and R by N. Conversely, the process M[Q] | M[R] = M[Q | R] indicates that Q and R are in the same frame. Frame shift operations can also be nested, with the process $M[N_1[Q] \mid N_2[R]]$ indicating that Q is in the frame shifted first by N_1 and then M, whereas R is shifted by N_2 then M. Since M denotes a general affine map, frame shift is more than just a change of location: it generalizes the d-π [6] notion of multiple process locations to multiple process frames.

Processes interact by exchanging data messages consisting of channel names or geometric data. Process interaction is not restricted by the frame (position) of a process. Geometric data is evaluated in its current frame and transmitted 'by value' to the receiver. For example, we have the interaction:

$$P \stackrel{\text{def}}{=} M[!x(\cdot\!\cdot).Q] \mid N[?x(z).R] \;\; {}_{\mathcal{A}}\!\rightarrow \;\; M[Q] \mid N[R\{z\backslash\varepsilon\}]$$

where M evaluates to $\mathcal{B}$ in the global frame $\mathcal{A}$ and $\cdot\!\cdot$ evaluates to $\varepsilon = \mathcal{A}\circ\mathcal{B}(\cdot\!\cdot)$. Technically, this interaction across frame shifts is achieved via the equality:

$$P = !x(M[+]).M[Q] \mid ?x(z).N[R]$$

which distributes the frame shifts throughout the process, thus exposing the output and input for interaction. In addition to communication, processes can compare data values. If R is $z{=}\cdot\!\cdot.R'$ in our above example, then after interaction this process computes whether $\mathcal{A}\circ\mathcal{B}(\cdot\!\cdot) = \mathcal{A}\circ C(\cdot\!\cdot)$, where C is the evaluation of N in $\mathcal{A}$, and evolves to R' only if the original output and input processes are at the same position.

Related Work. Affine geometry is the geometry of properties that are invariant under translation, rotation, reflection and stretching. Distances and angles are not necessarily preserved by affine maps, but relative positions are: for example, the notion of midpoint is an affine invariant. Affine geometry is widely used in computer graphics; probably the most accessible reference for computer scientists is Gallier's book [5]. It has also been used in conjunction with L-Systems in very successful models of plant development [11]. However, L-systems are contextual term rewriting systems and, unlike 3π, do not have an intrinsic notion of interaction, a notion which is important since biological development is regulated by sophisticated intra-cellular interactions.

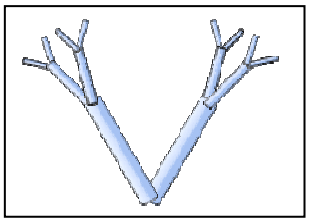
SpacePi [8] has been proposed as an extension of the π-calculus to model spatial dynamics in biological systems. This approach has similar general aims to our work, but is technically rather different: communication is limited to a radius, processes have velocity vectors, time is discrete, the preferred simulation technique is discrete event simulation, and there is no notion of applying a geometric transformation to a process. In future work, we will introduce continuous time in 3π by a stochastic extension, which will allow for Gillespie-style simulation. The velocity of a process then

becomes a consequence of stochastic moves at certain rates. This means that 3π need only use 3D space, because the 4th geometric dimension (time) comes from the intrinsic dynamics of stochastic π-calculus. Unlike SpacePi, we do not restrict communication to a radius because that can be achieved by comparing data values, because some physical forces have infinite radius, and because geometric constraints on interaction are not necessarily of such a simple form (e.g., interaction restricted to adjacent cells of odd shapes).

Example: Distance between processes. Let us assume that the global frame is just the identity map. Process P below is located at -1 on the x axis, because X applies a translation $T(-\updownarrow_x)$ to it. Similarly, process Q is located at +1 on the x axis by Y. When P outputs its origin, the actual value being communicated is thus the point ⟨-1,0,0⟩: this is a *computed value* that is not subject to any further transformation. Process Q receives that value as x, and computes the size of the vector x÷⊹ obtained by a point difference. In the frame of Q that computation amounts to the size of the vector ⟨-1,0,0⟩ ÷ ⟨1,0,0⟩, which is 2. Therefore, the comparison ‖x÷⊹‖=2 succeeds, and process R is activated, having verified that the distance between P and Q is 2.

X = $T(-\updownarrow_x)$[P] where P = !m(⊹)
Y = $T(\updownarrow_x)$[Q] where Q = ?m(x). ‖x÷⊹‖=2. R

Example: Orthogonal bifurcation in lung development. Lung development in mouse is based on three splitting processes [9]. We show how to represent the third (orthogonal bifurcation, Orth), which is a proper 3D process where bifurcations alternate between orthogonal planes.



Orth = !c(⊹). (M90(π/6)[Orth] | M90(-π/6)[Orth])
M90(θ) = R(M(θ)[$\updownarrow_y$],π/2)∘M(θ)
M(θ) = Sc(½)∘R($\updownarrow_z$,θ)∘T($\updownarrow_y$)

The output of the origin ⊹ to the c channel at each iteration provides a trace of the growing process that can be plotted. The transformation M(θ) applies a translation $T(\updownarrow_y)$ by $\updownarrow_y$, a rotation R($\updownarrow_z$,θ) by θ around $\updownarrow_z$, and a uniform scaling Sc(½) by ½. The transformation M90(θ) first applies an M(θ) transformation in the XY plane, and then applies a further 90° rotation around the 'current' direction of growth, which is M(θ)[$\updownarrow_y$], therefore rotating out of the XY plane for the next iteration. Opposite 30° rotations applied recursively to Orth generate the branching structure.

Example: Force fields. A force field is a process that receives the location of an 'object' process (and, if appropriate, a representation of its mass or charge), and tells it how to move by a discrete step. The latter is done by replying to the object with a transformation that the object applies to itself. This transformation can depend on the distance between the object and the force field, and can easily represent inverse square and linear (spring) attractions and repulsions. By nondeterministic interaction with multiple force fields, an object can be influenced by several of them.

Force = (?f(x,p). !x(M{p}))* f is the force field channel; M{p} is a map
Object = (νx) !f(x,⊹). ?x(Y). Y[Object]

A uniform field ('wind'):	$M\{p\} = T(\hat{1}_x)$
A linear attractive field at q ('spring'):	$M\{p\} = T(\frac{1}{2}\cdot(q-p))$
An inverse-square repulsive field at q ('charge'):	$M\{p\} = T((1/\|\|p-q\|\|^2)\cdot(p-q))$

The ability to express force fields is important for modeling constraints in physical systems. For example, by multiple force fields one can set up an arbitrary and time-varying network of elastic forces between neighboring cells in a cellular tissue.

2 Processes

We introduce a process algebra, 3π, where 3-dimensional geometric data (points, vectors, and affine maps) can be exchanged between processes, and where processes can be executed in different frames. This is a proper extension of π-calculus with by-value communication of geometric data Δ, data comparisons Δ=Δ'.P, and frame shifting M[P]. By-value communication is achieved by an evaluation relation $\Delta \ {}_{\mathcal{A}}\!\!\mapsto \varepsilon$, which evaluates a *data term* Δ with respect to a global frame $\mathcal{A}$ to a *data value* ε. Frame shifting is the characteristic construct of 3π: it means running the process P in the frame shifted by affine map M.

The syntax of processes, given in Definition 2.1–1, depends on the syntax of data Δ which is defined in full in Section 3. For now, it is enough to know that data terms include channel variables $x_{\boldsymbol{c}}$ and variables of geometric sorts; we use x (without sort subscript) to denote a generic variable when the sorting can be inferred. Data terms also include M[Δ], representing data Δ transformed by an affine map M. Other data terms (indicated by '...') are discussed in Section 3. Each data term has a *data sort* $\sigma \in \Sigma = \{\boldsymbol{c,a,p,v,m}\}$, denoting channels, scalars, points, vectors, and maps respectively. The channel variables $x_{\boldsymbol{c}} \in Var_{\boldsymbol{c}}$ have sort $\boldsymbol{c}$, and the sort of M[Δ] is the sort of Δ.

2.1–1 Definition: Syntax of Processes

$\Delta ::= x_{\boldsymbol{c}} \mid ... \mid M[\Delta]$	Data terms
$\pi ::= ?_\sigma x(x') \mid !_\sigma x(\Delta) \mid \Delta =_\sigma \Delta'$	Action terms
$P ::= 0 \mid \pi.P \mid P+P' \mid P\|P' \mid (\nu x)P \mid P^* \mid M[P]$	Process terms

An action term π can be an *input* $?_\sigma x(x')$, an *output* $!_\sigma x(\Delta)$, or a *data comparison* $\Delta =_\sigma \Delta'$. The input and output actions are analogous to π-calculus actions, where the input receives a data value of sort σ along channel x which it binds to x', and the output sends the value of Δ with sort σ along x. Process interaction only occurs between inputs $?_\sigma$, and outputs $!_\sigma$ of the same sort σ. A comparison of two data terms of sort σ blocks the computation if the terms do not match when evaluated using $_{\mathcal{A}}\!\!\mapsto$. The syntax of actions is restricted by sorting constraints: the x in $?_\sigma x(x')$ and $!_\sigma x(\Delta)$ has a channel sort $\boldsymbol{c}$; the x' in $?_\sigma x(x')$ must have sort σ; the Δ in $!_\sigma x_{\boldsymbol{c}}(\Delta)$ must have sort σ; and the Δ,Δ' in $\Delta =_\sigma \Delta'$ must have sort σ. We often omit sorting subscripts, and we assume that variables of distinct sorts are distinct.

A process term is a fairly standard π-calculus term, consisting of the empty

process 0, the action process $\pi.P$ for action π, choice P+P', parallel composition P | P', channel restriction $(\nu x)P$ where x has sort $\boldsymbol{c}$, and replication P*. In addition, we have process frame shifting M[P], which is the execution of process P in a shifted frame given by M. We shall see in Section 3 that channel variables do not occur in M; hence in $(\nu x)M[P]$ there then is no possibility that any variable in M is bound by x.

The free and bound channel variables are defined as for the π-calculus: in particular, the variable x in $?_\sigma y(x).P$ and $(\nu x)P$ acts as a binder. We write $fv_\sigma(P)$ to denote the free variables of sort σ in P, and assume α-convertibility. As usual, we define $\tau.P =_{def} (\nu x)(?x(x).0 \mid !x(x).P)$ for $x \notin fv_{\boldsymbol{c}}(P)$. The substitution $P\{x\backslash\varepsilon\}$ for data value ε follows the normal substitution for the π-calculus, with non-standard cases for frame shifted processes, $M[P]\{x\backslash\varepsilon\} = M\{x\backslash\varepsilon\}[P\{x\backslash\varepsilon\}]$, and for action processes, $(\Delta=_\sigma\Delta'.P)\{x\backslash\varepsilon\} = \Delta\{x\backslash\varepsilon\}=_\sigma\Delta'\{x\backslash\varepsilon\}.P\{x\backslash\varepsilon\}$ and $(!_\sigma y_{\boldsymbol{c}}(\Delta).P)\{x\backslash\varepsilon\} = !_\sigma y_{\boldsymbol{c}}(\Delta\{x\backslash\varepsilon\}).P\{x\backslash\varepsilon\}$ and $(?_\sigma y_{\boldsymbol{c}}(z).P)\{x\backslash\varepsilon\} = ?_\sigma y_{\boldsymbol{c}}(z).P\{x\backslash\varepsilon\}$ assuming an α-variant with $z \neq x$. We define the substitution $\Delta\{x\backslash\varepsilon\}$ in Section 3: it is straightforward as Δ contains no variable binding constructs. We say that a term is *closed* if it does not contain free variables of sorts ***a,p,v,m***; the free channel variables evaluate to themselves and so are admitted in the closed terms.

We now give a ternary *reduction* relation on process terms, written $\rightarrow$, which relates two processes and a global frame $\mathcal{A}$. Reduction depends on the evaluation relation $\Delta \ {}_{\mathcal{A}}\!\!\rightarrowtail \varepsilon$ from data Δ to values ε, again in a global frame $\mathcal{A}$. The frame $\mathcal{A}$ is an affine map: any geometric data, such as the origin and the basis vectors, are interpreted with respect to this global frame.

The reduction rules for process terms are just the rules of a by-value π-calculus with data terms Δ. Data evaluation is used in the (Red Comm) and (Red Cmp) rules. Data comparison $\Delta=_\sigma\Delta'.P$ requires the data evaluation $\Delta \ {}_{\mathcal{A}}\!\!\curvearrowright\!\!\curvearrowleft \Delta'$ which means there exists a data value ε such that $\Delta \ {}_{\mathcal{A}}\!\!\rightarrowtail \varepsilon$ and $\Delta' \ {}_{\mathcal{A}}\!\!\rightarrowtail \varepsilon$. Data comparison acquires a specific sense of *observation in the frame* $\mathcal{A}$, because Δ and Δ' may or may not match depending on $\mathcal{A}$. The reduction rules also support channel passing and channel matching in the standard π-calculus sense, because channels evaluate to themselves.

2.1–2 Definition: Reduction

(Red Comm)	$\Delta \ {}_{\mathcal{A}}\!\!\rightarrowtail \varepsilon \ \Rightarrow \ !_\sigma x(\Delta).P + P' \mid ?_\sigma x(y).Q + Q' \ {}_{\mathcal{A}}\!\!\rightarrow \ P \mid Q\{y\backslash\varepsilon\}$
(Red Cmp)	$\Delta \ {}_{\mathcal{A}}\!\!\curvearrowright\!\!\curvearrowleft \Delta' \ \Rightarrow \ \Delta=_\sigma\Delta'.P \ {}_{\mathcal{A}}\!\!\rightarrow P$
(Red Par)	$P \ {}_{\mathcal{A}}\!\!\rightarrow Q \ \Rightarrow \ P \mid R \ {}_{\mathcal{A}}\!\!\rightarrow Q \mid R$
(Red Res)	$P \ {}_{\mathcal{A}}\!\!\rightarrow Q \ \Rightarrow \ (\nu x)P \ {}_{\mathcal{A}}\!\!\rightarrow (\nu x)Q$
(Red ≡)	$P' \equiv P, \ P \ {}_{\mathcal{A}}\!\!\rightarrow Q, \ Q \equiv Q' \ \Rightarrow \ P' \ {}_{\mathcal{A}}\!\!\rightarrow Q'$

There is nothing specific in these rules about the use of the global frame $\mathcal{A}$: this is simply handed off to the data evaluation relation. There is also no rule for process frame shifting, M[P], which is handled next in the structural congruence relation.

In the now standard 'chemical' formulation [1] of π-calculus, the *structural congruence* relation has the role of bringing actions 'close together' so that the communication rule (Red Comm) can operate on them. We extend this idea to bringing actions

together even when they are initially separated by frame shifts, so that the standard (Red Comm) rule can still operate on them. Therefore, structural congruence, ≡ (Definition 2.1–3), consists of the normal π-calculus rules plus additional rules for frame shifting: the (≡ Map...) rules. These map rules essentially enable us to erase frame shifts from the process syntax and to push them to the data. In this sense, process frame shift M[P] is an illusion, or syntactic sugar, for a π-calculus with frame shift only on the data. However, frame shift is important for modularity because, without it, we would have to modify the process code to apply the frame to all the data items individually.

2.1–3 Definition: Structural Congruence

(≡ Refl)	$P \equiv P$	(≡ Sum Comm)	$P+Q \equiv Q+P$
(≡ Symm)	$P \equiv Q \Rightarrow Q \equiv P$	(≡ Sum Assoc)	$(P+Q)+R \equiv P+(Q+R)$
(≡ Tran)	$P \equiv Q, Q \equiv R \Rightarrow P \equiv R$	(≡ Sum Zero)	$P+0 \equiv P$
(≡ Act)	$P \equiv P' \Rightarrow \pi.P \equiv \pi.P'$	(≡ Par Comm)	$P \mid Q \equiv Q \mid P$
(≡ Sum)	$P \equiv P', Q \equiv Q' \Rightarrow P+Q \equiv P'+Q'$	(≡ Par Assoc)	$(P \mid Q) \mid R \equiv P \mid (Q \mid R)$
(≡ Par)	$P \equiv P', Q \equiv Q' \Rightarrow P \mid Q \equiv P' \mid Q'$	(≡ Par Zero)	$P \mid 0 \equiv P$
(≡ Res)	$P \equiv P' \Rightarrow (\nu x)P \equiv (\nu x)P'$	(≡ Res Zero)	$(\nu x)0 \equiv 0$
(≡ Repl)	$P \equiv P' \Rightarrow P^* \equiv P'^*$	(≡ Res Sum)	$(\nu x)(P+Q) \equiv P+(\nu x)Q$ $(x \notin fv_c(P))$
(≡ Map)	$P \equiv P' \Rightarrow M[P] \equiv M[P']$	(≡ Res Par)	$(\nu x)(P \mid Q) \equiv P \mid (\nu x)Q$ $(x \notin fv_c(P))$
(≡ Map Cmp)	$M[\Delta=_\sigma\Delta'.P] \equiv M[\Delta]=_\sigma M[\Delta'].M[P]$	(≡ Res Res)	$(\nu x)(\nu y)P \equiv (\nu y)(\nu x)P$
(≡ Map Out)	$M[!_\sigma x(\Delta).P] \equiv !_\sigma x(M[\Delta]).M[P]$	(≡ Repl Zero)	$0^* \equiv 0$
(≡ Map In)	$M[?_\sigma x(y).P] \equiv ?_\sigma x(y).M[P]$ $(y \notin fv_\sigma(M))$	(≡ Repl Par)	$(P \mid Q)^* \equiv P^* \mid Q^*$
(≡ Map Sum)	$M[P+Q] \equiv M[P]+M[Q]$	(≡ Repl Copy)	$P^* \equiv P \mid P^*$
(≡ Map Par)	$M[P \mid Q] \equiv M[P] \mid M[Q]$	(≡ Repl Repl)	$P^{**} \equiv P^*$
(≡ Map Res)	$M[(\nu x)P] \equiv (\nu x)M[P]$		
(≡ Map Comp)	$M[N[P]] \equiv (M \circ M[N])[P]$		

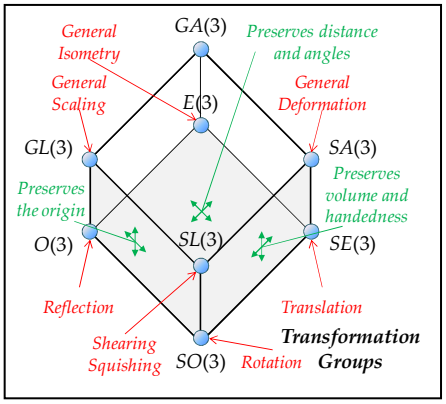
The only non-standard rules are the (≡ Map ...) rules. These can be read from left to right as pushing frames inside the syntax; the only situation that is not generally reversible is with (≡ Map In) because of the side condition. Correctness of the Map rules for Sum, Par, and Res is fairly obvious, because all parts of a given process should be in the same frame. Note that (≡ Map Par) mimics a rule in d-π [6], and that (≡ Map Res) relies on variable sorting so that x cannot occur in M. The rules (≡ Map Out) and (≡ Map In) have the effect of removing a layer of frame shifting around inputs and outputs. The (≡ Map Comp) rule pushes a frame shift across another frame shift, thus flattening the structure and allowing inputs and outputs in different nesting levels to come together; we might expect the rule to be $M[N[P]] \equiv (M \circ N)[P]$, but we need to keep N in its original frame. A rule to push frame shift inside P* is not needed, because (≡ Repl Copy) implies that $M[P^*] \equiv M[P|P^*] \equiv M[P]|M[P^*]$, thus solving the same recursive equation as M[P]*. A rule $M[0] \equiv 0$ is not included: it might introduce an unconstrained M leading to loss of induction hypotheses.

Many other rules can be derived, e.g., for communication across frames shifts at different depths, and for data comparison inside a local frame. In summary, the appli-

cation of the structural congruence rules allows us to 'flatten' the local frames so that the rules in Definition 2.1–2 can be applied directly. There still remains the issue of correctness, or plausibility, of the new structural congruence rules. This issue can be explored by analyzing the expected derived rules, as we briefly mentioned above, and by establishing general properties of the whole system, as done in Section 4.

We have not discussed recursion, which was used in some of the initial examples. However, ordinary recursive definitions in π-calculus can be encoded from replication and communication, and this extends in 3π to recursive definitions under frame shift by the ability to communicate transformations.

3 Data



A *vector space* over a field F is a set V with operations $+ \in V\times V\to V$ (vector addition) and $\cdot \in F\times V\to V$ (scalar multiplication), such that $(V,+)$ is an abelian group, with identity the zero vector ø and inverse $-v$ and: $a\cdot(v+w) = a\cdot v+a\cdot w$, $(a+b)\cdot v = a\cdot v+b\cdot v$, $(a\cdot b)\cdot v = a\cdot(b\cdot v)$, and $1\cdot v = v$. Three-dimensional space is our basic vector space over the field of reals: the vectors are the points of $\mathbf{R}^3$, $+$ is coordinatewise addition, and $\cdot$ is coordinatewise multiplication. *Euclidean spaces* such as $\mathbf{R}^3$ also have the ability to *measure*. This is achieved by extending the vector space with the *dot product* of vectors, $v\bullet w$, giving the ability to measure distances and angles, and with the *cross product* of vectors, $v\times w$, giving the ability to generate out-of-plane vectors, to measure areas and volumes, and to detect handedness.

An *affine space* consists of a set of points P, a vector space V, and for each point p a bijection θ_p between points and vectors giving rise to two operations: $p\dot{-}q = \theta_q(p)$ and $v\dot{+}p = \theta_p^{-1}(v)$. From two points p,q, we can obtain the *free vector* $p\dot{-}q$ from q to p, intended as a vector with magnitude and orientation but without fixed location. Given a vector v and a point p, we obtain the point $v\dot{+}p$, which is the point p translated by the vector v. The key property is the head-to-tail axiom $(p\dot{-}q) + (q\dot{-}r) = (p\dot{-}r)$. Throughout this paper, we use the three-dimensional affine space over $\mathbf{R}^3$, consisting of $\mathbf{R}^3$ as the set of points (including the origin point denoted $\dot{-}$), and $\mathbf{R}^3$ as the vector space (including the standard basis vectors denoted $\mathbf{1}_x,\mathbf{1}_y,\mathbf{1}_z$), and with the bijections $\theta_q(p)$ = the vector from the origin to p-q. (Technically, we take an isomorphic copy of the vector space, so we can distinguish points from vectors in the operational semantics.)

We are interested in three main groups of transformations over $\mathbf{R}^3$. The *General Affine Group* $GA(3)$ is the group of *affine maps* over $\mathbf{R}^3$, which are indicated by script letters $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$. Affine maps are presented as pairs $\langle A,p\rangle$ where A is 3x3 invertible matrix representing a linear transformation, and p is a point in $\mathbf{R}^3$ used as a translation vector. The *Euclidean Group* $E(3)$ is the subgroup of $GA(3)$ where $A^T = A^{-1}$: namely, it is the group of isometries of $\mathbf{R}^3$ consisting of rotations, translations and reflections. The *Special Euclidean Group* $SE(3)$ is the subgroup of $E(3)$ where $\det A = 1$: namely,

the *direct isometries* which exclude reflections. Elements of $SE(3)$ are known as the *rigid body transformations*, preserving distances, angles, and handedness.

An affine map $\mathcal{A}$ has a canonical associated affine frame, namely the frame $\mathcal{A}(\dot{-}),\mathcal{A}(\uparrow_x),\mathcal{A}(\uparrow_y),\mathcal{A}(\uparrow_z)$; we therefore refer to $\mathcal{A}$ itself as a frame. We next introduce geometric data and show how to compute data values with respect to an affine frame. We define five sets of data values, Val_σ, indexed by sorts $\sigma \in \Sigma = \{\boldsymbol{c},\boldsymbol{a},\boldsymbol{p},\boldsymbol{v},\boldsymbol{m}\}$ with channel sort $\boldsymbol{c}$ and four geometric data sorts for scalars, points, vectors and maps.

3.1–1 Definition: Data Values

The set of data values Val is the union of the following five sets.

$x_{\boldsymbol{c}} \in Val_{\boldsymbol{c}} \stackrel{\text{def}}{=} Var_{\boldsymbol{c}}$ are the channels.

$a \in Val_{\boldsymbol{a}} \stackrel{\text{def}}{=} \mathbf{R}$ are the scalars.

$p \in Val_{\boldsymbol{p}} \stackrel{\text{def}}{=} \mathbf{R}^3$ are the points, which we write $\langle x,y,z\rangle$.

$v \in Val_{\boldsymbol{v}}$ are the vectors, a set isomorphic to $Val_{\boldsymbol{p}}$ with a bijection $\uparrow : Val_{\boldsymbol{p}} \to Val_{\boldsymbol{v}}$ with inverse $\downarrow = \uparrow^{-1}$; elements of $Val_{\boldsymbol{v}}$ are written $\uparrow\langle x,y,z\rangle$.

$\mathcal{A} \in Val_{\boldsymbol{m}} \stackrel{\text{def}}{=} \{\langle A,p\rangle \in \mathbf{R}^{3\times 3}\times\mathbf{R}^3 \mid A^{-1} \text{ exists}\}$ are the affine maps.

The basic operators over the data values are given in Definition 3.1–2. Note that there are similar operations on different domains: for example, + between reals, + between vectors, and + between vectors and points. Note also that vector mapping ignores the translation component p (or rather, it cancels when applied to the end points of v); this is the sense in which vectors are 'free': invariant under translation.

3.1–2 Definition: Operations on Points, Vectors, and Maps

$\langle x,y,z\rangle \dot{-} \langle x',y',z'\rangle \stackrel{\text{def}}{=} \uparrow\langle x-x',y-y',z-z'\rangle$	point subtraction
$\uparrow\langle x,y,z\rangle + \langle x',y',z'\rangle \stackrel{\text{def}}{=} \langle x+x',y+y',z+z'\rangle$	point translation
$a\cdot\uparrow\langle x,y,z\rangle \stackrel{\text{def}}{=} \uparrow\langle a\cdot x,a\cdot y,a\cdot z\rangle$	vector scaling
$\uparrow\langle x,y,z\rangle + \uparrow\langle x',y',z'\rangle \stackrel{\text{def}}{=} \uparrow\langle x+x',y+y',z+z'\rangle$	vector addition
$\uparrow\langle x,y,z\rangle \bullet \uparrow\langle x',y',z'\rangle \stackrel{\text{def}}{=} x\cdot x' + y\cdot y' + z\cdot z'$	dot product
$\uparrow\langle x,y,z\rangle \times \uparrow\langle x',y',z'\rangle \stackrel{\text{def}}{=} \uparrow\langle y\cdot z'-z\cdot y',\ z\cdot x'-x\cdot z',\ x\cdot y'-y\cdot x'\rangle$	cross product
$\langle A,p\rangle(q) \stackrel{\text{def}}{=} A\cdot q+p$	point mapping
$\langle A,p\rangle(v) \stackrel{\text{def}}{=} (\uparrow\circ A\circ\downarrow)(v)$	vector mapping
$\langle A,p\rangle\circ\langle A',p'\rangle \stackrel{\text{def}}{=} \langle A\cdot A',\ A\cdot p'+p\rangle$	map composition
$\langle A,p\rangle^{-1} \stackrel{\text{def}}{=} \langle A^{-1},\ -A^{-1}\cdot p\rangle$	map inverse

We now define data terms that we later evaluate to data values in a frame. These include constants for the affine basis: $\dot{-}$ for the origin, and $\uparrow_x,\uparrow_y,\uparrow_z$ for the orthogonal unit vectors. The syntax of data terms includes the data values, indicated by $x_{\boldsymbol{c}} \in Val_{\boldsymbol{c}}$, $a \in Val_{\boldsymbol{a}}$, $p \in Val_{\boldsymbol{p}}$, $v \in Val_{\boldsymbol{v}}$ and $\mathcal{A} \in Val_{\boldsymbol{m}}$, and collectively indicated by $\varepsilon \in Val$. We use five disjoint sets of variables, one for each sort $\sigma \in \Sigma$, indicated by $x_\sigma \in Var_\sigma$.

3.1–3 Definition: Data Terms

$\Delta ::= x_{\boldsymbol{c}} \mid a \mid p \mid v \mid M \mid M[\Delta]$	Data

a ::=	$r \mid f(a_i) \mid v \bullet v' \mid x_{\boldsymbol{a}} \mid a$	$(i \in 1..arity(f))$	Scalars
p ::=	$\dot{-} \mid v+p \mid x_{\boldsymbol{p}} \mid p$		Points
v ::=	$\updownarrow_x \mid \updownarrow_y \mid \updownarrow_z \mid p\text{-}p' \mid a \cdot v \mid v+v' \mid v \times v' \mid x_{\boldsymbol{v}} \mid v$		Vectors
M ::=	$\langle a_{ij},a_k \rangle \mid M \circ M' \mid M^{-1} \mid x_{\boldsymbol{m}} \mid \mathcal{A}$	$(i,j,k \in 1..3)$	Maps
ε ::=	$x_{\boldsymbol{c}} \mid a \mid p \mid v \mid \mathcal{A}$		Values

Data terms consist of *pure terms* in roman style, which form the 'user syntax', and *data values* in italic style, which are inserted during by-value substitutions resulting from process interaction. Note that channels are regarded both as pure terms and values. Each Δ term has the appropriate sort $\sigma \in \Sigma$; the sort of a data frame shift M[Δ] is the sort of Δ. The substitution $\Delta\{x\backslash\varepsilon\}$ simply distributes the substitution on the structure of Δ, until the cases $x\{x\backslash\varepsilon\} = \varepsilon$, $y\{x\backslash\varepsilon\} = y$ for $y \neq x$, or $\varepsilon'\{x\backslash\varepsilon\} = \varepsilon'$.

The scalar terms include real number literals r, dot product v•v', and terms built from some basic functions $f_1(a_1,..,a_{m_1}) \mid ... \mid f_n(a_1,..,a_{m_n})$, abbreviated $f(a_i)$ for $i \in 1..arity(f)$, covering corresponding functions $f_1 \ldots f_n$ on the field of reals plus trigonometry. The point terms include the origin ($\dot{-}$) and addition of a vector to a point. The vector terms include the unit vectors of the standard basis ($\updownarrow_x$, $\updownarrow_y$, $\updownarrow_z$), point subtraction, the vector space operations, and cross product.

The map terms are constructed from base map terms, composition, and inverse. The syntax $\langle a_{11},a_{12},a_{13},a_{21},a_{22},a_{23},a_{31},a_{32},a_{33}, a_1,a_2,a_3 \rangle$, abbreviated $\langle a_{ij},a_k \rangle$ for $i,j,k \in 1..3$, represents a 3x3 square matrix and a translation vector constructed from scalar terms a_{ij},a_k. We require the 3x3 matrix to be invertible, which is verified by a run-time check of the determinant.

The term M[Δ] describes a *data frame shift*, which is used to evaluate Δ in the additional frame defined by M. Note that M[Δ] = Δ is not always true even on scalars; e.g., M[v•v'] is not the same as v•v' when M does not preserve distances. Hence, M[Δ] does not mean apply M to the data value produced by Δ; it means *shift frame* and evaluate the term Δ in the frame M composed with the global frame. The key feature of our semantics is the interplay between frame shifts and the global frame.

In Definition 3.1–4 we define the relation $\Delta \ {}_{\mathcal{A}}\!\!\rightarrowtail \varepsilon$, which describes the *computation* of a closed data term Δ to value ε, relative to global frame $\mathcal{A}$. The relation ${}_{\mathcal{A}}\!\!\rightarrowtail$ is a partial function, described in operational style for ease of induction. The key rule is (Frame Shift): when computation encounters a frame shift M[Δ], the value of M[Δ] in frame $\mathcal{A}$ is uniquely determined as the value of Δ in frame $\mathcal{A} \circ \mathcal{B}$, provided that the value of M in frame $\mathcal{A}$ is $\mathcal{B}$.

3.1–4 Definition: Computation of closed data terms in a frame $\mathcal{A}$

(Scalar Real)	$r \ {}_{\mathcal{A}}\!\!\rightarrowtail b$		if literal r represents $b \in Val_{\boldsymbol{a}}$
(Scalar Arith)	$a_i \ {}_{\mathcal{A}}\!\!\rightarrowtail b_i \Rightarrow f(a_i) \ {}_{\mathcal{A}}\!\!\rightarrowtail f(b_i)$	$i \in 1..arity(f)$	if $b_i \in Val_{\boldsymbol{a}}$, $f(b_i)$ defined
(Scalar Dot)	$v \ {}_{\mathcal{A}}\!\!\rightarrowtail w,\ v' \ {}_{\mathcal{A}}\!\!\rightarrowtail w' \Rightarrow v \bullet v' \ {}_{\mathcal{A}}\!\!\rightarrowtail w \bullet w'$		if $w,w' \in Val_{\boldsymbol{v}}$
(Point Origin)	$\dot{-} \ {}_{\mathcal{A}}\!\!\rightarrowtail \mathcal{A}(\langle 0,0,0 \rangle)$		

(Point Move)	$\mathrm{v}\ {}_{\mathcal{A}}\!\!\rightarrowtail w,\ \mathrm{p}\ {}_{\mathcal{A}}\!\!\rightarrowtail q \Rightarrow \mathrm{v}+\mathrm{p}\ {}_{\mathcal{A}}\!\!\rightarrowtail w \dot{+} q$	if $w \in Val_{\boldsymbol{v}},\ q \in Val_{\boldsymbol{p}}$
(Vect Unit)	$\updownarrow_{\mathrm{x}}\ {}_{\mathcal{A}}\!\!\rightarrowtail \mathcal{A}(\uparrow\langle 1,0,0\rangle),\ \updownarrow_{\mathrm{y}}\ {}_{\mathcal{A}}\!\!\rightarrowtail \mathcal{A}(\uparrow\langle 0,1,0\rangle),\ \updownarrow_{\mathrm{z}}\ {}_{\mathcal{A}}\!\!\rightarrowtail \mathcal{A}(\uparrow\langle 0,0,1\rangle)$	
(Vect Sub)	$\mathrm{p}\ {}_{\mathcal{A}}\!\!\rightarrowtail q,\ \mathrm{p}'\ {}_{\mathcal{A}}\!\!\rightarrowtail q' \Rightarrow \mathrm{p}-\mathrm{p}'\ {}_{\mathcal{A}}\!\!\rightarrowtail q \dot{-} q'$	if $q,q' \in Val_{\boldsymbol{p}}$
(Vect Scale)	$\mathrm{a}\ {}_{\mathcal{A}}\!\!\rightarrowtail b,\ \mathrm{v}\ {}_{\mathcal{A}}\!\!\rightarrowtail w \Rightarrow \mathrm{a}\cdot\mathrm{v}\ {}_{\mathcal{A}}\!\!\rightarrowtail b\cdot w$	if $b \in Val_{\boldsymbol{a}},\ w \in Val_{\boldsymbol{v}}$
(Vect Add)	$\mathrm{v}\ {}_{\mathcal{A}}\!\!\rightarrowtail w,\ \mathrm{v}'\ {}_{\mathcal{A}}\!\!\rightarrowtail w' \Rightarrow \mathrm{v}+\mathrm{v}'\ {}_{\mathcal{A}}\!\!\rightarrowtail w + w'$	if $w,w' \in Val_{\boldsymbol{v}}$
(Vect Cross)	$\mathrm{v}\ {}_{\mathcal{A}}\!\!\rightarrowtail w,\ \mathrm{v}'\ {}_{\mathcal{A}}\!\!\rightarrowtail w' \Rightarrow \mathrm{v}\times\mathrm{v}'\ {}_{\mathcal{A}}\!\!\rightarrowtail w \times w'$	if $w,w' \in Val_{\boldsymbol{v}}$
(Map Given)	$\mathrm{a_{ij}}\ {}_{\mathcal{A}}\!\!\rightarrowtail b_{ij},\ \mathrm{a_k}\ {}_{\mathcal{A}}\!\!\rightarrowtail b_k \Rightarrow \langle \mathrm{a_{ij}},\mathrm{a_k}\rangle\ {}_{\mathcal{A}}\!\!\rightarrowtail \langle b_{ij},b_k\rangle$	if $b_{ij},b_k \in Val_{\boldsymbol{a}},\ det(b_{ij}) \neq 0$
(Map Comp)	$\mathrm{M}\ {}_{\mathcal{A}}\!\!\rightarrowtail \mathcal{B},\ \mathrm{M}'\ {}_{\mathcal{A}}\!\!\rightarrowtail \mathcal{B}' \Rightarrow \mathrm{M}\circ\mathrm{M}'\ {}_{\mathcal{A}}\!\!\rightarrowtail \mathcal{B}\circ\mathcal{B}'$	if $\mathcal{B},\mathcal{B}' \in Val_{\boldsymbol{m}}$
(Map Inv)	$\mathrm{M}\ {}_{\mathcal{A}}\!\!\rightarrowtail \mathcal{B} \Rightarrow \mathrm{M}^{-1}\ {}_{\mathcal{A}}\!\!\rightarrowtail \mathcal{B}^{-1}$	if $\mathcal{B} \in Val_{\boldsymbol{m}}$
(Frame Shift)	$\mathrm{M}\ {}_{\mathcal{A}}\!\!\rightarrowtail \mathcal{B},\ \Delta\ {}_{\mathcal{A}\circ\mathcal{B}}\!\!\rightarrowtail \varepsilon \Rightarrow \mathrm{M}[\Delta]\ {}_{\mathcal{A}}\!\!\rightarrowtail \varepsilon$	if $\mathcal{B} \in Val_{\boldsymbol{m}}$
(Value)	$\varepsilon\ {}_{\mathcal{A}}\!\!\rightarrowtail \varepsilon$	if $\varepsilon \in Val$

Most of these rules express a straightforward correspondence between the syntactic operations on data terms and semantic operations on values. It is easy to check that terms of sort σ compute to elements of Val_{σ}. Note that the rules (Point Origin) and (Vect Unit) make essential use of the current frame. The rules (Scalar Arith) and (Map Given) are partial: they can cause 'divide by zero', 'zero determinant', and other errors. However, (Map Inv) is always defined because if $\mathrm{M}\ {}_{\mathcal{A}}\!\!\rightarrowtail \mathcal{B}$, then $\mathcal{B}$ must be invertible by (Map Given). The (Frame Shift) rule has already been discussed. The (Value) rule normally comes into play after a by-value substitution due to process interaction: a value that was already evaluated in some frame is not further evaluated in the current frame. Moreover, since $Val_{\boldsymbol{c}} = Var_{\boldsymbol{c}}$, the (Value) rule covers also the evaluation of channels to themselves; that is, $\mathrm{x}_{\boldsymbol{c}}\ {}_{\mathcal{A}}\!\!\rightarrowtail \mathrm{x}_{\boldsymbol{c}}$.

4 Frame Shift

In this section we derive our core results, establishing how data computations and process reductions change when we shift fames. The formal statements provide sufficient induction hypotheses, and the proofs are therefore straightforward inductions and are given in Appendix. These results are then used in Section 5 to establish invariance properties under frame shifts. We first give our main result on data computation, which describes what happens to data computation when we shift the global frame. It is enough to prove this result on closed data terms because the frame shifts only occur after the data variables have been substituted for values; see Theorem 4.1–6 for the global frame shifts on processes that contain open data terms.

We define a *compatibility* relation, $\mathcal{A} \vDash \Delta$, between maps $\mathcal{A}$ and closed data terms Δ. A compatibility assumption of the form $\mathcal{A} \vDash \Delta$ (and later $\mathcal{A} \vDash \mathrm{P}$) is used in our theorems to constrain the map $\mathcal{A}$ depending on the vector operators used in Δ; in Section 5 we show that Δ is then observably insensitive to being transformed by $\mathcal{A}$. A closed data term is *affine* if it does not contain v•v' and v×v' subterms, *Euclidean* if it does

not contain v×v' subterms, and *rigid* otherwise.

4.1–1 Definition: Frame and Group Compatibility

For $\mathcal{A}\in GA(3)$ and closed data term Δ, we write $\mathcal{A}\vDash\Delta$ ($\mathcal{A}$ compatible with Δ) iff:
- if Δ contains $\bullet$ then $\mathcal{A}\in E(3)$;
- if Δ contains $\times$ then $\mathcal{A}\in SE(3)$;
- otherwise, no restriction on $\mathcal{A}$.

For group $G\subseteq GA(3)$ and closed data term Δ, we write $G\vDash\Delta$ iff $\forall\,\mathcal{A}\in G.\ \mathcal{A}\vDash\Delta$.

Hence we have also: $GA(3)\vDash\Delta$ implies Δ is affine; $E(3)\vDash\Delta$ implies Δ is Euclidean; $SE(3)\vDash\Delta$ implies Δ is rigid (i.e., $SE(3)\vDash\Delta$ always).

Note in particular that $\mathcal{A}\vDash\varepsilon$ always.

In the formulation of our results we also require the notion of $C(\Delta)$, which is the application of the map C to all the value subterms of Δ:

4.1–2 Definition: Map Application on Data

For $C=\langle A,p\rangle\in Val_{\boldsymbol{m}}$, define

$C(\varepsilon)$	$\stackrel{\text{def}}{=}\ A\cdot\varepsilon+p$	if $\varepsilon\in Val_{\boldsymbol{p}}$	(on points)
$C(\varepsilon)$	$\stackrel{\text{def}}{=}\ (\uparrow\circ A\circ\downarrow)(\varepsilon)$	if $\varepsilon\in Val_{\boldsymbol{v}}$	(on vectors)
$C(\varepsilon)$	$\stackrel{\text{def}}{=}\ \varepsilon$	if $\varepsilon\in Val_{\boldsymbol{a}}\cup Val_{\boldsymbol{m}}\cup Val_{\boldsymbol{c}}$	(on scalars, maps, and channels)

$C(\Delta)$ is the term obtained by replacing all the value subterms ε of Δ with $C(\varepsilon)$.

For example, consider the term $\Delta = \uparrow\langle 1,0,0\rangle + \dot{\cdot}$, containing the value $\uparrow\langle 1,0,0\rangle$, and with reductions (by (Value) and (Point Origin)):

$$\begin{array}{lll} \Delta & = \uparrow\langle 1,0,0\rangle + \dot{\cdot} & \ {}_{\mathcal{A}}\!\!\rightarrowtail \ \uparrow\langle 1,0,0\rangle + \mathcal{A}(\langle 0,0,0\rangle) \\ C(\Delta) & = C(\uparrow\langle 1,0,0\rangle) + \dot{\cdot} & \ {}_{C\circ\mathcal{A}}\!\!\rightarrowtail \ C(\uparrow\langle 1,0,0\rangle) + (C\circ\mathcal{A})(\langle 0,0,0\rangle) \end{array}$$

That is, for $\varepsilon = \uparrow\langle 1,0,0\rangle + \mathcal{A}(\langle 0,0,0\rangle)$, we have:

$$\Delta\ {}_{\mathcal{A}}\!\!\rightarrowtail\varepsilon \quad\text{and}\quad C(\Delta)\ {}_{C\circ\mathcal{A}}\!\!\rightarrowtail C(\varepsilon)$$

Similarly, $\mathcal{B}[\dot{\cdot}]\ {}_{\mathcal{A}}\!\!\rightarrowtail(\mathcal{A}\circ\mathcal{B})(\langle 0,0,0\rangle)$ and $C(\mathcal{B}[\dot{\cdot}]) = \mathcal{B}[\dot{\cdot}]\ {}_{C\circ\mathcal{A}}\!\!\rightarrowtail(C\circ\mathcal{A}\circ\mathcal{B})(\langle 0,0,0\rangle) = C((\mathcal{A}\circ\mathcal{B})(\langle 0,0,0\rangle))$, where $C(\mathcal{B}) = \mathcal{B}$ because maps $\mathcal{B}$ are arrays of reals, and like reals are not affected by mapping. This suggests the general form of our next theorem: $C(\Delta)\ {}_{C\circ\mathcal{A}}\!\!\rightarrowtail$ means applying an extra C separately to the values inside Δ via $C(\Delta)$ (which are then *not* modified by the (Value) rule), and to the other terms inside Δ via ${}_{C\circ\mathcal{A}}\!\!\rightarrowtail$. The proof of Theorem 4.1–3 uses geometric facts that are derived in Appendix 1.

4.1–3 Theorem: Global Frame Shift for Data

$$C\vDash\Delta,\quad \Delta\ {}_{\mathcal{A}}\!\!\rightarrowtail\varepsilon\ \Rightarrow\ C(\Delta)\ {}_{C\circ\mathcal{A}}\!\!\rightarrowtail C(\varepsilon)$$

We now give a local frame shift result on processes that is the exact analog of the (Frame Shift) rule on data given in Definition 3.1–4. This result uses all the new (≡ Map...) rules in the structural congruence relation, except for the (≡ Map Comp) rule. The result depends on data computation only in using the (Frame Shift) and (Map Comp) rules. It would therefore hold for any data sublanguages and data computation

rules which were compatible with these rules. Recall that process reduction, $P \xrightarrow{\mathcal{A}} Q$, was introduced in Definition 2.1–2.

4.1–4 Theorem: Local Frame Shift

$M \xrightarrow{\mathcal{A}} \mathcal{B},\ P \xrightarrow{\mathcal{A}\circ\mathcal{B}} Q \Rightarrow M[P] \xrightarrow{\mathcal{A}} M[Q]$

The main theorem in this section, Theorem 4.1–6 (Global Frame Shift for Processes), is the extension to processes of Theorem 4.1–3. We show that we can shift process reductions to different frames. A shifted process does not reduce to exactly the same process as in the original version, e.g. changing from Q to $C(Q)$, but those differences have *no effect on process traces* (under the usual $\vDash$ assumptions). That is, temporary differences due to value substitutions in different frames can then cancel out in data comparisons.

The $\vDash$ relation extends to processes in the obvious way: $C \vDash P$ holds if and only if $C \vDash \Delta$ holds for all data subterms Δ of P, where $C \vDash \Delta$ is given in Definition 4.1–1. $C(P)$ is the process obtained by replacing all the value subterms ε of P with $C(\varepsilon)$.

4.1–5 Lemma

$C \vDash P,\ P \xrightarrow{\mathcal{A}} Q \Rightarrow C \vDash Q$

4.1–6 Theorem: Global Frame Shift for Processes

$C \vDash P,\ \ P \xrightarrow{\mathcal{A}} Q \ \Rightarrow\ C(P) \xrightarrow{C\circ\mathcal{A}} C(Q)$

5 Observation and Equivalence

In this section we establish the invariance of process congruence under certain transformations of the global frame. We base our results on *barbed congruence*, which is one of the most general notions of process congruence for the π-calculus [4][7][10]**Error! Reference source not found.** and gives rise to a standard definition of algebraic process equation. For 3π, we relativize process equations to affine frames, and investigate how the validity of the equality changes when shifting frames.

Different notions of observation can be characterized by different classes of contexts. We choose to observe processes only via channels, that is, only by interaction and by restricting the interaction channels. Therefore, we do not allow observation contexts that have the flavor of manipulating a whole process, like ?x(y).[] (injecting a process into the observer's code) or M[] (injecting a process into a frame).

5.1–1 Definition (Barbed Congruence)

- ***Observation Context***: An observation context Γ is given by:

  $\Gamma ::= [] \ \vdots\ P|\Gamma \ \vdots\ \Gamma|P \ \vdots\ (\nu x)\Gamma$ where [] only occurs once in Γ.

  The process, $\Gamma[Q]$ is the process obtained by replacing the unique [] in Γ with Q.
- ***Strong Barb on x***: $P\downarrow_x \ \stackrel{\text{def}}{\equiv}\ P \equiv (\nu y_1)..(\nu y_n)\ (!x(\Delta).P' \mid P'')$ with $x \neq y_1..y_n$.
- ***$\mathcal{A}$Barb on x***: $P \xrightarrow{\mathcal{A}}\!\!\downarrow_x \ \stackrel{\text{def}}{\equiv}\ \exists P'.\ P \xrightarrow{\mathcal{A}}{}^{*} P' \wedge P'\downarrow_x$.
- ***$\mathcal{A}$Candidate Relation***: $\mathcal{R}$ is an *$\mathcal{A}$candidate relation* iff for all $P\mathcal{R}Q$:

  (1) if $P\downarrow_x$ then $Q \xrightarrow{\mathcal{A}}\!\!\downarrow_x$; conversely if $Q\downarrow_x$ then $P \xrightarrow{\mathcal{A}}\!\!\downarrow_x$;

(2) if $P \,{}_{\mathcal{A}}\!\!\rightarrow P'$ then there is Q' such that $Q \,{}_{\mathcal{A}}\!\!\rightarrow^* Q'$ and $P'\boldsymbol{\mathcal{R}}Q'$,
if $Q \,{}_{\mathcal{A}}\!\!\rightarrow Q'$ then there is P' such that $P \,{}_{\mathcal{A}}\!\!\rightarrow^* P'$ and $P'\boldsymbol{\mathcal{R}}Q'$;
(3) for all observation contexts Γ, we have $\Gamma[P]\ \boldsymbol{\mathcal{R}}\ \Gamma[Q]$.

- ***$\mathcal{A}$Barbed Congruence***: ${}_{\mathcal{A}}\!\approx$ is the union of all ${}_{\mathcal{A}}$candidate relations, which is itself an ${}_{\mathcal{A}}$candidate relation.

The following theorem, based on Theorem 4.1–6, establishes that barbed congruence is preserved under frame shift. Recall that $C(P)$ denotes the process P with the values shifted by C. We also use $C(\Gamma)$, with $C([]) = []$, so that $C(\Gamma[P]) = C(\Gamma)[C(P)]$.

5.1–2 Theorem: Global Frame Shift for Barbed Congruence

$C \vDash P,Q,\quad P \,{}_{\mathcal{A}}\!\approx Q \;\Rightarrow\; C(P) \,{}_{C\circ\mathcal{A}}\!\approx C(Q)$

Proof

Consider the relation $\boldsymbol{\mathcal{R}} = \{\langle C(P),C(Q)\rangle \mid P \,{}_{\mathcal{A}}\!\approx Q\}$. We show that $\boldsymbol{\mathcal{R}}$ is a ${}_{C\circ\mathcal{A}}$candidate relation. The statement then follows since if $P \,{}_{\mathcal{A}}\!\approx Q$ then $C(P)\boldsymbol{\mathcal{R}}C(Q)$ and $C(P) \,{}_{C\circ\mathcal{A}}\!\approx C(Q)$. Fact: $P{\downarrow_x}$ if and only if $C(P){\downarrow_x}$.

1) Consider any $\langle C(P),C(Q)\rangle$ in $\boldsymbol{\mathcal{R}}$ with $P \,{}_{\mathcal{A}}\!\approx Q$. If $C(P){\downarrow_x}$ then $P{\downarrow_x}$. Since $P \,{}_{\mathcal{A}}\!\approx Q$ and $P{\downarrow_x}$, we have $Q_{\mathcal{A}}{\Downarrow_x}$; that is, $\exists Q'.\ Q \,{}_{\mathcal{A}}\!\!\rightarrow^* Q' \wedge Q'{\downarrow_x}$. By Theorem 4.1–6 and Lemma 4.1–5 we have $C(Q) \,{}_{C\circ\mathcal{A}}\!\!\rightarrow^* C(Q')$. Moreover $Q'{\downarrow_x}$ implies $C(Q'){\downarrow_x}$, and hence $C(Q)_{C\circ\mathcal{A}}{\Downarrow_x}$. The converse is similar.

2) Consider any $\langle C(P),C(Q)\rangle$ in $\boldsymbol{\mathcal{R}}$ with $P \,{}_{\mathcal{A}}\!\approx Q$. If $C(P) \,{}_{C\circ\mathcal{A}}\!\!\rightarrow P''$ then, by Theorem 4.1–6, $C^{-1}(C(P)) \,{}_{C^{-1}\circ C\circ\mathcal{A}}\!\!\rightarrow C^{-1}(P'')$; that is, $P \,{}_{\mathcal{A}}\!\!\rightarrow P' = C^{-1}(P'')$. Since $P \,{}_{\mathcal{A}}\!\approx Q$, there is Q' such that $Q \,{}_{\mathcal{A}}\!\!\rightarrow^* Q'$ and $P' \,{}_{\mathcal{A}}\!\approx Q'$. Hence, by Theorem 4.1–6, there is $Q'' = C(Q')$ such that $C(Q) \,{}_{C\circ\mathcal{A}}\!\!\rightarrow^* Q''$. Rewrite $P' \,{}_{\mathcal{A}}\!\approx Q'$ as $C^{-1}(P'') \,{}_{\mathcal{A}}\!\approx C^{-1}(Q'')$; then, by definition of $\boldsymbol{\mathcal{R}}$, $C(C^{-1}(P''))\ \boldsymbol{\mathcal{R}}\ C(C^{-1}(Q''))$; that is, $P''\boldsymbol{\mathcal{R}}Q''$. We have shown that if $C(P) \,{}_{C\circ\mathcal{A}}\!\!\rightarrow P''$ then there is Q'' such that $C(Q) \,{}_{C\circ\mathcal{A}}\!\!\rightarrow^* Q''$ and $P''\boldsymbol{\mathcal{R}}Q''$. The converse is similar.

3) Consider any $\langle C(P),C(Q)\rangle$ in $\boldsymbol{\mathcal{R}}$ with $P \,{}_{\mathcal{A}}\!\approx Q$. For any observation context Γ, $C^{-1}(\Gamma)$ is an observation context, and hence we have that $C^{-1}(\Gamma)[P] \,{}_{\mathcal{A}}\!\approx C^{-1}(\Gamma)[Q]$. By definition of $\boldsymbol{\mathcal{R}}$, we then have that $C(C^{-1}(\Gamma)[P])\ \boldsymbol{\mathcal{R}}\ C(C^{-1}(\Gamma)[Q])$, that is $\Gamma[C(P)]\ \boldsymbol{\mathcal{R}}\ \Gamma[C(Q)]$.

■

Normally, we are interested in equations between processes without computed values; that is, with the property that $C(P) = P$ which hence simplifies Theorem 5.1–2. We now restrict our attention to such process terms, which we call *pure* terms.

5.1–3 Definition: Pure Terms

We say that a data term Δ and process term P is *pure* if it does not contain a value subterm ε of sort $\sigma \in \{\boldsymbol{a},\boldsymbol{p},\boldsymbol{v},\boldsymbol{m}\}$. We use $\Delta^{¤}$ and $P^{¤}$ to denote such pure terms.

The invariance of equations between pure terms under certain maps is described by a relativity theorem. The key property is that G-equations are G-invariant, meaning that for a group G, the validity or invalidity of equations that are syntactically compat-

ible with G is not changed by G transformations.

5.1–4 Definition: Equations and Laws

An *equation* is a pair of pure process terms $P^{\natural},Q^{\natural}$, written $P^{\natural} = Q^{\natural}$. It is:

- a *G-equation* for $G \subseteq GA(3)$ iff $G \vDash P^{\natural}$ and $G \vDash Q^{\natural}$;
- a *law in* $\mathcal{A}$ for $\mathcal{A} \in GA(3)$ iff $P^{\natural}\ {}_{\mathcal{A}}{\approx}\ Q^{\natural}$;
- a *law in G* for $G \subseteq GA(3)$ iff, $\forall \mathcal{A} \in G$ it is a law in $\mathcal{A}$;
- *$\mathcal{B}$-invariant* for $\mathcal{B} \in GA(3)$ iff $\forall \mathcal{A} \in GA(3)$ it is a law in $\mathcal{A}$ iff it is a law in $\mathcal{B} \circ \mathcal{A}$;
- *G-invariant* for $G \subseteq GA(3)$ iff $\forall \mathcal{B} \in G$ it is *$\mathcal{B}$-invariant*;
- *invariant across G* for $G \subseteq GA(3)$ iff $\forall \mathcal{A},\mathcal{B} \in G$ it is a law in $\mathcal{B}$ if it is a law in $\mathcal{A}$.

5.1–5 Theorem: Relativity

G-equations are *G*-invariant, and hence invariant across *G*.

Proof

Take $\mathcal{A} \in GA(3)$ and $\mathcal{B} \in G \subseteq GA(3)$, and assume that $P^{\natural} = Q^{\natural}$ is a law in $\mathcal{A}$, that is, $P^{\natural}\ {}_{\mathcal{A}}{\approx} Q^{\natural}$. By Theorem 5.1–2, since $\mathcal{B} \vDash P^{\natural},Q^{\natural}$, we have $\mathcal{B}(P^{\natural})\ {}_{\mathcal{B}\circ\mathcal{A}}{\approx}\ \mathcal{B}(Q^{\natural})$. But $P^{\natural},Q^{\natural}$ are pure, so we obtain $P^{\natural}\ {}_{\mathcal{B}\circ\mathcal{A}}{\approx}\ Q^{\natural}$ and hence $P^{\natural} = Q^{\natural}$ is a law in $\mathcal{B} \circ \mathcal{A}$. Conversely, assume $P^{\natural} = Q^{\natural}$ is a law in $\mathcal{B} \circ \mathcal{A}$, that is $P^{\natural}\ {}_{\mathcal{B}\circ\mathcal{A}}{\approx}\ Q^{\natural}$. By Theorem 5.1–2, since $\mathcal{B}^{-1} \vDash P^{\natural},Q^{\natural}$, we have $\mathcal{B}^{-1}(P^{\natural})\ {}_{\mathcal{B}^{-1}\circ\mathcal{B}\circ\mathcal{A}}{\approx}\ \mathcal{B}^{-1}(Q^{\natural})$. Again, $P^{\natural}\ {}_{\mathcal{A}}{\approx}\ Q^{\natural}$, and $P^{\natural} = Q^{\natural}$ is a law in $\mathcal{A}$. We have shown that *G*-equations are *G*-invariant. Assume $P^{\natural} = Q^{\natural}$ is a *G*-equation, and hence *G*-invariant, and take $\mathcal{A},\mathcal{B} \in G$. If $P^{\natural} = Q^{\natural}$ is a law in $\mathcal{A}$ then, since $\mathcal{B} \circ \mathcal{A}^{-1} \in G$, it is also a law in $\mathcal{B} \circ \mathcal{A}^{-1} \circ \mathcal{A}$ by definition of *G*-invariance, and hence it is a law in $\mathcal{B}$. We have shown that *G*-equations are invariant across *G*

■

For the three main transformation groups of interest, Theorem 5.1–5 has the following corollaries: (1) $GA(3)$-equations (those not using $\bullet$ or $\times$) are $GA(3)$-invariant: that is, ***affine equations are invariant under all maps***; (2) $E(3)$-equations (those not using $\times$) are $E(3)$-invariant: that is, ***Euclidean equations are invariant under isometries***; (3) $SE(3)$-equations (*all* equations, since $SE(3)$ imposes no syntactic restrictions) are $SE(3)$-invariant: that is, ***all equations are invariant under rigid-body maps***. Further, *'G-equations are invariant across G'* can be read as *'G laws are the same in all G frames'*; we then obtain that: (1) ***affine laws are the same in all frames***; (2) ***Euclidean laws are same in all Euclidean frames***; (3) ***all laws are the same in all rigid body frames***.

For example, the Euclidean equation $(\updownarrow_x \bullet \updownarrow_x = 1.\ P^{\natural}) = P^{\natural}$ is a law in the $\mathcal{id}$ frame, and hence is a law in all Euclidean frames. Moreover, this equation may be valid or not in some initial frame (possibly a non-Euclidean one like a scaling $S(2\cdot\updownarrow_y)$), but its validity does not change under any further Euclidean transformation. Note also that this equation can be read from left to right as saying that $\updownarrow_x \bullet \updownarrow_x = 1.P^{\natural}$ computes to $P^{\natural}$. Hence equational invariance implies also computational invariance (but this only for computations from pure terms to pure terms, where any value introduced by communication is subsequently eliminated by data comparison).

As a second example, for any three points $p^{\natural},q^{\natural},r^{\natural}$, the affine equation $((q^{\natural} \dot{-} p^{\natural}) + (r^{\natural} \dot{-} q^{\natural}) = (r^{\natural} \dot{-} p^{\natural}).\ P^{\natural}) = P^{\natural}$ is a law in the $\mathcal{id}$ frame, and so is a law in all frames; in

fact it is the head-to-tail axiom of affine space. As a third example, for any point $p^{¤}$, the equation ($p^{¤}$=⊹. $P^{¤}$) = $P^{¤}$ is invariant under all translations (because all equations are invariant under rigid-body maps); hence, the comparison $p^{¤}$=⊹ gives the same result under all translations, and cannot be used to test the true value of the origin no matter how $p^{¤}$ is expressed, as long as it is a pure term.

In conclusion, we have shown that all process equations are invariant under rigid body transformations (rotations and translations, not reflections), implying that no pure process can observe the location of the origin, nor the orientation of the basis vectors in the global frame. Moreover, processes that do not perform absolute measurements (via • and ×) are invariant under all affine transformations, meaning that they are also unable to observe the size of the basis vectors and the angles between them. Finally, processes that use • but not × are invariant under all the isometries, meaning that they cannot observe whether they have been reflected.

Conclusions

We have introduced 3π, an extension of the π-calculus based on affine geometry, with the primary aim of describing the concurrent evolution of geometric structures in 3D space. We have proven a relativity theorem stating that all algebraic equations (under a version of barbed congruence) are invariant under all rigid body transformations. If a process is unable to observe distances, angles or orientations, then similar results also apply to larger classes of transformations. These results have implications for the extent to which a process can observe its geometric frame and for the behavior of a process in different geometric frames.

References

[1] G. Berry, G. Boudol. The Chemical Abstract Machine. Proc. POPL'89, 81-94..
[2] L. Cardelli, A.D. Gordon. Mobile Ambients. Theoretical Computer Science, Special Issue on Coordination, D. Le Métayer Editor. Vol 240/1, June 2000. pp 177-213.
[3] H.S.M. Coxeter, Introduction to geometry, Wiley, 1961.
[4] C Fournet, G Gonthier. A Hierarchy of Equivalences for Asynchronous Calculi. Proc. 25th ICALP. LNCS 1443, 844-855. Springer 1998.
[5] J. Gallier. Geometric Methods and Applications for Computer Science and Engineering. Springer, 2001.
[6] M. Hennessy. A Distributed Pi-Calculus. Cambridge University Press, 2007.
[7] K. Honda, N. Yoshida. On Reduction-Based Process Semantics. Theoretical Computer Science, 152(2), pp. 437-486, 1995.
[8] M. John, R. Ewald, A.M. Uhrmacher. A Spatial Extension to the π Calculus. Electronic Notes in Theoretical Computer Science, 194(3) 133-148, Elsevier, 2008.
[9] R.J. Metzger, O.D. Klein, G.R. Martin, M.A. Krasnow. The branching programme of mouse lung development. Nature 453(5), June 2008.
[10] R. Milner. Communicating and Mobile Systems: The π-Calculus. CUP, 1999. R. Milner, D. Sangiorgi. Barbed Bisimulation. In Proc. 19-the International Colloquium on Automata, Languages and Programming (ICALP '92), LNCS 623, Springer, 1992.
[11] P. Prusinkiewicz, A. Lindenmayer. The Algorithmic Beauty of Plants. Springer, 1991.

6 Appendix 1 of 2: Geometry (optional reading)

6.1 Vector Spaces and Automorphism Groups

A *vector space* over a field F is a set V with operations $+ \in V\times V\to V$ (vector addition) and $\cdot \in F\times V\to V$ (scalar multiplication), such that $(V,+)$ is an abelian group, with identity the zero vector $\emptyset$ and inverse $-v$, and moreover: $a\cdot(v+w) = a\cdot v+a\cdot w$, $(a+b)\cdot v = a\cdot v+b\cdot v$, $(a\cdot b)\cdot v = a\cdot(b\cdot v)$, and $1\cdot v = v$. Three-dimensional space, $\mathbf{R}^3$, is our basic vector space over the field of reals: the vectors are the points of $\mathbf{R}^3$, $+$ is coordinatewise addition, and $\cdot$ is coordinatewise multiplication. A *linear map* over a vector space V is an $f \in V\to V$ such that $f(v+w) = f(v)+f(w)$ and $f(a\cdot v) = a\cdot f(v)$; group axioms then ensure that it preserves also unit and inverse. $Lin(V)$ is the set of such linear maps. In *Euclidean spaces*, e.g. $\mathbf{R}^3$, one considers the ability to *measure*. This is achieved by extending the underlying vector space with the *dot product* of vectors, giving the ability to measure distances and angles, and with the *cross product* of vectors, giving the ability to generate out-of-plane vectors, to measure areas and volumes, and to detect handedness. Both dot and cross product are linear maps in each argument.

The *General Linear Group* $GL(V) \subseteq Lin(V)$ of a vector space V is the group of all the automorphisms (bijective linear maps) over V, i.e., invertible elements of $Lin(V)$. When studying subgroups of $GL(V)$, it is convenient to use linear algebra to represent the group elements. In particular, $GL(\mathbf{R}^3)$, the group of automorphisms of the $\mathbf{R}^3$ vector space, can be given as the group of invertible 3×3 matrices A in linear algebra, where matrix multiplication $(A\cdot B)$ is an operation over sizes $(n\times m) \times (m\times n) \to (m\times n)$. On matrices we use also A^{T} for transposition, $A+B$ for addition, $a\cdot A$ for scalar multiplication, and A^{-1} for inverse. With the elements $v\in\mathbf{R}^3$ interpreted as 1×3 (column) matrices, we obtain the required linearity properties from linear algebra: $A\cdot(v+v') = A\cdot v + A\cdot v'$ and $A\cdot(a\cdot v) = a\cdot(A\cdot v)$ for any scalar a. Note again that only the invertible, i.e. bijective, matrices are members of $GL(\mathbf{R}^3)$. The *Special Linear Group* $SL(\mathbf{R}^3)$ is the subgroup of matrices with determinant 1: as transformations these preserve volume and handedness.

The *General Affine Group* $GA(V)$ is the group of *affine vector maps* over V; these maps are presented as pairs $\langle A,u\rangle$ where $A\in GL(V)$, and where $u\in V$ is a translation vector. In particular, $GA(\mathbf{R}^3)$ is the affine group over the $\mathbf{R}^3$ vector space. We use 3×3 invertible matrices for A, with $\langle A,u\rangle(v) \stackrel{\text{def}}{=} A\cdot v+u$ for any $v\in\mathbf{R}^3$. Geometrically, affine vector maps transform straight lines into straight lines, and preserve ratios such as midpoints. The *Special Affine Group* $SA(\mathbf{R}^3)$ is the subgroup with matrices with determinant 1.

Concretely, we work always over the field $\mathbf{R}$ and the vector space $\mathbf{R}^3$, hence we abbreviate these groups as $GA(3)$, $SA(3)$, $GL(3)$, $SL(3)$.

For the next automorphisms groups we need to investigate some special matrices. An *orthogonal matrix* is a square matrix A such that $A^{\mathrm{T}} = A^{-1}$ (and hence $A\cdot A^{\mathrm{T}} = A^{\mathrm{T}}\cdot A = id$, and also $det(A) = \pm 1$). All orthogonal matrices are *isometries*, i.e., preserve distances, which can be seen as follows. The vector dot product (of column matrices) is defined as $v\bullet w \stackrel{\text{def}}{=} v^{\mathrm{T}}\cdot w$, and $v^2 \stackrel{\text{def}}{=} v\bullet v$. If $A^{\mathrm{T}} = A^{-1}$ we then have that $A\cdot v\bullet A\cdot w = (A\cdot v)^{\mathrm{T}}\cdot(A\cdot w) = v^{\mathrm{T}}\cdot A^{\mathrm{T}}\cdot A\cdot w = v^{\mathrm{T}}\cdot id\cdot w = v^{\mathrm{T}}\cdot w = v\bullet w$. And also $(A\cdot v)^2 = v^2$. Distance in a vector space equipped with dot product is defined as $d(v,w) \stackrel{\text{def}}{=} \sqrt{(v-w)^2}$. For A orthogonal, we then have $d(A\cdot v,A\cdot w) = \sqrt{(A\cdot v-A\cdot w)^2} = \sqrt{(A\cdot(v-w))^2} = \sqrt{(v-w)^2} = d(v,w)$, that is, A preserves distances.

The *Orthogonal Group* $O(3)$, subgroup of $GL(3)$, is the group *linear isometries* of $\mathbf{R}^3$, that is, the group of orthogonal matrices, which correspond to rotations and reflections around the origin. As we have just shown, members of $O(3)$ preserve dot product: $A\cdot v \bullet A\cdot w = v\bullet w$. The *special orthogonal group* $SO(3)$ contains only the direct linear isometries, that is, just the rotations. Members of $SO(3)$ distribute over cross product: $A\cdot v \times A\cdot w = A\cdot(v\times w)$ [12]. Intuitively that is because cross product can measure areas and handedness, but is insensitive to isometries that do not change handedness.

The *Euclidean Group* $E(3)$, subgroup of $GA(3)$, is the group of isometries of $\mathbf{R}^3$; its elements can be given as affine vector maps $\langle A,u\rangle$ where A is an orthogonal matrix (a rotation or reflection) and u is a translation vector. We have seen that members of $O(3)$ are isometries, but such $\langle A,u\rangle$ are too: for $A\in O(3)$ we have that $d(\langle A,u\rangle(v),\langle A,u\rangle(w)) = d(A\cdot v+u,A\cdot w+u) = \sqrt{(A\cdot v+u-(A\cdot w+u))^2} = \sqrt{(A\cdot v-A\cdot w)^2} = d(v,w)$. That is, all affine vector maps $\langle A,u\rangle$ where A is an orthogonal matrix are also isometries.

The subgroup $SE(3)$ of $E(3)$ of *direct isometries* excludes reflections; that is, the determinant of A must be 1. Elements of $SE(3)$ are then the *rigid body motions*, preserving handedness and distances.

The subgroup relation on the automorphism groups discussed so far forms a cube standing on the $SO(3)$ vertex, with $GA(3)$ at the top. Maps contained in the bottom faces of the cube have the following interpretation: the face below $E(3)$ preserves distances and angles; the face below $SA(3)$ preserves volumes and orientation; the face below $GL(3)$ preserves the origin. Various vertices of the cube hold the basic geometric transformations: rotation, translation, reflection, shearing, isotropic scaling, and volume-preserving squishing (non-orthogonal matrices with det=1). There are many more automorphism groups; e.g., the group of pure translations, below $SE(3)$, the group of pure reflections, below $O(3)$, and the group of identities below all of them. However, the cube depicts the most studied automorphism groups, and a finer structure is not necessary for the study of geometric invariance properties, at least not in this paper.

We work in $GA(3)$ and its subgroups. For example, we regard an affine vector map $\langle A,u\rangle\in GA(3)$ as a member of $GL(3)$ when $u=0$, and as a member of $E(3)$ when A is orthogonal, and further as a member of $O(3)$ when $u=0$. We fix a representation of affine vector maps based on linear algebra.

6.2 Affine Spaces and Affine Maps

Affine geometry is intuitively the geometry of properties invariant under translation, rotation, reflection and stretching. It can be properly formulated by the notions of affine spaces and affine maps [3][5].

6.2–1 Definition: Affine spaces

An *affine space* is a triple (P,V,θ) where P is a set (of points), V is a vector space, and $\theta \in P\times P\to V$ is a function which characterizes 'the unique vector $\theta(p,q)$ from p to q'. The map θ must satisfy:

1) for each $p \in P$, $\theta_p \in P\to V = \lambda q.\theta(p,q)$ is a bijection;
2) the *head-to-tail equation* holds: $\theta(p,q) + \theta(q,r) = \theta(p,r)$.

Because of (1), P and V are isomorphic, but there is no canonical isomorphism. The vector $\theta(p,q)$ is sometimes called the *point difference*, written $q\dot{-}p$. We also define *vector-point addition* as $+ \in V\times P\to P = \lambda v,p.\ \theta_p^{-1}(v)$ (which is a *group action* of $(V,+)$ on P).

The *affine space of free vectors* over P is a canonical affine space constructed over a set of points P that is also a vector space. It is common to take $V=P$ in such a construction. In our operational semantics, however, we need to distinguish between points and vectors; hence we take for V a set isomorphic but distinguishable from P. We focus on the space of free vectors over the points of $\mathbf{R}^3$. Note that $\mathbf{R}^3$ is also a vector space, with the null vector indicated by ø.

6.2–2 Definition: The affine space of free vectors over $\mathbf{R}^3$

The affine space of free vectors over $\mathbf{R}^3$ is $(\mathbf{R}^3, FV(\mathbf{R}^3), \Uparrow)$, where:
- The set of points of the affine space is $\mathbf{R}^3$.
- $FV(\mathbf{R}^3) \stackrel{\text{def}}{=} \{ø\}\times\mathbf{R}^3$ is a vector space equipped with $\bullet$ and $\times$, given by the product structure.
- $\Uparrow \in \mathbf{R}^3\times\mathbf{R}^3\to FV(\mathbf{R}^3) \stackrel{\text{def}}{=} \lambda(p,q).\langle ø,q-p\rangle$.

Auxiliary definitions and properties:

- $\Uparrow_p \stackrel{\text{def}}{\equiv} \lambda(q).\Uparrow(p,q)$ is a bijection for each p.
- $q \dot{-} p \stackrel{\text{def}}{\equiv} \Uparrow(p,q) = \langle \emptyset, q-p \rangle$
- $v \dot{+} p \stackrel{\text{def}}{\equiv} \Uparrow_p^{-1}(v)$ with $\langle \emptyset, q \rangle \dot{+} p = q+p$.
- $\uparrow \stackrel{\text{def}}{\equiv} \Uparrow_\emptyset$ and $\downarrow \stackrel{\text{def}}{\equiv} \uparrow^{-1}$ are linear maps, with $\uparrow p = \langle \emptyset, p \rangle$, and $\downarrow\langle \emptyset, p \rangle = p$.

The set $\{\emptyset\}\times\mathbf{R}^3$ can be seen also as the set of canonical representatives of *free vectors* (equivalence classes of vectors with the same size and orientation), and can be explained as the vectors rooted at the origin.

Affine vector maps of the form $\lambda v.\ f(v)+u \in V \to V$ with $f \in Lin(V)$ are common in the literature of automorphism groups, as presented in Section 6.1. *Affine point maps* of the form $\lambda q.\ f(q \dot{-} o) \dot{+} p \in P \to P$ instead are common in the literature of affine spaces [5]. Confusingly, they are both called just 'affine maps'. Bijective point and vector maps form groups under function composition, identity, and inverse, and these groups are related by a group isomorphism: for each choice of origin o, just like there is an isomorphism θ_o between points P and vectors V, there is also a group isomorphism $\psi_o = \lambda h.\ \theta_o \circ h \circ \theta_o^{-1}$ between the group of bijective affine point maps with origin o, and the group of bijective affine vector maps $GA(V)$. The isomorphism transforms a point map that maps a point p seen as a vector $p \dot{-} o$ to the point $f(p \dot{-} o) \dot{+} q$, into a vector map that maps the vector $p \dot{-} o$ to the vector $f(p \dot{-} o)+(q \dot{-} o)$, which when rooted at the origin leads to the point $(f(p \dot{-} o)+(q \dot{-} o)) \dot{+} o = f(p \dot{-} o) \dot{+} q$. Up to this group isomorphism, we consider affine point maps (then called just *affine maps* in the body of this paper) as members of $GA(V)$.

Affine point maps over the affine space of free vectors over $\mathbf{R}^3$ are denoted by script letters $\mathcal{A},\mathcal{B},\mathcal{C},...$ and are represented as pairs $\mathcal{A} = \langle A,q \rangle$. They are applied to points p to obtain transformed points $A \cdot p+q$, and are extended to vectors $v = \Uparrow(p,q)$ by taking $\mathcal{A}(\Uparrow(p,q)) = \Uparrow(\mathcal{A}(p),\mathcal{A}(q))$, which means that $\mathcal{A}(v) = (\uparrow \circ A \circ \downarrow)(v)$, where the translation components cancel: this reflects the fact that v are 'free' vectors, invariant under translation. These rules for applying maps, and the rules for composing and inverting maps, are given in Definition 6.2–3.

6.2–3 Definition: Affine point maps

$\mathcal{A} \in GA(3)$ means $\mathcal{A} = \langle A,p \rangle$ where $\uparrow \circ A \circ \downarrow \in GL(3)$, A is an invertible 3×3 matrix, and $p \in \mathbf{R}^3$.
$\mathcal{A} \in E(3)$ means $\mathcal{A} = \langle A,p \rangle$ where $\uparrow \circ A \circ \downarrow \in O(3)$; that is, $A^T = A^{-1}$.
$\mathcal{A} \in SE(3)$ means $\mathcal{A} = \langle A,p \rangle$ where $\uparrow \circ A \circ \downarrow \in SO(3)$; that is, $det(A) = 1$.

$\forall q \in \mathbf{R}^3,\ \forall \mathcal{A} \in GA(3).$	$\mathcal{A}(q) = \langle A,p \rangle(q) \stackrel{\text{def}}{\equiv} A \cdot q+p$	$\in \mathbf{R}^3$
$\forall v \in FV(\mathbf{R}^3),\ \forall \mathcal{A} \in GA(3).$	$\mathcal{A}(v) = \langle A,p \rangle(v) \stackrel{\text{def}}{\equiv} (\uparrow \circ A \circ \downarrow)(v)$	$\in FV(\mathbf{R}^3)$
$\forall \mathcal{A},\mathcal{B} \in G$ subgroup of $GA(3).$	$\mathcal{A} \circ \mathcal{B} = \langle A,p \rangle \circ \langle B,q \rangle \stackrel{\text{def}}{\equiv} \langle A \cdot B, A \cdot q+p \rangle$	$\in G$
$\forall \mathcal{A} \in G$ subgroup of $GA(3).$	$\mathcal{A}^{-1} = \langle A,p \rangle^{-1} \stackrel{\text{def}}{\equiv} \langle A^{-1}, -A^{-1} \cdot p \rangle$	$\in G$

It should be noted that this definition can be formulated as a theorem in a general treatment of the groups of affine vector maps and affine point maps, and their representation in terms of linear algebra. For conciseness, we take it here as a given.

The following proposition collects all the geometric facts needed in Theorem 4.1–3.

6.2–4 Proposition: Distribution laws of affine point maps

1) $\forall p,q \in \mathbf{R}^3,\ \mathcal{A} \in GA(3).$	$\mathcal{A}(q) \dot{-} \mathcal{A}(p) = \mathcal{A}(q \dot{-} p)$	$\in FV(\mathbf{R}^3)$
2) $\forall v \in FV(\mathbf{R}^3),\ p \in \mathbf{R}^3,\ \mathcal{A} \in GA(3).$	$\mathcal{A}(v) \dot{+} \mathcal{A}(p) = \mathcal{A}(v \dot{+} p)$	$\in \mathbf{R}^3$
3) $\forall v,w \in FV(\mathbf{R}^3),\ \mathcal{A} \in GA(3).$	$\mathcal{A}(v)+\mathcal{A}(w) = \mathcal{A}(v+w)$	$\in FV(\mathbf{R}^3)$
4) $\forall a \in \mathbf{R},\ v \in FV(\mathbf{R}^3),\ \mathcal{A} \in GA(3).$	$a \cdot \mathcal{A}(v) = \mathcal{A}(a \cdot v)$	$\in FV(\mathbf{R}^3)$
5) $\forall v,w \in FV(\mathbf{R}^3),\ \mathcal{A} \in E(3).$	$\mathcal{A}(v) \bullet \mathcal{A}(w) = v \bullet w$	$\in \mathbf{R}$
6) $\forall v,w \in FV(\mathbf{R}^3),\ \mathcal{A} \in SE(3).$	$\mathcal{A}(v) \times \mathcal{A}(w) = \mathcal{A}(v \times w)$	$\in FV(\mathbf{R}^3)$

Proof
Let $\mathcal{A} = \langle A,r\rangle$. By Definition 6.2–3: if $\mathcal{A}\in GA(3)$, then the vector map $\uparrow\circ A\circ\downarrow\in GL(3)$ is a linear map; if $\mathcal{A}\in E(3)$, then $\uparrow\circ A\circ\downarrow\in O(3)$; and if $\mathcal{A}\in SE(3)$ then $\uparrow\circ A\circ\downarrow\in SO(3)$. Recall that if $f \in O(3)$ then $f(v) \bullet f(w) = v\bullet w$, and if $f \in SO(3)$ then $f(v) \times f(w) = f(v\times w)$ [12].
1) $\langle A,r\rangle(q) \div \langle A,r\rangle(p) = A\cdot q+r \div A\cdot p+r = \langle \emptyset,(A\cdot q+r)-(A\cdot p+r)\rangle = \langle \emptyset,A\cdot(q-p)\rangle = (\uparrow\circ A\circ\downarrow)\cdot\langle \emptyset,q-p\rangle = \langle A,r\rangle(q\div p)$.
2) $\langle A,r\rangle(\langle \emptyset,q\rangle) + \langle A,r\rangle(p) = (\uparrow\circ A\circ\downarrow)(\langle \emptyset,q\rangle) + A\cdot p+r = \langle \emptyset,A\cdot q\rangle + A\cdot p+r = A\cdot(q+p)+r = \langle A,r\rangle(q+p) = \langle A,r\rangle(\langle \emptyset,q\rangle + p)$.
3) $\langle A,r\rangle(\langle \emptyset,p\rangle) + \langle A,r\rangle(\langle \emptyset,q\rangle) = (\uparrow\circ A\circ\downarrow)(\langle \emptyset,p\rangle) + (\uparrow\circ A\circ\downarrow)(\langle \emptyset,q\rangle) = \uparrow(A\cdot p) + \uparrow(A\cdot q) = \uparrow(A\cdot p + A\cdot q) = \uparrow(A\cdot(p+q)) = (\uparrow\circ A\circ\downarrow)(\langle \emptyset,p+q\rangle) = (\uparrow\circ A\circ\downarrow)(\langle \emptyset,p\rangle+\langle \emptyset,q\rangle) = \langle A,r\rangle(\langle \emptyset,p\rangle+\langle \emptyset,q\rangle)$.
4) $a\cdot\langle A,r\rangle(\langle \emptyset,p\rangle) = a\cdot(\uparrow\circ A\circ\downarrow)(\langle \emptyset,p\rangle) = a\cdot\uparrow(A\cdot p) = \uparrow(a\cdot A\cdot p) = \uparrow(A\cdot(a\cdot p)) = (\uparrow\circ A\circ\downarrow)(\langle \emptyset,a\cdot p\rangle) = \langle A,r\rangle(a\cdot\langle \emptyset,p\rangle)$.
5) $\langle A,r\rangle(v) \bullet \langle A,r\rangle(w) = (\uparrow\circ A\circ\downarrow)(v) \bullet (\uparrow\circ A\circ\downarrow)(w) = v\bullet w$ since $\uparrow\circ A\circ\downarrow\in O(3)$.
6) $\langle A,r\rangle(v) \times \langle A,r\rangle(w) = (\uparrow\circ A\circ\downarrow)(v) \times (\uparrow\circ A\circ\downarrow)(w) = (\uparrow\circ A\circ\downarrow)(v\times w)$ since $\uparrow\circ A\circ\downarrow\in SO(3) = \langle A,r\rangle(v\times w)$.
■

References

[12] F. Jones. Vector Calculus. Chapter 7: Cross Product. (Unpublished book; available at http://www.owlnet.rice.edu/~fjones/chap7.pdf.)

7 Appendix 2 of 2: Proofs (optional reading)

7.1–1 Theorem 4.1–3: Global Frame Shift for Data

$C\vDash\Delta,\ \Delta\ {}_{\mathcal{A}}\!\!\rightarrowtail \varepsilon\ \Rightarrow\ C(\Delta)\ {}_{C\circ\mathcal{A}}\!\!\rightarrowtail C(\varepsilon)$

Proof
The proof is by mutual induction on the derivation of $\Delta\ {}_{\mathcal{A}}\!\!\rightarrowtail \varepsilon$; that is, by induction on the conjunction of 5 statements for the 5 sorts σ of Δ, as given in the 5 cases below. When $\Delta = \varepsilon$, the ε of the various sorts fall into the respective subcases. Since all these subcases are handled equally, we show the (Value) case first:

Rule (Value): Show that $C\vDash\varepsilon,\ \varepsilon\ {}_{\mathcal{A}}\!\!\rightarrowtail \varepsilon\ \Rightarrow\ C(\varepsilon)\ {}_{C\circ\mathcal{A}}\!\!\rightarrowtail C(\varepsilon)$, for ε of any sort.
Then, by (Value) $C(\Delta) = C(\varepsilon)\ {}_{C\circ\mathcal{A}}\!\!\rightarrowtail C(\varepsilon)$.

Case (σ=***c***): Show that $C\vDash\Delta,\ \Delta\ {}_{\mathcal{A}}\!\!\rightarrowtail \mathrm{x}_{\boldsymbol{c}}\ \Rightarrow\ C(\Delta)\ {}_{C\circ\mathcal{A}}\!\!\rightarrowtail \mathrm{x}_{\boldsymbol{c}}$.
Then, $\Delta\ {}_{\mathcal{A}}\!\!\rightarrowtail \mathrm{x}_{\boldsymbol{c}}$ is the consequent of Rule (Value) or:

Rule (Frame Shift): $\mathrm{M}\ {}_{\mathcal{A}}\!\!\rightarrowtail \mathcal{B},\ \Delta'\ {}_{\mathcal{A}\circ\mathcal{B}}\!\!\rightarrowtail \mathrm{x}_{\boldsymbol{c}}\ \Rightarrow\ \mathrm{M}[\Delta']\ {}_{\mathcal{A}}\!\!\rightarrowtail\ \mathrm{x}_{\boldsymbol{c}}$.
Since $C\vDash\mathrm{M}[\Delta']$, we have $C\vDash\mathrm{M}$ and $C\vDash\Delta'$.
It follows that $C(\mathrm{M})\ {}_{C\circ\mathcal{A}}\!\!\rightarrowtail \mathcal{B}$ and $C(\Delta')\ {}_{C\circ\mathcal{A}\circ\mathcal{B}}\!\!\rightarrowtail \mathrm{x}_{\boldsymbol{c}}$ (by induction).
Hence $C(\mathrm{M}[\Delta']) = C(\mathrm{M})[C(\Delta')]\ {}_{C\circ\mathcal{A}}\!\!\rightarrowtail \mathrm{x}_{\boldsymbol{c}}$ by (Frame Shift).

Case (σ=***a***): Show that $C\vDash\mathrm{a},\ \mathrm{a}\ {}_{\mathcal{A}}\!\!\rightarrowtail b\ \Rightarrow\ C(\mathrm{a})\ {}_{C\circ\mathcal{A}}\!\!\rightarrowtail b$.
Then $\mathrm{a}\ {}_{\mathcal{A}}\!\!\rightarrowtail b$ is the consequent of Rule (Value) or one of the rules:

Rule (Scalar Real): $r \ {}_{\mathcal{A}}\!\!\rightarrowtail b$. Then $C(r) = r \ {}_{C\circ\mathcal{A}}\!\!\rightarrowtail b$ (by (Scalar Real)).

Rule (Scalar Arith): $a_i \ {}_{\mathcal{A}}\!\!\rightarrowtail b_i \Rightarrow f(a_i) \ {}_{\mathcal{A}}\!\!\rightarrowtail f(b_i)$ with $i \in 1..arity(f)$
Since $f(a_i) \ {}_{\mathcal{A}}\!\!\rightarrowtail f(b_i)$ we know that $f(b_i)$ is defined.
Then $C(f(a_i)) = f(C(a_i)) \ {}_{C\circ\mathcal{A}}\!\!\rightarrowtail f(b_i)$ (by induction and (Scalar Arith)).

Rule (Scalar Dot): $v \ {}_{\mathcal{A}}\!\!\rightarrowtail w$, $v' \ {}_{\mathcal{A}}\!\!\rightarrowtail w' \Rightarrow v \bullet v' \ {}_{\mathcal{A}}\!\!\rightarrowtail w \bullet w'$, and $C \in E(3)$.
$C(v \bullet v') = C(v) \bullet C(v') \ {}_{C\circ\mathcal{A}}\!\!\rightarrowtail C(w) \bullet C(w')$ (by induction and (Scalar Dot))
$= w \bullet w'$ (by Prop. 6.2–4).

Rule (Frame Shift)($\sigma = \boldsymbol{a}$): $M \ {}_{\mathcal{A}}\!\!\rightarrowtail \mathcal{B}$, $a' \ {}_{\mathcal{A}\circ\mathcal{B}}\!\!\rightarrowtail b \Rightarrow M[a'] \ {}_{\mathcal{A}}\!\!\rightarrowtail b$.
Since $C \vDash M[a']$, we have $C \vDash M$ and $C \vDash a'$.
It follows that $C(M) \ {}_{C\circ\mathcal{A}}\!\!\rightarrowtail \mathcal{B}$ and $C(a') \ {}_{C\circ\mathcal{A}\circ\mathcal{B}}\!\!\rightarrowtail b$ (by induction).
Hence $C(M[a']) = C(M)[C(a')] \ {}_{C\circ\mathcal{A}}\!\!\rightarrowtail b$ by (Frame Shift).

Case ($\sigma = \boldsymbol{p}$): Show that $C \vDash p$, $p \ {}_{\mathcal{A}}\!\!\rightarrowtail q \Rightarrow C(p) \ {}_{C\circ\mathcal{A}}\!\!\rightarrowtail C(q)$.
Then $p \ {}_{\mathcal{A}}\!\!\rightarrowtail q$ is the consequent of Rule (Value) or one of the rules:

Rule (Point Origin): $\dot{\cdot} \ {}_{\mathcal{A}}\!\!\rightarrowtail \mathcal{A}(\langle 0,0,0\rangle)$.
$C(\dot{\cdot}) = \dot{\cdot} \ {}_{C\circ\mathcal{A}}\!\!\rightarrowtail (C\circ\mathcal{A})(\langle 0,0,0\rangle)$ (by (Point Origin)) $= C(\mathcal{A}(\langle 0,0,0\rangle))$

Rule (Point Move): $v \ {}_{\mathcal{A}}\!\!\rightarrowtail w$, $p' \ {}_{\mathcal{A}}\!\!\rightarrowtail q' \Rightarrow v + p' \ {}_{\mathcal{A}}\!\!\rightarrowtail w + q'$.
$C(v + p') = C(v) + C(p') \ {}_{C\circ\mathcal{A}}\!\!\rightarrowtail C(w) + C(q')$ (by induction and (Point Move))
$= C(w + q')$ (by Prop. 6.2–4).

Rule (Frame Shift)($\sigma = \boldsymbol{p}$): $M \ {}_{\mathcal{A}}\!\!\rightarrowtail \mathcal{B}$, $p' \ {}_{\mathcal{A}\circ\mathcal{B}}\!\!\rightarrowtail q \Rightarrow M[p'] \ {}_{\mathcal{A}}\!\!\rightarrowtail q$.
Since $C \vDash M[p']$, we have $C \vDash M$ and $C \vDash p'$.
It follows that $C(M) \ {}_{C\circ\mathcal{A}}\!\!\rightarrowtail \mathcal{B}$ and $C(p') \ {}_{C\circ\mathcal{A}\circ\mathcal{B}}\!\!\rightarrowtail C(q)$ (by induction).
Hence $C(M[p']) = C(M)[C(p')] \ {}_{C\circ\mathcal{A}}\!\!\rightarrowtail C(q)$ by (Frame Shift).

Case ($\sigma = \boldsymbol{v}$): Show that $C \vDash v$, $v \ {}_{\mathcal{A}}\!\!\rightarrowtail w \Rightarrow C(v) \ {}_{C\circ\mathcal{A}}\!\!\rightarrowtail C(w)$.
Then $v \ {}_{\mathcal{A}}\!\!\rightarrowtail w$ is the consequent of Rule (Value) or one of the rules:

Rule (Vect Unit): $\updownarrow_x \ {}_{\mathcal{A}}\!\!\rightarrowtail \mathcal{A}(\langle 1,0,0\rangle)$.
$C(\updownarrow_x) = \updownarrow_x \ {}_{C\circ\mathcal{A}}\!\!\rightarrowtail (C\circ\mathcal{A})(\langle 1,0,0\rangle)$ (by (Vect Unit)) $= C(\mathcal{A}(\langle 1,0,0\rangle))$. Similarly for $\updownarrow_y$ and $\updownarrow_z$.

Rule (Vect Sub): $p \ {}_{\mathcal{A}}\!\!\rightarrowtail q$, $p' \ {}_{\mathcal{A}}\!\!\rightarrowtail q' \Rightarrow p - p' \rightarrowtail q \dot{-} q'$.
$C(p - p') = C(p) - C(p') \ {}_{C\circ\mathcal{A}}\!\!\rightarrowtail C(q) \dot{-} C(q')$ (by induction and (Vect Sub))
$= C(q \dot{-} q')$ (by Prop. 6.2–4).

Rule (Vect Scale): $a \ {}_{\mathcal{A}}\!\!\rightarrowtail b$, $v' \ {}_{\mathcal{A}}\!\!\rightarrowtail w' \Rightarrow a \cdot v' \ {}_{\mathcal{A}}\!\!\rightarrowtail b \cdot w'$.
$C(a \cdot v') = C(a) \cdot C(v') \ {}_{C\circ\mathcal{A}}\!\!\rightarrowtail b \cdot C(w')$ (by induction and (Vect Scale))
$= C(b \cdot w')$ (by Prop. 6.2–4).

Rule (Vect Add): $v' \ {}_{\mathcal{A}}\!\!\rightarrowtail w'$, $v'' \ {}_{\mathcal{A}}\!\!\rightarrowtail w'' \Rightarrow v' + v'' \rightarrowtail w' + w''$.
$C(v' + v'') = C(v') + C(v'') \ {}_{C\circ\mathcal{A}}\!\!\rightarrowtail C(w') + C(w'')$ (by induction and (Vect Add))
$= C(w' + w'')$ (by Prop. 6.2–4).

Rule (Vect Cross): $v' \ {}_{\mathcal{A}}\!\!\rightarrowtail w'$, $v'' \ {}_{\mathcal{A}}\!\!\rightarrowtail w'' \Rightarrow v' \times v'' \rightarrowtail w' \times w''$, and $C \in SE(3)$.
$C(v' \times v'') = C(v') \times C(v'') \ {}_{C\circ\mathcal{A}}\!\!\rightarrowtail C(w') \times C(w'')$ (by induction and (Vect Cross))

$= C(w' \times w'')$ (by Prop. 6.2–4).

Rule (Frame Shift)(σ=$\boldsymbol{v}$): $M \ {}_{\mathcal{A}}\!\!\rightarrowtail \mathcal{B}$, $v' \ {}_{\mathcal{A}\circ\mathcal{B}}\!\!\rightarrowtail w \Rightarrow M[v'] \ {}_{\mathcal{A}}\!\!\rightarrowtail w$.
Since $C \vDash M[v']$, we have $C \vDash M$ and $C \vDash v'$.
It follows that $C(M) \ {}_{C\circ\mathcal{A}}\!\!\rightarrowtail \mathcal{B}$ and $C(v') \ {}_{C\circ\mathcal{A}\circ\mathcal{B}}\!\!\rightarrowtail C(w)$ (by induction).
Hence $C(M[v']) = C(M)[C(v')] \ {}_{C\circ\mathcal{A}}\!\!\rightarrowtail C(w)$ by (Frame Shift).

Case (σ=$\boldsymbol{m}$): Show that $C \vDash M, \ M \ {}_{\mathcal{A}}\!\!\rightarrowtail \mathcal{B} \Rightarrow C(M) \ {}_{C\circ\mathcal{A}}\!\!\rightarrowtail \mathcal{B}$.
Then $M \ {}_{\mathcal{A}}\!\!\rightarrowtail \mathcal{B}$ is the consequent of Rule (Value) or one of the rules:

Rule (Map Given): $a_{ij} \ {}_{\mathcal{A}}\!\!\rightarrowtail b_{ij}$, $a_k \ {}_{\mathcal{A}}\!\!\rightarrowtail b_k \Rightarrow \langle a_{ij},a_k\rangle \ {}_{\mathcal{A}}\!\!\rightarrowtail \langle b_{ij},b_k\rangle$, for i,j,k$\in$1..3 and $det(b_{ij})\neq 0$.
Then $C(\langle a_{ij},a_k\rangle) = \langle C(a_{ij}),C(a_k)\rangle \ {}_{C\circ\mathcal{A}}\!\!\rightarrowtail \langle b_{ij},b_k\rangle$ (by induction and (Map Given))

Rule (Map Comp): $M' \ {}_{\mathcal{A}}\!\!\rightarrowtail \mathcal{B}'$, $M'' \ {}_{\mathcal{A}}\!\!\rightarrowtail \mathcal{B}'' \Rightarrow M'\circ M'' \ {}_{\mathcal{A}}\!\!\rightarrowtail \mathcal{B}'\circ\mathcal{B}''$
We have $C(M'\circ M'') = C(M') \circ C(M'') \ {}_{C\circ\mathcal{A}}\!\!\rightarrowtail \mathcal{B}'\circ\mathcal{B}''$ (by induction and (Map Comp)).

Rule (Map Inv): $M' \ {}_{\mathcal{A}}\!\!\rightarrowtail \mathcal{B}' \Rightarrow M'^{-1} \ {}_{\mathcal{A}}\!\!\rightarrowtail \mathcal{B}'^{-1}$
We have $C(M'^{-1}) = C(M')^{-1} \ {}_{C\circ\mathcal{A}}\!\!\rightarrowtail \mathcal{B}'^{-1}$ (by induction and (Map Inv)).

Rule (Frame Shift)(σ=$\boldsymbol{m}$): $M' \ {}_{\mathcal{A}}\!\!\rightarrowtail \mathcal{D}$, $M'' \ {}_{\mathcal{A}\circ\mathcal{D}}\!\!\rightarrowtail \mathcal{B} \Rightarrow M'[M''] \ {}_{\mathcal{A}}\!\!\rightarrowtail \mathcal{B}$.
Since $C \vDash M'[M'']$, we have $C \vDash M'$ and $C \vDash M''$.
It follows that $C(M') \ {}_{C\circ\mathcal{A}}\!\!\rightarrowtail \mathcal{D}$ and $C(M'') \ {}_{C\circ\mathcal{A}\circ\mathcal{D}}\!\!\rightarrowtail \mathcal{B}$ (by induction).
Hence $C(M'[M'']) = C(M')[C(M'')] \ {}_{C\circ\mathcal{A}}\!\!\rightarrowtail \mathcal{B}$ by (Frame Shift).
■

7.1–2 Theorem 4.1–4: Local Frame Shift
$M \ {}_{\mathcal{A}}\!\!\rightarrowtail \mathcal{B}, \ P \ {}_{\mathcal{A}\circ\mathcal{B}}\!\!\rightarrow Q \Rightarrow M[P] \ {}_{\mathcal{A}}\!\!\rightarrow M[Q]$

Proof
The proof is by induction on the derivation of $P \ {}_{\mathcal{A}\circ\mathcal{B}}\!\!\rightarrow Q$.

Rule (Red Comm): $\Delta \ {}_{\mathcal{A}\circ\mathcal{B}}\!\!\rightarrowtail \varepsilon \Rightarrow \ !_\sigma x(\Delta).P' + P'' \mid ?_\sigma x(y).Q' + Q'' \ {}_{\mathcal{A}\circ\mathcal{B}}\!\!\rightarrow \ P' \mid Q'\{y\backslash\varepsilon\}$
From $M \ {}_{\mathcal{A}}\!\!\rightarrowtail \mathcal{B}$, we obtain $M[\Delta] \ {}_{\mathcal{A}}\!\!\rightarrowtail \varepsilon$ by (Frame Shift). By (Red Comm) we than have:
$!_\sigma x(M[\Delta]).M[P'] + M[P''] \mid ?_\sigma x(y).M[Q'] + M[Q''] \ {}_{\mathcal{A}}\!\!\rightarrow \ M[P'] \mid M[Q']\{y\backslash\varepsilon\}$
Since $M \ {}_{\mathcal{A}}\!\!\rightarrowtail \mathcal{B}$, we know that M is closed.
Hence, for any variable y, we have $M[Q']\{y\backslash\varepsilon\} = M[Q'\{y\backslash\varepsilon\}]$.
Therefore, $M[!_\sigma x(\Delta).P' + P'' \mid ?_\sigma x(y).Q' + Q''] \ {}_{\mathcal{A}}\!\!\rightarrow \ M[P' \mid Q'\{y\backslash\varepsilon\}]$
by (≡ Map Sum), (≡ Map Out), (≡ Map In), (≡ Map Par) and (Red ≡).

Rule (Red Cmp): $\Delta_{\mathcal{A}\circ\mathcal{B}}\curvearrowright\curvearrowleft\Delta' \Rightarrow \Delta =_\sigma \Delta'.P' \ {}_{\mathcal{A}\circ\mathcal{B}}\!\!\rightarrow P'$
Since $M \ {}_{\mathcal{A}}\!\!\rightarrowtail \mathcal{B}$, we have $M[\Delta]_{\mathcal{A}}\curvearrowright\curvearrowleft M[\Delta']$ by (Frame Shift),
so from (Red Cmp) we obtain $M[\Delta] =_\sigma M[\Delta'].M[P'] \ {}_{\mathcal{A}}\!\!\rightarrow M[P']$.
Therefore $M[\Delta =_\sigma \Delta'.P'] \ {}_{\mathcal{A}}\!\!\rightarrow M[P']$ by (≡ Map Cmp) and (Red ≡).

Rule (Red Par): $P' \ {}_{\mathcal{A}\circ\mathcal{B}}\!\!\rightarrow Q' \Rightarrow P'|R \ {}_{\mathcal{A}\circ\mathcal{B}}\!\!\rightarrow Q'|R$
By induction $M[P'] \ {}_{\mathcal{A}}\!\!\rightarrow M[Q']$, hence $M[P']|M[R] \ {}_{\mathcal{A}}\!\!\rightarrow M[Q']|M[R]$ by (Red Par)
and $M[P'|R] \ {}_{\mathcal{A}}\!\!\rightarrow M[Q'|R]$ by (≡ Map Par) and (Red ≡).

Rule (Red Res): $P' \ {}_{\mathcal{A}\circ\mathcal{B}}\!\!\rightarrow Q' \Rightarrow (\nu x)P' \ {}_{\mathcal{A}\circ\mathcal{B}}\!\!\rightarrow (\nu x)Q'$

By induction $M[P'] \xrightarrow{\mathcal{A}} M[Q']$, hence $(\nu x)M[P'] \xrightarrow{\mathcal{A}} (\nu x)M[Q']$ by (Red Res) and $M[(\nu x)P'] \xrightarrow{\mathcal{A}} M[(\nu x)Q']$ by (≡ Map Res) and (Red ≡).

Rule (Red ≡): $P \equiv P',\ P' \xrightarrow{\mathcal{A}\circ\mathcal{B}} Q',\ Q' \equiv Q \Rightarrow P \xrightarrow{\mathcal{A}\circ\mathcal{B}} Q$

By (≡ Map), $M[P] \equiv M[P']$ and $M[Q'] \equiv M[Q]$. By induction $M[P'] \xrightarrow{\mathcal{A}} M[Q']$.
Hence $M[P] \xrightarrow{\mathcal{A}} M[Q]$ by (Red ≡).

■

The (≡ Map Comp) rule is not used in the proof of the theorem. This indicates that we might restrict ourselves to a d-π style calculus without the nesting of frames. In our nested calculus, the derived reduction for nested process frame, using Theorem 4.1–4 twice, is:

$$M \xrightarrow{\mathcal{A}} \mathcal{B},\ N \xrightarrow{\mathcal{A}\circ\mathcal{B}} C,\ P \xrightarrow{\mathcal{A}\circ\mathcal{B}\circ C} Q \Rightarrow N[P] \xrightarrow{\mathcal{A}\circ\mathcal{B}} N[Q] \Rightarrow M[N[P]] \xrightarrow{\mathcal{A}} M[N[Q]]$$

In a non-nested calculus, we could emulate this reduction, from the same assumptions, by:

$$M \xrightarrow{\mathcal{A}} \mathcal{B},\ N \xrightarrow{\mathcal{A}\circ\mathcal{B}} C,\ P \xrightarrow{\mathcal{A}\circ\mathcal{B}\circ C} Q \Rightarrow M[N] \xrightarrow{\mathcal{A}} C \Rightarrow M\circ M[N] \xrightarrow{\mathcal{A}} \mathcal{B}\circ C \Rightarrow (M\circ M[N])[P] \xrightarrow{\mathcal{A}} (M\circ M[N])[Q]$$

using (Frame Shift), (Map Comp) and Theorem 4.1–4. In other words, if we had neither (≡ Map Comp) nor nested process frames, we could still emulate M[N[P]] by (M∘M[N])[P]. But with 3 nested process frames, we end up with 3 nested frames on the maps. Hence we would still need to handle nested frames at least on the data.

7.1–3 Lemma: Congruence Mapping

$P \equiv Q \Rightarrow C(P) \equiv C(Q)$

Proof

The proof is by induction on the derivation of $P \equiv Q$.
The interesting rules are the (≡ Map …) rules; we look at two of them.

Rule (≡ Map): $P' \equiv Q' \Rightarrow M[P'] \equiv M[Q']$. By induction $C(P') \equiv C(Q')$,
hence $C(M)[C(P')] \equiv C(M)[C(Q')]$ by (≡ Map), that is $C(M[P']) \equiv C(M[Q'])$.

Rule (≡ Map In): $M[?_\sigma x(y).P'] \equiv ?_\sigma x(y).M[P']$ ($y \notin fv(M)$). Then $y \notin fv(C(M))$, and we have
$C(M[?_\sigma x(y).P']) = C(M)[?_\sigma x(y).C(P')] \equiv ?_\sigma x(y).C(M)[C(P')] = C(?_\sigma x(y).M[P'])$
by (≡ Map In).

■

The ⊨ relation is extended to the process syntax in the obvious way: $\mathcal{A} \models P$ holds if $\mathcal{A} \models \Delta$ holds for all data subterms Δ of P, where $\mathcal{A} \models \Delta$ is given in Definition 4.1–1.

7.1–4 Lemma

$P \equiv Q \Rightarrow (\mathcal{A} \models P \Leftrightarrow \mathcal{A} \models Q)$

Proof

The proof is by induction on the derivation of the derivation of $P \equiv Q$.
Rule (≡ Symm): $Q \equiv P \Rightarrow P \equiv Q$.
Then by induction we have that $Q \equiv P \Rightarrow (\mathcal{A} \models Q \Leftrightarrow \mathcal{A} \models P)$ and hence $\mathcal{A} \models P \Leftrightarrow \mathcal{A} \models Q$.
Rule (≡ Map): $P \equiv Q \Rightarrow M[P] \equiv M[Q]$
Then by induction we have $(\mathcal{A} \models P \Leftrightarrow \mathcal{A} \models Q)$, hence $(\mathcal{A} \models M[P] \Leftrightarrow \mathcal{A} \models M[Q])$.
The other cases are routine because of the same data subterms on both sides.

■

7.1–5 Lemma 4.1–5
$\mathcal{B}\models P, P \xrightarrow{\mathcal{A}} Q \Rightarrow \mathcal{B}\models Q$

Proof
Reduction does not introduce new subterms, except for (Red Comm) where the result follows from $\mathcal{B}\models\varepsilon$ and $\mathcal{B}\models Q \Rightarrow \mathcal{B}\models Q\{y\backslash\varepsilon\}$, and for (Red ≡) where the result follows from Lemma 7.1–4.
■

To motivate the theorem, assume the data computation $\Delta \xrightarrow{\mathcal{A}} \varepsilon$ which, by (Red Comm), implies the process reduction:

$$!c(\Delta) \mid ?c(x).x=\varepsilon' \xrightarrow{\mathcal{A}} \varepsilon=\varepsilon'$$

Also assume $C\models\Delta$, so we have $C(\Delta) \xrightarrow{C\circ\mathcal{A}} C(\varepsilon')$ by Theorem 4.1–3. Hence by (Red Comm):

$$!c(C(\Delta)) \mid ?c(x).x=C(\varepsilon') \xrightarrow{C\circ\mathcal{A}} C(\varepsilon)=C(\varepsilon')$$

and since $C(!c(\Delta) \mid ?c(x).x=\varepsilon') = !c(C(\Delta)) \mid ?c(x).x=C(\varepsilon')$ and $C(\varepsilon=\varepsilon') = C(\varepsilon)=C(\varepsilon')$, we have:

$$C(!c(\Delta) \mid ?c(x).x=\varepsilon') \xrightarrow{C\circ\mathcal{A}} C(\varepsilon=\varepsilon')$$

For this example we have shown that $P \xrightarrow{\mathcal{A}} Q \Rightarrow C(P) \xrightarrow{C\circ\mathcal{A}} C(Q)$. Although P has to be replaced by $C(P)$ in the shifted frame, the process shape P remains unchanged up to the embedded values. Moreover the change does not affect data comparisons in that, if the comparison $\varepsilon=\varepsilon'$ succeeds in $\mathcal{A}$, then the comparison $C(\varepsilon)=C(\varepsilon')$ succeeds in $C\circ\mathcal{A}$. This example suggests the statement of the following theorem.

7.1–6 Theorem 4.1–6: Global Frame Shift for Processes
$C\models P, \quad P \xrightarrow{\mathcal{A}} Q \Rightarrow C(P) \xrightarrow{C\circ\mathcal{A}} C(Q)$

Proof
The proof is by induction on the derivation of $P \xrightarrow{\mathcal{A}} Q$.

Rule (Red Comm): $\Delta \xrightarrow{\mathcal{A}} \varepsilon \Rightarrow !_\sigma x(\Delta).P' + P'' \mid ?_\sigma x(y).Q' + Q'' \xrightarrow{\mathcal{A}} P' \mid Q'\{y\backslash\varepsilon\}$, $C\models$l.h.s.
By Theorem 4.1–3, $C\models P'$, $\Delta \xrightarrow{\mathcal{A}} \varepsilon \Rightarrow C(\Delta) \xrightarrow{C\circ\mathcal{A}} C(\varepsilon)$.
Hence, we can produce the following instance of (Red Comm):
$!_\sigma x(C(\Delta)).C(P') + C(P'') \mid ?_\sigma x(y).C(Q') + C(Q'') \xrightarrow{C\circ\mathcal{A}} C(P') \mid C(Q')\{y\backslash C(\varepsilon)\}$
Since $C(Q')\{y\backslash C(\varepsilon)\} = C(Q'\{y\backslash\varepsilon\})$, it follows that
$C(!_\sigma x(\Delta).P' + P'' \mid ?_\sigma x(y).Q' + Q'') \xrightarrow{C\circ\mathcal{A}} C(P' \mid Q'\{y\backslash\varepsilon\})$

Rule (Red Cmp): $\Delta_{\mathcal{A}}{\curvearrowright}{\curvearrowleft}\Delta' \Rightarrow (\Delta=_\sigma\Delta'.Q) \xrightarrow{\mathcal{A}} Q$, with $C\models(\Delta=_\sigma\Delta'.Q)$.
By Theorem 4.1–3, since $C\models\Delta=_\sigma\Delta'$ and $\exists\varepsilon.\ \Delta \xrightarrow{\mathcal{A}} \varepsilon$ and $\Delta' \xrightarrow{\mathcal{A}} \varepsilon$,
we have that $\exists\varepsilon' = C(\varepsilon)$. $C(\Delta) \xrightarrow{C\circ\mathcal{A}} \varepsilon'$ and $C(\Delta') \xrightarrow{C\circ\mathcal{A}} \varepsilon'$;
hence $C(\Delta) \xrightarrow{C\circ\mathcal{A}}{\curvearrowright}{\curvearrowleft} C(\Delta')$. Therefore, by (Red Cmp) we obtain
$C(\Delta)=_\sigma C(\Delta').C(Q) \xrightarrow{C\circ\mathcal{A}} C(Q)$. It follows that $C(\Delta=_\sigma\Delta'.Q) \xrightarrow{C\circ\mathcal{A}} C(Q)$.

Rule (Red Par): $P' \xrightarrow{\mathcal{A}} Q' \Rightarrow P'|R \xrightarrow{\mathcal{A}} Q'|R$, with $C\models P'|R$.
By induction, since $C\models P'$, we have $C(P') \xrightarrow{C\circ\mathcal{A}} C(Q')$.
Hence by (Red Par), $C(P')|C(R) \xrightarrow{C\circ\mathcal{A}} C(Q')|C(R)$, that is, $C(P'|R) \xrightarrow{C\circ\mathcal{A}} C(Q'|R)$.

Rule (Red Res): $P' \ {}_{\mathcal{A}}\!\rightarrow\ Q' \ \Rightarrow\ (\nu x)P' \ {}_{\mathcal{A}}\!\rightarrow\ (\nu x)Q'$, with $C \vDash (\nu x)P'$.

By induction, since $C \vDash P'$, we have $C(P') \ {}_{C \circ \mathcal{A}}\!\rightarrow\ C(Q')$.

Hence by (Red Res) $(\nu x)C(P') \ {}_{C \circ \mathcal{A}}\!\rightarrow\ (\nu x)C(Q')$, that is, $C((\nu x)P') \ {}_{C \circ \mathcal{A}}\!\rightarrow\ C((\nu x)Q')$.

Rule (Red $\equiv$): $P \equiv P',\ P' \ {}_{\mathcal{A}}\!\rightarrow Q',\ Q' \equiv Q \ \Rightarrow\ P \ {}_{\mathcal{A}}\!\rightarrow\ Q$, with $C \vDash P$.

By Lemma 7.1–4, we have $C \vDash P$, $P \equiv P' \ \Rightarrow\ C \vDash P'$.

By induction, we have $C \vDash P',\ P' \ {}_{\mathcal{A}}\!\rightarrow Q' \ \Rightarrow\ C(P') \ {}_{C \circ \mathcal{A}}\!\rightarrow\ C(Q')$.

By Lemma 7.1–3, we have $C(P) \equiv C(P')$ and $C(Q') \equiv C(Q)$.

Hence, $C(P) \ {}_{C \circ \mathcal{A}}\!\rightarrow\ C(Q)$ by (Red $\equiv$).

■