Logical Types for Asynchronous Mobile Processes

Luís Dominguez, FCT, Universidade NOVA de Lisboa[1]
luidomi@gmail.com

António Ravara, NOVA-LINCS and DI-FCT, Universidade NOVA de Lisboa

2015 September 30

[1]Financiado por Bolsa de Investigação para Mestre da FCT/MCTES (PIDDAC) do projecto Liveness, Statically (PTDC/EIA-CCO/117513/2010)

Contents

Part I

Preliminaries and Overview

Chapter 1

Asynchronous Pi-Calculus

We consider the monadic asynchronous pi-calculus with input (guarded) replication and input (guarded) sum.

1.1 Syntax

$$\begin{array}{rcl} P, Q, R & ::= & 0 \mid \bar{a}a' \mid S \mid \ (!a(x).P) \mid (P|P') \mid ((\nu x : \sigma)P) \\ S & ::= & (a(x).P) \mid (S + S') \\ \sigma & ::= & \mid n \\ n & ::= & 0 \mid 1 + n \end{array}$$

Semi Sorted Restriction / Processes A semi sort σ is either ellided in an unsorted restriction, or a natural number in a sorted restriction, isomorphic to a simple sort of a monadic channel. We abbreviate $\boxed{(\nu x)P \stackrel{df}{=} (\nu x :)P}$ an unsorted (restriction) process.

We write $\boxed{?a(x).P}$ for either $a(x).P$ or $!a(x).P$.

We omit outer (and) assuming the next precedences: input . and replication ! have highest priority; then sum +; followed by parallel |; and finally restriction ν.

Write $\boxed{P(a/x)}$ for the non capturing substitution of a for all free occurrences of x in P. We omit its expected structural recursive definition.

1.1.1 Sets of Names in a Process

The next sets are defined as usual by structural recursion over the given process P. We omit the expected definitions.

$\boxed{na(P)} = fn(P) \cup bn(P)$ is the set of names of P; each either free or bound in P. As usual $sb(P)$ is the set of subject names of P and $ob(P)$ the set of its object names.

$\boxed{fn(P)}$ is the set of free names of P; ie whose occurrences in P are not bound by any outer binder.

$\boxed{bn(P)} = bi(P) \cup br(P)$ is the set of bound names of P; ie the set of names occurring in (replicated) input binders or restriction ones in P.

$\boxed{bi(P)}$ is the set of input bound names of P; ie the set of names occurring in (replicated) input guard binders in P.

$\boxed{br(P)} = ru(P) \cup rs(P)$ is the set of restricted names of P; ie the set of names occurring in unsorted or sorted restriction binders in P.

$\boxed{ru(P)}$ is the set of unsorted restricted names of P; ie the set of names occurring in unsorted restriction binders in P.

$\boxed{rs(P)}$ is the set of sorted restricted names of P; ie the set of names occurring in sorted restriction binders in P. Thus

$$na(P) = fn(P) \cup ru(P) \cup rs(P) \cup bi(P)$$

where $fn(P), ru(P), rs(P), bi(P)$ are mutually disjoint.

We write $\boxed{D \# D'}$ for the disjointness of name sets D and D'.

1.2 Alpha Congruence

$\boxed{\stackrel{\alpha}{=}}$ is the process relation $\mathcal{R}$ inductively defined by (ie the smallest one closed under) the following equivalence, compatibility and alpha conversion rules.

1.2.1 Equivalence Rules

$$\frac{}{P \ \mathcal{R} \ P} \qquad \frac{P \ \mathcal{R} \ Q}{Q \ \mathcal{R} \ P} \qquad \frac{P \ \mathcal{R} \ Q \quad Q \ \mathcal{R} \ R}{P \ \mathcal{R} \ R}$$

These equivalence rules are not process structural. Besides, using them backwards when proof searching, allows looping, in two steps by the symmetry rule, or in one step by the transitivity rule and reflexivity axiom. Thus back proving can be non well founded, though back checking a given forward proof must be well founded, by its inductive definition. So when back proving we regard such rules as declarative.

1.2.2 Compatibility Rules

$$\frac{}{0 \ \mathcal{R} \ 0} \qquad \frac{}{\bar{a}a' \ \mathcal{R} \ \bar{a}a'} \qquad \frac{P \ \mathcal{R} \ Q}{!a(x).P \ \mathcal{R} \ \ !a(x).Q}$$

$$\frac{P \ \mathcal{R} \ Q}{a(x).P \ \mathcal{R} \ a(x).Q} \qquad \frac{S_1 \ \mathcal{R} \ S_1' \quad S_2 \ \mathcal{R} \ S_2'}{(S_1 + S_2) \ \mathcal{R} \ (S_1' + S_2')}$$

$$\frac{P_1 \ \mathcal{R} \ Q_1 \quad P_2 \ \mathcal{R} \ Q_2}{(P_1|P_2) \ \mathcal{R} \ (Q_1|Q_2)} \qquad \frac{P \ \mathcal{R} \ Q}{(\nu x:\sigma)P \ \mathcal{R} \ (\nu x:\sigma)Q}$$

1.2.3 Alpha Conversion Rules

$$\frac{P(z/x) \ \mathcal{R} \ Q(z/y)}{a(x).P \ \mathcal{R} \ \ a(y).Q} z \notin \{x\} \cup \{y\} \cup na(P) \cup na(Q)$$

$$\frac{P(z/x) \ \mathcal{R} \ Q(z/y)}{!a(x).P \ \mathcal{R} \ \ !a(y).Q} z \notin \{x\} \cup \{y\} \cup na(P) \cup na(Q)$$

$$\frac{P(z/x) \ \mathcal{R} \ Q(z/y)}{(\nu x:\sigma)P \ \mathcal{R} \ \ (\nu y:\sigma)Q} z \notin \{x\} \cup \{y\} \cup na(P) \cup na(Q)$$

The compatibility and alpha conversion rules are process structural. Not only back checking, but also back proving is well founded and almost deterministic, up to the chosen fresh common renamed bound variable z. We prefer such structural, well founded and deterministic rules, better for the automatic static back proving we seek. We regard them as candidate algorithmic rules.

1.3 Structural Congruence

$\boxed{\equiv}$ is the process relation $\mathcal{R}$ inductively defined by the next structural laws and the former equivalence, compatibility and alpha conversion rules; ie it is the smallest one generated by these laws and closed under such rules.

1.3.1 Structural Laws

$$\begin{aligned}
0|P &\equiv P && (1.1)\\
P|Q &\equiv Q|P && (1.2)\\
(P|Q)|R &\equiv P|(Q|R) && (1.3)\\
(\nu x:\sigma)0 &\equiv 0 && (1.4)\\
(\nu x:\sigma)(\nu y:\sigma')P &\equiv (\nu y:\sigma')(\nu x:\sigma)P && (1.5)\\
x \notin fn(P) \Rightarrow rsx:\sigma P|Q &\equiv P|(\nu x:\sigma)Q && (1.6)\\
a \neq x \neq z \Rightarrow (\nu x:\sigma)a(z).P &\equiv a(z).(\nu x:\sigma)P && (1.7)\\
(a(x).P) + (a(x).P) &\equiv a(x).P && (1.8)\\
S + S' &\equiv S' + S && (1.9)\\
(S + S') + S'' &\equiv S + (S' + S'') && (1.10)\\
(a(x).P)|(!a(x).P) &\equiv (!a(x).P) && (1.11)
\end{aligned}$$

$\stackrel{\alpha}{=} \subseteq \equiv$; ie if $P \stackrel{\alpha}{=} Q$ then $P \equiv Q$.
This holds since $\equiv$ is closed under the same rules which generate $\stackrel{\alpha}{=}$.

1.4 Instantiation

If $P \equiv Q$ and $x \notin (bn(P) \cup bn(Q))$ then $P(a/x) \equiv Q(a/x)$.
By inspecting each equational law, considering the bound names assumption namely for the commutation of restriction and input guard (law 1.7).

$$\frac{P \equiv Q}{P(a/x) \equiv Q(a/x)} x \notin (bn(P) \cup bn(Q))$$

This may be proved sound by induction on the derivation of $P \equiv Q$.

1.5 Late Transition System with Scoped Output

We integrate free $\bar{a}a'$ and (semi sorted) bound output $\bar{a}(x : \sigma)$ as scoped output action $\rho\bar{a}a'$. Either $\rho\bar{a}a' = \bar{a}a'$ with void $\rho = \epsilon$, or $\rho\bar{a}a' = (\nu x : \sigma)\bar{a}x$ with $\rho = (\nu x : \sigma)$ and $a' = x$.

$$\text{action } \alpha \ ::= \ \tau \mid a(x) \mid \rho\bar{a}a'$$

$$\text{scope } \rho \ ::= \mid (\nu x : \sigma)$$

The calculus operational semantics is given by the late LTS whose transitions $\boxed{P \xrightarrow{\alpha} Q}$ are inductively defined by the next rules.

$$msg\frac{}{\bar{a}a' \xrightarrow{\bar{a}a'} 0}$$

$$inp\frac{}{a(x).P \xrightarrow{a(x)} P}$$

$$rep\frac{}{!a(x).P \xrightarrow{a(x)} P|!a(x).P}$$

$$sum\frac{S \xrightarrow{a(x)} P}{S + S' \xrightarrow{a(x)} P \quad S' + S \xrightarrow{a(x)} P}$$

$$open'\frac{P \xrightarrow{\bar{a}x} P'}{(\nu x : \sigma)P \xrightarrow{(\nu x:\sigma)\bar{a}x} P'} a \neq x$$

$$clom\frac{P \xrightarrow{\rho\bar{a}a'} P' \quad Q \xrightarrow{a(x)} Q'}{P|Q \xrightarrow{\tau} \rho(P'|Q'(a'/x)) \quad Q|P \xrightarrow{\tau} \rho(Q'(a'/x)|P')} bn(\rho)\#fn(Q)$$

$$par\frac{P \xrightarrow{\alpha} P'}{P|Q \xrightarrow{\alpha} P'|Q \quad Q|P \xrightarrow{\alpha} Q|P'} bn(\alpha)\#fn(Q)$$

$$res\frac{P \xrightarrow{\alpha} P'}{(\nu x : \sigma)P \xrightarrow{\alpha} (\nu x : \sigma)P'} x \notin na(\alpha)$$

The renaming a'/x in rule clom dispenses alpha-conversion and structural congruence rules.

With these 8 rules the (late) transition system dispenses alpha-conversion and structural congruence. Thus we have separated the static structural congruence from the dynamic transition system.

This presentation integrates the standard free and bound actions of late semantics for monadic pi-calculus.

The transition system is image-finite; since sums are finite and structural congruence is not explicitly built in the rules, namely alpha-congruence and for replication, which would turn it image infinite.

Structural congruence is a bisimulation in the late LTS.

Assume $P_1 \equiv P_2$, any $i \in \{1,2\}$ and $P_i \xrightarrow{\alpha} P_i'$. Then there is P_{3-i}' such that $P_{3-i}' \xrightarrow{\alpha} P_{3-i}'$ and $P_1' \equiv P_2'$.
Each i case may be proved by induction (essentially on the structure of P_i) on the derivations of $P_1 \equiv P_2$ and $P_i \xrightarrow{\alpha} P_i'$.

1.6 Port Notions and Notations

$$\text{port } p \ ::= \ a \mid \bar{a}$$

A channel with name a has an input port $p = a$ and an output port $p = \bar{a}$.

$\boxed{n(p)}$ is the name of port p, given by $n(a) \stackrel{df}{=} a$ and $n(\bar{a}) \stackrel{df}{=} a$.

$\boxed{sg(p)}$ is the polarity of port p, discriminating output $sg(\bar{a}) \stackrel{df}{=} +$ from input $sg(a) \stackrel{df}{=} -$ ports.

$\boxed{\bar{p}}$ is the (port) dual of port p; ie $(\bar{a}) \stackrel{df}{=} \bar{a}$ and $(\bar{\bar{a}}) \stackrel{df}{=} a$

$\boxed{a^p}$ p-like port of channel a; ie

$$a^p \stackrel{df}{=} a \text{ if } p = a'$$

$$a^p \stackrel{df}{=} \bar{a} \text{ if } p = \bar{a}'$$

$\boxed{\rho p a'}$ a port action is either: an input action $a(x)$, corresponding to $\rho = \epsilon$, $p = a$ and $a' = x$; or a scoped output action $\rho \bar{a} a'$, with $p = \bar{a}$.

1.7 Process Immediate Action Set

The immediate action set $\boxed{A(P)}$ of a process P is defined by structural recursion as follows.

$$\begin{aligned} A(0) &\stackrel{df}{=} \emptyset \\ A(\bar{a}a') &\stackrel{df}{=} \{\bar{a}a'\} \end{aligned}$$

$$
\begin{aligned}
A(!a(x).P) &\stackrel{df}{=} \{a(x)\} \\
A(a(x).P) &\stackrel{df}{=} \{a(x)\} \\
A(S_1 + S_2) &\stackrel{df}{=} \cup_{i\in\{1,2\}} A(S_i) \\
A((\nu x:\sigma).P) &\stackrel{df}{=} \{\alpha \in A(P) \mid x \notin na(\alpha)\} \cup \\
&\quad \{(\nu x:\sigma)\bar{a}x \mid a \neq x wedge (\nu x:\sigma)\bar{a}x \in A(P)\} \\
A(P_1|P_2) &\stackrel{df}{=} (\cup_{i\in\{1,2\}}\{\alpha \in A(P_i) \mid bn(\alpha)\#fn(P_{3-i})\}) \cup \\
&\quad \{\tau \mid \vee_{i\in\{1,2\}} \exists\rho\exists a\exists a'\exists x \\
&\quad (\rho\bar{a}a' \in A(P_i) wedge a(x) \in A(P_{3-i}))\}.
\end{aligned}
$$

$A(P) = \{\alpha \mid \exists Q(P \stackrel{\alpha}{\rightarrow} Q)\}$; ie $\alpha \in A(P)$ iff $P \stackrel{\alpha}{\rightarrow} Q$ for some Q.
This may be proved by induction on the (process subterm) structure of P.

[Finitely Branching] $A(P)$ is finite for every P.

This holds by induction on the structure of P. It is trivial for the first four process constructions. For binary sum, as by IH (induction hypothesis) each action set is finite, so must be their union. For restriction, also must be finite the union of both subsets of $A(P)$. For paralell composition, also must be finite the union of the three subsets: of $A(P_1)$, of $A(P_2)$ and of the silent (τ) action singleton.

$\boxed{\stackrel{A}{=}}$ Let $(P \stackrel{A}{=} Q) \stackrel{df}{=} (A(P) = A(Q))$. Eg $e(x).0 \stackrel{A}{=} !e(x).0$. But $e(x).0 \not\equiv !e(x).0$.

$\equiv \subset \stackrel{A}{=}$; ie If $P \equiv Q$ then $A(P) = A(Q)$, but not reciprocally.

This may be proved by induction on the derivation of $P \equiv Q$.

1.8 Process Image by an Action of the Late LTS

Given a set $\boxed{T}$ of processes, we abbreviate

$$
\begin{aligned}
(\nu x:\sigma)T &\stackrel{df}{=} \{(\nu x:\sigma)P \mid P \in T\} \\
T^{\psi} &\stackrel{df}{=} \{P \in T \mid \psi\} \\
&= \text{ if } \psi \text{ then } T \text{ else } \emptyset \\
(T|T') &\stackrel{df}{=} \{(P|Q) \mid P \in T wedge Q \in T'\} \\
T(a'/x) &\stackrel{df}{=} \{P(a'/x) \mid P \in T\}.
\end{aligned}
$$

These operations preserve finiteness (finite cardinality)

$$
\begin{aligned}
card((\nu x:\sigma)T) &= card(T) \\
card(T^{\psi}) &\leq card(T) \\
card(T|T') &= card(T) \times card(T') \\
card(T(e'/x)) &\leq card(T)
\end{aligned}
$$

of their results for finite T and T' arguments.

The image $\boxed{P \cdot \alpha}$ of a process P by an action α is defined by structural recursion as follows.

$$
\begin{array}{rcl}
0 \cdot \alpha & = & \emptyset \\
\bar{a}a' \cdot \alpha & \stackrel{df}{=} & \{0\} \text{ if } \alpha = \bar{a}a' \\
\bar{a}a' \cdot \alpha & \stackrel{df}{=} & \emptyset \text{ if } \alpha \neq \bar{a}a' \\
(!a(x).P) \cdot \alpha & \stackrel{df}{=} & \{P|!a(x).P\} \text{ if } \alpha = a(x) \\
(!a(x).P) \cdot \alpha & \stackrel{df}{=} & \emptyset \text{ if } \alpha \neq a(x) \\
(a(x).P) \cdot \alpha & \stackrel{df}{=} & \{P\} \text{ if } \alpha = a(x) \\
(a(x).P) \cdot \alpha & \stackrel{df}{=} & \emptyset \text{ if } \alpha \neq a(x) \\
(S + S') \cdot \alpha & \stackrel{df}{=} & (S \cdot \alpha) \cup (S' \cdot \alpha) \\
((\nu x : \sigma)P) \cdot \alpha & \stackrel{df}{=} & (\nu x : \sigma)(P \cdot \alpha) \text{ if } x \notin na(\alpha) \\
((\nu x : \sigma)P) \cdot (\nu x : \sigma)\bar{a}x & \stackrel{df}{=} & P \cdot \bar{a}x \text{ if } x \neq a \\
((\nu x : \sigma)P) \cdot \alpha & \stackrel{df}{=} & \emptyset \text{ if } x \in na(\alpha) wedge \\
 & & \forall a(a = x \vee \alpha \neq (\nu x : \sigma)\bar{a}x) \\
(P|Q) \cdot \alpha & \stackrel{df}{=} & ((P \cdot \alpha)|\{Q\})^{bn(\alpha)\#fn(Q)} \cup \\
 & & (\{P\}|(Q \cdot \alpha))^{bn(\alpha)\#fn(P)} \text{ if } \alpha \neq \tau \\
(P|Q) \cdot \tau & \stackrel{df}{=} & ((P \cdot \tau)|\{Q\}) \cup (\{P\}|(Q \cdot \tau)) \cup \\
 & & (\cup_{\rho\bar{a}a' \in A(P)} \cup_{a(x) \in A(Q)} \\
 & & (\rho(P \cdot \rho\bar{a}a'|(Q \cdot a(x))(a'/x)))^{bn(\rho)\#fn(Q)} \cup \\
 & & (\cup_{a(x) \in A(P)} \cup_{\rho\bar{a}a' \in A(Q)} \\
 & & (\rho((P \cdot a(x))(a'/x)|(Q \cdot \rho\bar{a}a')))^{bn(\rho)\#fn(P)}.
\end{array}
$$

The parallel composition image rules amount to

$$
\begin{array}{rcl}
(P|Q) \cdot \alpha & = & \{(P'|Q) \mid bn(\alpha)\#fn(Q) wedge P' \in (P \cdot \alpha)\} \cup \\
 & & \{(P|Q') \mid bn(\alpha)\#fn(P) wedge Q' \in (Q \cdot \alpha)\} \text{ if } \alpha \neq \tau \\
(P|Q) \cdot \tau & = & \{(P'|Q) \mid P' \in (P \cdot \tau)\} \cup \\
 & & \{(P|Q') \mid Q' \in (Q \cdot \tau)\} \cup \\
 & & (\cup_{\rho\bar{a}a' \in A(P)} \cup_{a(x) \in A(Q)} \\
 & & \{\rho(P'|Q'(a'/x)) \mid bn(\rho)\#fn(Q) wedge \\
 & & P' \in (P \cdot \rho\bar{a}a') wedge Q' \in (Q \cdot a(x))\}) \cup \\
 & & (\cup_{a(x) \in A(P)} \cup_{\rho\bar{a}a' \in A(Q)} \\
 & & \{\rho(P'(a'/x)|Q') \mid bn(\rho)\#fn(P) wedge \\
 & & P' \in (P \cdot a(x)) wedge Q' \in (Q \cdot \rho\bar{a}a')\}).
\end{array}
$$

$P \cdot \alpha = \{Q \mid P \xrightarrow{\alpha} Q\}$; ie $Q \in (P \cdot \alpha)$ iff $P \xrightarrow{\alpha} Q$.

This may be proved by induction on the (process subterm) structure of P.

If $Q \in (P \cdot \alpha)$ then $\alpha \in A(P)$.

If $\alpha \notin A(P)$ then $P \cdot \alpha = \emptyset$.

Thus $A(0) = \emptyset$ determined the first rule $0 \cdot \alpha = \emptyset$.

$P \cdot \alpha$ is finite for every P and α.

This holds by induction on the structure of P. It is trivial for the empty and singleton non recursive cases. For binary sum, as by IH (induction hypothesis) the image of each summand is finite, so must be the union of both images. For restriction, there are three cases: the empty image one is trivial as already mentioned; the first case image is equipotent to $P \cdot \alpha$ which is finite by IH; and the second case image $P \cdot \bar{a}x$ is also finite by IH. For parallel composition, the first case holds since binary union, disjointness conditions and composition preserve finiteness of $P \cdot \alpha$ and $Q \cdot \alpha$ which both hold by IH. The other case for silent actions, not only holds since binary unions and composition preserve finiteness of $P \cdot \tau$ and $Q \cdot \tau$ both true by IH, but further stems from the finite index sets $A(P)$ and $A(Q)$ of the unions which, as so do binary union, scoping, disjointness conditions, composition and name substitution, preserve the finiteness of $P \cdot \rho\bar{a}a'$ and $Q \cdot a(x)$ and $P \cdot a(x)$ and $Q \cdot \rho\bar{a}a'$, which hold by IH.

1.9 Process (Set) Image by an Action (Set)

$\boxed{P \cdot A}$ Generalising for action sets A,

$P \cdot A \stackrel{df}{=} \cup_{\alpha \in A}(P \cdot \alpha) = \{Q \mid \exists(\alpha \in A)Q \in (P \cdot \alpha)\}$.

$P \cdot A = P \cdot (A(P) \cap A) = \cup_{\alpha \in A(P) \cap A}(P \cdot \alpha)$.

Since $A(P)$ is finite, then $P \cdot A$ is a finitely indexed union of finite images $P \cdot \alpha$, for a image finite LTS as we proved this late one to be. So

%begincl $P \cdot A$ is a finite set for every P and A.

$\boxed{T \cdot A}$ Further generalising for process sets T,

$T \cdot A \stackrel{df}{=} \cup_{P \in T}(P \cdot A) = \cup_{P \in T} \cup_{\alpha \in A} (P \cdot \alpha) = \cup_{P \in T} \cup_{\alpha \in A(P) \cap A} (P \cdot \alpha)$.

$T \cdot A$ is a finite set for every A and finite set T.

$\boxed{P|_i Q}$ Let $(P|_1 Q) \stackrel{df}{=} (P|Q)$ and $(P|_2 Q) \stackrel{df}{=} (Q|P)$.

We write $d(\Sigma)$ for the domain of sorting Σ; ie the set of names which the sorting maps to numbers (coding simple sorts).

Part II

Logical Fragments

Chapter 2

Basic Fragments

2.1 Syntax

$$
\begin{array}{rcl}
(p)\ \phi & ::= & tt \mid ff \mid \phi wedge \phi \mid \phi \vee \phi \mid \neg\phi \mid \phi \Rightarrow \phi \mid \phi \Leftrightarrow \phi \\
(q)\ \phi & ::= & \forall(a)\phi \mid \forall(P)\phi \mid \forall(\alpha)\phi \mid \forall(\sigma)\phi \mid \forall(\rho)\phi \mid \forall(p)\phi \mid \\
 & & \exists(a)\phi \mid \exists(P)\phi \mid \exists(\alpha)\phi \mid \exists(\sigma)\phi \mid \exists(\rho)\phi \mid \exists(p)\phi \\
(z)\ \phi & ::= & Z \mid \forall(Z)\phi \mid \exists(Z)\phi \\
(u)\ \phi & ::= & \langle\equiv\rangle\phi \mid [\equiv]\phi \\
(e)\ \phi & ::= & 0 \mid \bar{a}a' \mid (!a(x).\phi) \mid \varphi \mid (\phi|\phi) \mid ((\nu x:\sigma)\phi) \mid \\
 & & \%\phi \mid [\phi]\phi \mid (x:\sigma)@\phi \mid (x:\sigma)\oslash\phi \mid \\
 & & (\phi||\phi) \mid \langle\phi\rangle\phi \\
\varphi & ::= & \mathrm{sum} \mid (a(x).\phi) \mid (\varphi + \varphi) \\
(n)\ \phi & ::= & (aa)\phi \mid N(x)\phi \mid \copyright a \mid \\
 & & a = a \mid \alpha = \alpha \mid p = p \mid sg(p) = sg(p) \\
(i)\ \phi & ::= & (\alpha = \alpha) \mid \alpha \in A \mid (A = A) \mid (K = K) \mid P \in B \mid \\
 & & (\hat{a} = \hat{a}) \mid \hat{a} \in D \mid (D = D) \mid (\Sigma = \Sigma) \mid \\
A & ::= & \{\tilde{\alpha}\} \mid A_P \mid (K)_P \mid (K) \\
B & ::= & \{\tilde{P}\} \mid (P \cdot \alpha) \mid (B \cdot A) \\
C & ::= & na \mid fn \mid bn \mid bi \mid br \mid ru \mid rs \\
D & ::= & \{\tilde{a}\} \mid C(P) \mid C(\alpha) \mid C\rho \\
\hat{a} & ::= & a \mid n(q) \mid na(\alpha) \mid s(\alpha) \mid o(\alpha) \\
q & ::= & p \mid p^p \mid \overline{q} \\
p & ::= & a \mid \bar{a} \\
a & ::= & a \mid b \mid \ldots \mid x \mid y \mid z \mid \ldots
\end{array}
$$

2.1.1 Propositional

The usual propositional connectives.

But for negation, we may regard it as an operation over formulas, corresponding to their classical duality.

2.1.2 FO

The usual first order quantifiers, but over: process, name, sort, scope, action, port, (meta)variables.

2.1.3 SO

The usual second order quantifiers, but over process sets.

2.1.4 Pre

This novel fragment isolates structural congruence reasoning from the other fragments. We may abbreviate $\langle \equiv \circ \mathcal{R} \rangle \phi \stackrel{df}{=} \langle \equiv \rangle \langle \mathcal{R} \rangle \phi$ and $\langle L \circ \equiv \rangle \phi \stackrel{df}{=} \langle L \rangle \langle \equiv \rangle \phi$ for a process relation $\mathcal{R}$, like $\equiv$, or a transition label, namely α. Eg $\langle \equiv \circ \equiv \rangle \phi \stackrel{df}{=} \langle \equiv \rangle \langle \equiv \rangle \phi$ and $\langle \equiv \circ \alpha \circ \equiv \rangle \phi \stackrel{df}{=} \langle \equiv \rangle \langle \alpha \rangle \langle \equiv \rangle \phi$.

Up to (structural congruence) means closed under relational composition with structural congruence on the left (pre) and right (post).

Post (up to) means up to just on the right.

Pre (up to) means up to just on the left.

2.1.5 Pattern

The productions of the first line of the ϕ grammar (including both rules for φ) generate all processes. So any well formed process P is also a pattern formula; ie it would be redundant to add $\phi ::= \ldots \mid P$ or the next axiom.

$$\frac{}{P : P}$$

The particular case of (positive) sum formulas are restricted by the productions for the auxilliary meta non terminal symbol φ. In particular any well formed (positive) sum S is a sum formula; also dispensing $\phi ::= \ldots \mid S$.

2.1.6 Nominal

Name (In)Equality We take as primitive both next complementary binary predicates over name terms: $a = a'$, $a \neq a'$.

Action (In)Equality We may also regard as primitive the (in)equality binary predicates over actions $(\alpha \neq \alpha')$ $\alpha = \alpha'$. The latter is given by the next rules.

$$\frac{}{\tau = \tau} \qquad \frac{}{a(x) = a(x)} \qquad \frac{}{\rho\bar{a}a' = \rho\bar{a}a'}$$

Thus action inequality could be given as follows.

$$\frac{a(x) = \alpha}{\tau \neq \alpha} \qquad \frac{\rho\bar{a}a' = \alpha}{\tau \neq \alpha}$$

$$\frac{\tau = \alpha}{a(x) \neq \alpha} \qquad \frac{\rho\bar{a}a' = \alpha}{a(x) \neq \alpha} \qquad \frac{a'(x') = \alpha}{a(x) \neq \alpha}(a \neq a') \vee (x \neq x')$$

$$\frac{\tau = \alpha}{\rho\bar{a}a' \neq \alpha} \qquad \frac{a(x) = \alpha}{\rho\bar{a}a' \neq \alpha} \qquad \frac{\rho'\bar{a}''a''' = \alpha}{\rho\bar{a}a' \neq \alpha}(\rho \neq \rho') \vee (a \neq a'') \vee (a' \neq a''')$$

2.1.7 Primitive

This fragment gathers (atomic) formulas concerning (meta) finite sets of syntactic elements. Such formulas are taken as primitive for convenience, though most could be derived from the other basic fragments.

2.2 Semantics

2.2.1 Propositional

Standard, interpreted classically (dually).

2.2.2 FO

Also standard.

2.2.3 SO

Interpreted over process sets.

2.2.4 Pre

Let $P^{\equiv} \stackrel{df}{=} (\equiv P) \stackrel{df}{=} \{Q \mid Q \equiv P\}$.

$$\begin{array}{rcl} \|\langle\equiv\rangle\phi\| & \stackrel{df}{=} & \{P \mid \exists(Q)(P \equiv Q)wedge(Q \in \|\phi\|)\} \\ & = & \{P \mid \emptyset \subset P^{\equiv} \cap \|\phi\|\} \\ & = & \{P \mid P^{\equiv} \not\subseteq \neg\|\phi\|\} \\ \|[\equiv]\phi\| & \stackrel{df}{=} & \{P \mid \forall(Q)(P \equiv Q) \Rightarrow (Q \in \|\phi\|)\} \\ & = & \{P \mid P^{\equiv} \subseteq \|\phi\|\}. \end{array}$$

2.2.5 Pattern

$\boxed{\|\varphi\|}$

$$\begin{aligned}\|\text{sum}\| &\stackrel{df}{=} \{P \mid \exists S_1 \exists S_2 (P = (S_1 + S_2))\} \\ \|\varphi_1 + \varphi_2\| &\stackrel{df}{=} \{S_1 + S_2 \mid wedge_{i\in\{1,2\}}(S_i \in \|\varphi_i\|)\}\end{aligned}$$

$\boxed{\|\phi\|}$

$$\begin{aligned}\|0\| &\stackrel{df}{=} \{0\} \\ \|\bar{a}a'\| &\stackrel{df}{=} \{\bar{a}a'\} \\ \|(\phi_1|\phi_2)\| &\stackrel{df}{=} \{(P_1|P_2) \mid wedge_{i\in\{1,2\}}(P_i \in \|\phi_i\|)\} \\ \|(x:\sigma)@\phi\| &\stackrel{df}{=} \{(\nu x:\sigma)P \mid P \in \|\phi\|\} \\ \|(x:\sigma)\oslash\phi\| &\stackrel{df}{=} \{P \mid ((\nu x:\sigma)P) \in \|\phi\|\} \\ \|(\phi_1||\phi_2)\| &\stackrel{df}{=} \{P \mid \forall P_1 \forall P_2 (((P_1|P_2) = P) \Rightarrow \\ &\qquad (\vee_{i\in\{1,2\}} P_i \notin \|\phi_i\|))\} \\ \|\%\phi\| &\stackrel{df}{=} \{(P_1|P_2) \mid (P_2|P_1) \in \|\phi\|\} \\ \|[\phi]\phi'\| &\stackrel{df}{=} \{P \mid \forall Q((Q \in \|\phi\|) \Rightarrow ((P|Q) \in \|\phi'\|))\} \\ \|\langle\phi\rangle\phi'\| &\stackrel{df}{=} \{P \mid \exists Q((Q \in \|\phi\|) wedge((P|Q) \in \|\phi'\|))\}.\end{aligned}$$

Without Alpha Conversion More simply the remaining cases would be:

$$\begin{aligned}\|a(x).\phi\| &\stackrel{df}{=} \{a(x).P \mid P \in \|\phi\|\} \\ \|!a(x).\phi\| &\stackrel{df}{=} \{!a(x).P \mid P \in \|\phi\|\} \\ \|(\nu x:\sigma)\phi\| &\stackrel{df}{=} \{(\nu x:\sigma)P \mid P \in \|\phi\|)\}.\end{aligned}$$

With Alpha Conversion Preferably we take them to be:

$$\|a(x).\phi\| \stackrel{df}{=} \{a(y).P \mid N^{x,\phi}_{y,P}(z)(P(z/y) \in \|\phi(z/x)\|)\}$$

$$\begin{aligned}\|!a(x).\phi\| &\stackrel{df}{=} \{!a(y).P \mid N^{x,\phi}_{y,P}(z)(P(z/y) \in \|\phi(z/x)\|)\} \\ \|(\nu x:\sigma)\phi\| &\stackrel{df}{=} \{(\nu y:\sigma)P \mid N^{x,\phi}_{y,P}(z)(P(z/y) \in \|\phi(z/x)\|)\}.\end{aligned}$$

In particular for $\phi = P$ the following hold.
Without alpha conversion $\|P\| = \{P\}$.

Let $P^{\stackrel{\alpha}{=}} \stackrel{df}{=} (\stackrel{\alpha}{=} P) \stackrel{df}{=} \{Q \mid Q \stackrel{\alpha}{=} P\}$.

With alpha conversion $\|P\| = P^{\stackrel{\alpha}{=}}$.

Both may be proved by induction on the structure of P.

$(x:\sigma)@(+)$ is the lower adjoint of $(x:\sigma)\oslash(+)$

$$\|(x:\sigma)@\phi\| \subseteq \|\phi'\| \text{ iff } \|\phi\| \subseteq \|(x:\sigma)\oslash\phi'\|.$$

Ie for every P, $(\nu x:\sigma)P \in \|\phi\|$ iff $P \in \|(x:\sigma)\oslash\phi\|$.

The right parallel upper adjoint is right rely (guarantee)

$$\|\phi|\phi'\| \subseteq \|\phi''\| \text{ iff } \|\phi\| \subseteq \|[\phi']\phi''\|.$$

The left parallel upper adjoint amounts to rely swap

$$\|\phi|\phi'\| \subseteq \|\phi''\| \text{ iff } \|\phi'\| \subseteq \|[\phi]\%\phi''\|.$$

Thus the following are equivalent:

1. $Q \in \|[P]\%\phi\|$
2. $(Q|P) \in \|\%\phi\|$
3. $(P|Q) \in \|\phi\|$
4. $P \in \|[Q]\phi\|$.

2.2.6 Nominal

Informal Fresh Name Quantification The meta notation $(N^{\tilde{x},\tilde{\phi}}_{\tilde{y},\tilde{P}}(z)\ldots)$ stands for both

$$\exists(z)((z\#^{\tilde{x},\tilde{\phi}}_{\tilde{y},\tilde{P}})wedge\ldots)$$

and the equivalent

$$\forall(z)((z\#^{\tilde{x},\tilde{\phi}}_{\tilde{y},\tilde{P}}) \Rightarrow \ldots)$$

where the name disjointness # atom requires z to be fresh wrt the tuples $\tilde{x},\tilde{\phi},\tilde{y},\tilde{P}$. For clarity we may separate: in superscript formulas and related name variables; in subscript processes and related name variables.

Permutation Substitutions A permutation $P(aa')$ is definable by the ring-like renaming $P(z/a/a'/z) \stackrel{df}{=} P(z/a)(a/a')(a'/z)$, implemented with name substitutions, provided z is fresh wrt P, a, a'.

Semantics For primitive ϕ_{atom}, like $a = a'$, false denotes the emptyset and dually; ie true denotes the process universe.

$$\begin{array}{rcl}
\|\phi_{atom}\| & \stackrel{df}{=} & \emptyset \text{ if } \phi_{atom} \text{ is false} \\
\|\phi_{atom}\| & \stackrel{df}{=} & Pr \text{ if } \phi_{atom} \text{ is true} \\
\|p = p'\| & \stackrel{df}{=} & Pr \text{ if } n(p) = n(p') \text{ and } sg(p) = sg(p') \\
\|p = p'\| & \stackrel{df}{=} & \emptyset \text{ if } n(p) \neq n(p') \text{ or } sg(p) \neq sg(p') \\
\|p \neq p'\| & \stackrel{df}{=} & Pr \setminus \|p = p'\| \\
\|©a\| & \stackrel{df}{=} & \{P \mid a \in fn(P)\} \\
\|N(x)\phi\| & \stackrel{df}{=} & \{P \mid N_P^{x,\phi}(z)(P \in \|\phi(z/x)\|)\} \\
\|(aa')\phi\| & \stackrel{df}{=} & \{P \mid N^{a,a',\phi}(z)(P(z/a/a'/z) \in \|\phi\|)\}.
\end{array}$$

2.2.7 Primitive

Its semantics is implicit from preliminary definitions of meta notation and sets, as well as some syntactic sets and finitary operations given below, namely K in the action set (based multi) modal fragment k.

2.3 Duality

$$\boxed{\neg\phi \stackrel{df}{=} \phi^{\neg}}$$

2.3.1 Propositional

$$\begin{array}{rcl}
\neg ff & \stackrel{df}{=} & tt \\
\neg tt & \stackrel{df}{=} & ff \\
\neg(\phi \Rightarrow \phi') & \stackrel{df}{=} & \phi wedge \neg\phi' \\
\neg(\phi \iff \phi') & \stackrel{df}{=} & \phi \iff \neg\phi' \\
 & \vdots & .
\end{array}$$

2.3.2 FO

$$\begin{array}{rcl}
\neg\forall a\phi & \stackrel{df}{=} & \exists a\neg\phi \\
\neg\exists a\phi & \stackrel{df}{=} & \forall a\neg\phi \\
 & \vdots & .
\end{array}$$

2.3.3 SO

$$\begin{aligned}\neg\forall Z\phi &\stackrel{df}{=} \exists Z\neg\phi\\ \neg\exists Z\phi &\stackrel{df}{=} \forall Z\neg\phi.\end{aligned}$$

2.3.4 Pre

$$\begin{aligned}\neg\langle\equiv\rangle\phi &\stackrel{df}{=} [\equiv]\neg\phi\\ \neg[\equiv]\phi &\stackrel{df}{=} \langle\equiv\rangle\neg\phi.\end{aligned}$$

2.3.5 Pattern

Abbreviate

$$\begin{aligned}Msg &\stackrel{df}{=} \exists a\exists a'(\bar{a}a')\\ Inp &\stackrel{df}{=} \exists a\exists x(a(x).tt)\\ Rep &\stackrel{df}{=} \exists a\exists x(!a(x).tt)\\ Res &\stackrel{df}{=} \exists x\exists\sigma((\nu x:\sigma)tt).\end{aligned}$$

Homo Duals

$$\begin{aligned}\neg\%\phi &\stackrel{df}{=} \%\neg\phi\\ \neg[\phi]\phi' &\stackrel{df}{=} \langle\phi\rangle\neg\phi'\\ \neg\langle\phi\rangle\phi' &\stackrel{df}{=} [\phi]\neg\phi'\\ \neg(x:\sigma)\oslash\phi &\stackrel{df}{=} (x:\sigma)\oslash\neg\phi\\ \neg(\phi|\phi') &\stackrel{df}{=} (\neg\phi)||(\neg\phi')\\ \neg(\phi||\phi') &\stackrel{df}{=} (\neg\phi)|(\neg\phi').\end{aligned}$$

Non Homo Duals

$$\begin{aligned}\neg 0 &\stackrel{df}{=} (tt|tt)\vee\mathrm{sum}\vee Inp\vee Msg\vee Rep\vee Res\\ \neg\bar{a}a' &\stackrel{df}{=} 0\vee\mathrm{sum}\vee(tt|tt)\vee Inp\vee Res\vee Rep\\ &\quad(\exists a''\exists a'''((a''\neq a\vee a'''\neq a')wedge(\bar{a}''a''')))\\ \neg!a(x).\phi &\stackrel{df}{=} 0\vee\mathrm{sum}\vee(tt|tt)\vee Msg\vee Inp\vee Res\vee\\ &\quad(\exists a'(a'\neq awedge\ !a'(x).tt))\vee\ !a(x).\neg\phi\\ \neg(\nu x:\sigma)\phi &\stackrel{df}{=} 0\vee\mathrm{sum}\vee(tt|tt)\vee Msg\vee Inp\vee Rep\vee\\ &\quad((\nu x:\sigma)\neg\phi).\end{aligned}$$

Sum Duals $\boxed{\neg\varphi}$

$$\begin{array}{rcl}
\neg\text{sum} & \stackrel{df}{=} & 0 \vee (tt|tt) \vee Msg \vee Inp \vee Rep \vee Res \\
\neg(\varphi + \varphi') & \stackrel{df}{=} & 0 \vee (tt|tt) \vee Msg \vee Inp \vee Rep \vee Res \vee \\
 & & (\varphi + \neg\varphi') \vee ((\neg\varphi) + \varphi') \vee ((\neg\varphi) + \neg\varphi') \\
\neg a(x).\phi & \stackrel{df}{=} & 0 \vee \text{sum} \vee (tt|tt) \vee Msg \vee Rep \vee Res \vee \\
 & & (\exists a'(a' \neq awedgea'(x).tt)) \vee a(x).\neg\phi.
\end{array}$$

Homo duals distribute negation keeping formula size.

Non homo and sum duals increase the formula size, but the negation also recurs structurally on few smaller subformulas.

2.3.6 Nominal

$$\begin{array}{rcl}
\neg N(x)\phi & \stackrel{df}{=} & N(x)\neg\phi \\
\neg(aa')\phi & \stackrel{df}{=} & (aa')\neg\phi \\
\neg(a = a') & \stackrel{df}{=} & (a \neq a') \\
\neg(a \neq a') & \stackrel{df}{=} & (a = a') \\
 & \vdots & .
\end{array}$$

The defined negation operation is terminating, besides deterministic.

2.4 Formula Equivalence

[Formula Equivalence] $(\phi \stackrel{f}{=} \phi') \stackrel{df}{=} (\|\phi\| = \|\phi'\|)$; ie
$\forall(P)((P \in \|\phi\|) \iff (P \in \|\phi'\|))$.

If $\neg\phi \stackrel{df}{=} \phi^{\neg}$ then $\neg\phi \stackrel{f}{=} \phi^{\neg}$; ie $\|\neg\phi\| = \|\phi^{\neg}\| = Pr \setminus \|\phi\|$.

This may be proved by induction on the structure of ϕ.

2.4.1 Pre

$\langle\equiv\rangle\langle\equiv\rangle\phi \stackrel{f}{=} \langle\equiv\rangle\phi$.

This stems from $\equiv$ transitivity.

$[\equiv][\equiv]\phi \stackrel{f}{=} [\equiv]\phi$.

2.4.2 Pattern

[Swap involution] $\%\%\phi \stackrel{f}{=} \phi$.

2.4.3 Pre vs Pattern

$\langle\equiv\rangle\%\phi \stackrel{f}{=} \langle\equiv\rangle\phi \stackrel{f}{=} \%\langle\equiv\rangle\phi.$

$[\equiv]\%\phi \stackrel{f}{=} [\equiv]\phi \stackrel{f}{=} \%[\equiv]\phi.$

$\langle\equiv\rangle(x:\sigma)\oslash\phi \stackrel{f}{=} (x:\sigma)\oslash\langle\equiv\rangle\phi.$

$\langle\phi\rangle\langle\equiv\rangle\phi' \stackrel{f}{=} \langle\equiv\rangle\langle\phi\rangle\langle\equiv\rangle\phi'.$

$[\phi][\equiv]\phi' \stackrel{f}{=} [\equiv][\phi][\equiv]\phi'.$

$\langle\equiv\rangle\langle\phi\rangle\phi' \stackrel{f}{\neq} \langle\phi\rangle\langle\equiv\rangle\phi'.$

$[\equiv][\phi]\phi' \stackrel{f}{\neq} [\phi][\equiv]\phi'.$

2.5 Satisfaction

2.5.1 Propositional

Standard rules for propositional connectives; but for negation

$$(\substack{\leq\\ \equiv})\frac{P:\phi^{\neg}}{P:\neg\phi}\neg\phi \stackrel{df}{=} \phi^{\neg}$$

Although this rule may increase the formula size, negation is pushed into smaller subformulas, until in each proof branch it is eventually eliminated and only the other rules could apply. So with negation and this rule, the decidability of the chosen proof system depends on the decidability of the remaining rules.

2.5.2 FO

Standard rules and provisos, adapted for the quantified (meta) variables.

2.5.3 SO

Standard rules and provisos.

2.5.4 Pre

Existential

$$(\substack{\geq\\ \leqq\\ >})\frac{\exists Q((P\equiv Q)wedge(Q:\phi))}{P:\langle\equiv\rangle\phi}$$

Universal

$$(\substack{\geq\\ \leq\\ >}) \frac{\forall Q((P \equiv Q) \Rightarrow (Q : \phi))}{P : [\equiv]\phi}$$

$P : \langle\equiv\rangle Q$ iff $P \equiv Q$.

2.5.5 Pattern

Process Constructors

$$() \frac{}{0 : 0}$$

$$() \frac{}{\bar{a}a' : \bar{a}a'}$$

$$(\substack{\geq\\ >}) \frac{P_1 : \phi_1 \quad P_2 : \phi_2}{(P_1|P_2) : (\phi_1|\phi_2)}$$

Proper Sum

$$() \frac{}{S_1 + S_2 : \text{sum}}$$

$$(\substack{\geq\\ >}) \frac{S_1 : \varphi_1 \quad S_2 : \varphi_2}{(S_1 + S_2) : (\varphi_1 + \varphi_2)}$$

Without Alpha Conversion

$$(\substack{\geq\\ >}) \frac{P : \phi}{a(x).P : a(x).\phi}$$

$$(\substack{\geq\\ >}) \frac{P : \phi}{!a(x).P : !a(x).\phi}$$

$$(\substack{\geq\\ >}) \frac{P : \phi}{(\nu x : \sigma)P : (\nu x : \sigma)\phi}$$

With Alpha Conversion

$$(\substack{\geq\\ >}) \frac{P(z/x) : \phi(z/y)}{a(x).P : a(y).\phi} z \notin \{x\} \cup \{y\} \cup (na(P) \cup na(\phi))$$

$$(\substack{\geq\\ >}) \frac{P(z/x) : \phi(z/y)}{!a(x).P : !a(y).\phi} z \notin \{x\} \cup \{y\} \cup (na(P) \cup na(\phi))$$

$$(\substack{\geq\\ >}) \frac{P(z/x) : \phi(z/y)}{(\nu x : \sigma)P : (\nu y : \sigma)\phi} z \notin \{x\} \cup \{y\} \cup (na(P) \cup na(\phi))$$

Revelation and its upper adjoint

$$(\begin{smallmatrix}\geq\\>\end{smallmatrix})\frac{P:\phi}{(\nu x:\sigma)P:(x:\sigma)@\phi}$$

$$=(\begin{smallmatrix}\geq\\<\end{smallmatrix})\frac{(\nu x:\sigma)P:\phi}{P:(x:\sigma)\oslash\phi}$$

Decomposition $\boxed{P:(\phi||\phi')}$

$$()\frac{}{0:(\phi||\phi')}$$

$$()\frac{}{\bar{a}a':(\phi||\phi')}$$

$$()\frac{}{S:(\phi||\phi')}$$

$$()\frac{}{!a(x).P:(\phi||\phi')}$$

$$()\frac{}{(\nu x:\sigma)P:(\phi||\phi')}$$

$$(\begin{smallmatrix}\geq\\>\end{smallmatrix})\frac{P_i:\phi_i}{(P_1|P_2):(\phi_1||\phi_2)}i\in\{1,2\}$$

Swap, Rely and its Dual Pick

$$swap(\begin{smallmatrix}\geq\\=\end{smallmatrix})\frac{(Q|P):\phi}{(P|Q):\%\phi}$$

$$rely(\begin{smallmatrix}\geq\\\leqq\\>\end{smallmatrix})\frac{\forall Q(Q:\phi\Rightarrow(P|Q):\phi')}{P:[\phi]\phi'}$$

$$pick(\begin{smallmatrix}\geq\\\leqq\\>\end{smallmatrix})\frac{\exists Q(Q:\phi wedge(P|Q):\phi')}{P:\langle\phi\rangle\phi'}$$

Pattern Satisfaction Properties $(P|Q):(\phi||ff)$ iff $P:\phi$ iff $(P|Q):(\phi|tt)$.

$(P|Q):\neg(\phi||ff)$ iff $P:\neg\phi$ iff $(P|Q):(\neg\phi|tt)$.

If $P\stackrel{\alpha}{=}Q$ then $P:Q$.

This may be proved by induction on the structure of P (or Q); as well as on the derivation of $P\stackrel{\alpha}{=}Q$.

If $P:Q$ then $P\stackrel{\alpha}{=}Q$.

This may be proved by induction on the derivation of $P:Q$.

$P:Q$ iff $P\stackrel{\alpha}{=}Q$.

$P:P$ for every process P.

With modulo alpha conversion rules we have:

1. $P : \langle Q \rangle \phi$ iff $(P|Q') : \phi$ for some $Q' \stackrel{\alpha}{=} Q$.
2. $P : [Q]\phi$ iff $(P|Q') : \phi$ for every $Q' \stackrel{\alpha}{=} Q$.

Without modulo alpha conversion rules we have:
$P : \langle Q \rangle \phi$ iff $(P|Q) : \phi$ iff $P : [Q]\phi$.

2.5.6 Nominal

Name and Action (In)Equality Binary Atoms

$$()\frac{}{P : (a = a')} a = a' \qquad ()\frac{}{P : (a \neq a')} a \neq a'$$

$$()\frac{}{P : (\alpha = \alpha')} \alpha = \alpha' \qquad ()\frac{}{P : (\alpha \neq \alpha')} \alpha \neq \alpha'$$

Free Name Unary Atom

$$()\frac{}{\bar{a}a' : ©a''} a = a'' \vee a'' = a'$$

$$()\frac{}{!a(x).P : ©a}$$

$$()\frac{}{a(x).P : ©a}$$

$$(\stackrel{=}{>})\frac{P : ©a}{!a'(x).P : ©a} a' \neq awedgea \neq x$$

$$(\stackrel{=}{>})\frac{P : ©a}{a'(x).P : ©a} a' \neq awedgea \neq x$$

$$(\stackrel{=}{>})\frac{P : ©a}{(\nu x : \sigma).P : ©a} a \neq x$$

$$(\stackrel{=}{>})\frac{S_i : ©a}{S_1 + S_2 : ©a} i \in \{1, 2\}$$

$$(\stackrel{=}{>})\frac{P_i : ©a}{(P_1|P_2) : ©a} i \in \{1, 2\}$$

Back checking a proof, side conditions like $i \in \{1, 2\}$ are given. But back proving, with these tableaux like rules we have to choose non deterministically such i.

Fresh Name Quantifier

$$(\stackrel{>}{=})\frac{P : \phi(a/x)}{P : N(x)\phi} a \notin (fn(\phi) \cup fn(P))$$

Note the fresh quantification of x over the cofinite set of names non free in both the process and formula.

Name Permutation in Formula (Meta Permutation in Process)

$$(\stackrel{\geq}{=})\frac{(aa')P : \phi}{P : (aa')\phi}$$

meta abbreviating $(aa')P \stackrel{df}{=} P(z/a)(a/a')(a'/z)$; permuting by composing substitutions, swapping via an extra z fresh wrt a, a', P; ie $a \neq z \neq a'$ and $z \notin na(P)$.

2.6 Satisfaction Soundness

If $P : \phi$ then $P \in \|\phi\|$.

This may be proved by induction on the derivation of $P : \phi$.

2.7 Satisfaction Relative (SO In)Completeness

Since the semantics and satisfaction are mostly finitary, we actually expect completeness (besides soundness) for most fragments.

If $P \in \|\phi\|$ then $P : \phi$, for most basic fragments; but SO in principle, as second-order logic in general is incomplete.

This is expected by induction on the structure of ϕ.

2.8 Satisfaction Relative Decidability

2.8.1 Propositional and FO/SO Quantificational

The propositional fragment is relative decidable, since its standard rules are finitary and formula decreasing.

But the FO and SO fragments are not expected relative decidable, as usual, due to the possibly infinitary quantifications. With just (universal) existential ones, we expect both fragments relative (co)semidecidable.

2.8.2 Nominal Satisfaction Relative Decidability

2.8.3 Nominal Satisfaction Relative Decidability

$P : (a = a')$ iff $a = a'$.

Each is either explicitly confuted by an axiom, maybe with predicate sidecondition, or otherwise implicitly refuted.

$P : (a = a')$ is decidable iff so is $a = a'$.

Every unary atom judgement $P : ©a$ is decidable.

This may be proved by induction on P. It is refuted for 0. It is conditionally confuted by the unary atom axioms. Each process decreasing rule may confute it from the induction hypothesis premiss. Otherwise it is implicitly refuted.

$P : N(x)\phi$ iff $P : \phi(a/x)$, freshing $a \notin fn(P, \phi)$.

$P : N(x)\phi$ is decidable iff $P : \phi(a/x)$ is decidable, provided freshing $a \notin fn(P, \phi)$ is computable.

$P : (aa')\phi$ iff $(aa')P : \phi$, taking $(aa')P$ as defined.

$P : (aa')\phi$ is decidable iff $(aa')P : \phi$ is decidable, assuming $(aa')P$ is definable.

Assume a language L and its nominal (fragment) extension nL.

Suppose $a = a'$ is decidable and freshing $a \notin fn(P, \phi)$ is computable and $(aa')P$ is definable. Thus $\vdash_{nL} P_0 : \phi_0$ iff $\vdash_L P_i : \phi_i$ for $i = 1, \ldots, k$, with suitable $k \geq 0$, $P_1, \ldots, P_k$ and $\phi_1, \ldots, \phi_k \in L$.

Write $P :_L \phi$ for $\vdash_L P : \phi$. Thus

$:_{nL}$ is decidable iff so is $:_L$, provided $a = a'$ is decidable and freshing $a \notin fn(P, \phi)$ is computable and $(aa')P$ is definable.

2.8.4 Pre Satisfaction Relative Decidability

$P : \langle\equiv\rangle\phi$ iff $Q : \phi$ for some $Q \equiv P$. It suffices to find one such Q, but the structural congruence class of P, besides infinite, may be complex. We may search for Q, by induction on the derivation of $Q \equiv P$, checking $Q : \phi$.

If $\equiv$ and $:_L$ are decidable then $:_{uL} \langle\equiv\rangle\phi$ is semidecidable. Indeed the latter amounts to existentially projecting (P, ϕ) out of (P, Q, ϕ), with $P \equiv Q$ and $Q :_L \phi$.

$P : [\equiv]\phi$ iff $Q : \phi$ whenever $Q \equiv P$. Both amount to $(Q \equiv P) \Rightarrow (Q : \phi)$ for every Q. Ie showing $Q : \phi$ by induction on the derivation of structural congruence $Q \equiv P$.

If $\equiv$ and $:_L$ are decidable then $:_{uL} [\equiv]\phi$ is cosemidecidable.

2.8.5 Pattern Satisfaction Relative Decidability

Since all pattern satisfaction rules are formula decreasing, back proving with them is well founded.

But in the last two, for rely and pick, note the increased $P|Q$ where the Q may be arbitrarily large and it is quantified over ϕ whose denotation may be infinite and complex.

If such quantifications were finitary, then back proving would be decidable. However in general such quantifications are infinitary. Thus with the

rely dual (existential) rule, satisfaction should be semidecidable; and with the rely (universal) one, is expected cosemidecidable.

Chapter 3

Dynamic Fragments

3.1 Syntax

$$
\begin{array}{rcl}
(a)\ \phi & ::= & [\alpha]\phi \mid \langle\alpha\rangle\phi \mid (\alpha).\phi \\
(k)\ \phi & ::= & [K]\phi \mid \langle K\rangle\phi \mid (K).\phi \\
\text{menu } K & ::= & l \mid -l \\
\text{list } l & ::= & \epsilon \mid in \mid out \mid k \mid l,k \\
\text{mode } k & ::= & \alpha \mid p \mid pa \\
(c)\ \phi & ::= & AX_K\phi \mid A(\phi U_K \phi) \mid AF_K\phi \mid AG_K\phi \mid \\
 & & EX_K\phi \mid E(\phi U_K \phi) \mid EF_K\phi \mid EG_K\phi \mid \\
 & & A(\phi R_K \phi) \mid E(\phi R_K \phi)
\end{array}
$$

3.1.1 Action Modal

3.1.2 Action Set (Based Multi-) Modal

Co(mplement) Menu $\boxed{-K}$ $-(-l) \stackrel{df}{=} l$ and $-(l) \stackrel{df}{=} -l$.

3.1.3 Action Set Branching Future Temporal Logic

Future (Operators) KCTL (Branching Tree) Subsumes CTL; eg $EX\phi = EX_{-}\phi$ and $AX\phi = AX_{-}\phi$.

3.2 Semantics

3.2.1 Action Modal

$$
\begin{array}{rcl}
\|\langle\alpha\rangle\phi\| & \stackrel{df}{=} & \{P \mid \exists(Q \in (P \cdot \alpha))(Q : \phi)\} \\
\|[\alpha]\phi\| & \stackrel{df}{=} & \{P \mid \forall(Q \in (P \cdot \alpha))(Q : \phi)\}
\end{array}
$$

$$\|(\alpha).\phi\| \stackrel{df}{=} \{P \mid A(P) = \{\alpha\} wedge \exists (Q \in (P \cdot \alpha))(Q : \phi)\}.$$

3.2.2 Action Set (Based Multi-) Modal

Let $Act = \{\tau\} \cup Obs$ and $Obs = In \cup Out$ where

$$\begin{aligned} In &\stackrel{df}{=} \{\alpha \mid \exists a \exists x (a(x) = \alpha)\} \\ Out &\stackrel{df}{=} \{\alpha \mid \exists \rho \exists a \exists a' (\rho \bar{a} a' = \alpha)\} \\ In_a &\stackrel{df}{=} \{\alpha \mid \exists x (a(x) = \alpha)\} \\ Out_{\bar{a}} &\stackrel{df}{=} \{\alpha \mid \exists \rho \exists a' (\rho \bar{a} a' = \alpha)\} \\ Out_{\bar{a}a'} &\stackrel{df}{=} \{\alpha \mid \exists \rho (\rho \bar{a} a' = \alpha)\} \subseteq Out_{\bar{a}}. \end{aligned}$$

Menu Semantics $\boxed{\|K\|}$

$$\begin{aligned} \|\epsilon\| &\stackrel{df}{=} \emptyset \\ \|in\| &\stackrel{df}{=} In \\ \|out\| &\stackrel{df}{=} Out \\ \|\alpha\| &\stackrel{df}{=} \{\alpha\} \\ \|ax\| &\stackrel{df}{=} \{a(x)\} \\ \|\bar{a}a'\| &\stackrel{df}{=} Out_{\bar{a}a'} \\ \|a\| &\stackrel{df}{=} In_a \\ \|\bar{a}\| &\stackrel{df}{=} Out_{\bar{a}} \\ \|l, k\| &\stackrel{df}{=} \|l\| \cup \|k\| \\ \|-l\| &\stackrel{df}{=} Act \setminus \|l\|. \end{aligned}$$

Finite Submenu $\boxed{\|K\|_Q}$

$$\|K\|_Q \stackrel{df}{=} \|K\| \cap A(Q).$$

Formula Semantics

$$\begin{aligned} \|\langle K \rangle \phi\| &\stackrel{df}{=} \{P \mid \exists (Q \in (P \cdot \|K\|_P))(Q : \phi)\} \\ &= \{P \mid \exists (\alpha \in \|K\|_P) \exists (Q \in (P \cdot \alpha))(Q : \phi)\} \\ \|[K]\phi\| &\stackrel{df}{=} \{P \mid \forall (Q \in (P \cdot \|K\|_P))(Q : \phi)\} \\ &= \{P \mid \forall (\alpha \in \|K\|_P) \forall (Q \in (P \cdot \alpha))(Q : \phi)\} \\ \|(K).\phi\| &\stackrel{df}{=} \{P \mid \|-K\|_P = \emptyset wedge \exists (Q \in (P \cdot \|K\|_P))(Q : \phi)\} \\ &= \{P \mid \|-K\|_P = \emptyset wedge \exists (\alpha \in \|K\|_P) \exists (Q \in (P \cdot \alpha))(Q : \phi)\}. \end{aligned}$$

3.2.3 KCTL Temporal Fragment

Semantical Modal Operators over a process set T.

$$\begin{aligned}
[K]T &\stackrel{df}{=} \{P \mid \forall(Q \in (P \cdot \|K\|_P))(Q \in T)\} \\
&= \{P \mid \forall(\alpha \in \|K\|_P)\forall(Q \in (P \cdot \alpha))(Q \in T)\} \\
\langle K\rangle T &\stackrel{df}{=} \{P \mid \exists(Q \in (P \cdot \|K\|_P))(Q \in T)\} \\
&= \{P \mid \exists(\alpha \in \|K\|_P)\exists(Q \in (P \cdot \alpha))(Q \in T)\}.
\end{aligned}$$

Modal Connectives

$$\begin{aligned}
\|AX_K\phi\| &\stackrel{df}{=} \|[K]\phi\| \\
\|EX_K\phi\| &\stackrel{df}{=} \|\langle K\rangle\phi\|.
\end{aligned}$$

Inductive Connectives

$$\begin{aligned}
\|A(\phi U_K\phi')\| &\stackrel{df}{=} \cap\{T \mid T \supseteq \|\phi'\| \cup (\|\phi\| \cap [K]T)\} \\
\|E(\phi U_K\phi')\| &\stackrel{df}{=} \cap\{T \mid T \supseteq \|\phi'\| \cup (\|\phi\| \cap \langle K\rangle T)\} \\
\|AF_K\phi\| &\stackrel{df}{=} \cap\{T \mid T \supseteq \|\phi\| \cup [K]T)\} \\
\|EF_K\phi\| &\stackrel{df}{=} \cap\{T \mid T \supseteq \|\phi\| \cup \langle K\rangle T)\}.
\end{aligned}$$

Coinductive Connectives

$$\begin{aligned}
\|AG_K\phi\| &\stackrel{df}{=} \cup\{T \mid T \subseteq \|\phi\| \cap [K]T)\} \\
\|EG_K\phi\| &\stackrel{df}{=} \cup\{T \mid T \subseteq \|\phi\| \cap \langle K\rangle T)\}.
\end{aligned}$$

3.3 Duality

3.3.1 Action Modal

Homo

$$\neg[\alpha]\phi \stackrel{f}{=} \langle\alpha\rangle\neg\phi$$

$$\neg\langle\alpha\rangle\phi \stackrel{f}{=} [\alpha]\neg\phi$$

Non Homo

$$\neg(\alpha).\phi \stackrel{f}{=} (\exists\alpha'(\alpha' \neq \alpha wedge\langle\alpha'\rangle tt)) \vee [\alpha]\neg\phi$$

3.3.2 Action Set (Based Multi-) Modal

Homo

$$\neg[K]\phi \stackrel{f}{=} \langle K\rangle\neg\phi$$

$$\neg\langle K\rangle\phi \stackrel{f}{=} [K]\neg\phi$$

Non Homo

$$\neg(K).\phi \stackrel{f}{=} (\langle -K\rangle tt \vee [K]\neg\phi)$$

3.3.3 KCTL

$$\neg AX_K\phi \stackrel{f}{=} EX_K\neg\phi$$

$$\neg EX_K\phi \stackrel{f}{=} AX_K\neg\phi$$

$$\neg AF_K\phi \stackrel{f}{=} EG_K\neg\phi$$

$$\neg EG_K\phi \stackrel{f}{=} AF_K\neg\phi$$

$$\neg AG_K\phi \stackrel{f}{=} EF_K\neg\phi$$

$$\neg EF_K\phi \stackrel{f}{=} AG_K\neg\phi$$

$$\neg A(\phi U_K\phi') \stackrel{f}{=} E(\neg\phi R_K\neg\phi')$$

$$\neg E(\phi R_K\phi') \stackrel{f}{=} A(\neg\phi U_K\neg\phi')$$

$$\neg A(\phi R_K\phi') \stackrel{f}{=} E(\neg\phi U_K\neg\phi')$$

$$\neg E(\phi U_K\phi') \stackrel{f}{=} A(\neg\phi R_K\neg\phi')$$

3.4 Formula Equivalences

3.4.1 Action Modal

$$[\alpha]tt \stackrel{f}{=} tt$$

$$\langle\alpha\rangle ff \stackrel{f}{=} ff$$

$$(\alpha).\phi \stackrel{f}{=} (\forall\alpha'(\alpha' \neq \alpha \Rightarrow [\alpha']ff)))wedge\langle\alpha\rangle\phi$$

3.4.2 Action Set (Based Multi-) Modal

$$[\epsilon]\phi \stackrel{f}{=} tt$$

$$(\epsilon).\phi \stackrel{f}{=} \langle\epsilon\rangle\phi \stackrel{f}{=} ff$$

$$(K).\phi \stackrel{f}{=} (([-K]ff)wedge\langle K\rangle\phi) \stackrel{f}{=} ((\neg\langle -K\rangle tt)wedge\langle K\rangle\phi)$$

3.4.3 KCTL

$$\boxed{A(\phi R_K \phi') \stackrel{f}{=} \neg E(\neg\phi U_K \neg\phi')}$$

$$\boxed{E(\phi R_K \phi') \stackrel{f}{=} \neg A(\neg\phi U_K \neg\phi')}$$

$$AX_K\phi \stackrel{f}{=} [K]\phi$$

$$EX_K\phi \stackrel{f}{=} \langle K\rangle\phi$$

$$A(\phi U_K \phi') \stackrel{f}{=} (\mu(Z)\phi' \vee (\phi wedge(\neg\phi')wedge[K]Z))$$

$$E(\phi U_K \phi') \stackrel{f}{=} (\mu(Z)\phi' \vee (\phi wedge(\neg\phi')wedge\langle K\rangle Z))$$

$$AF_K\phi \stackrel{f}{=} \mu(Z)(\phi \vee [K]Z) \stackrel{f}{=} A(tt\ U_K\ \phi)$$

$$EF_K\phi \stackrel{f}{=} \mu(Z)(\phi \vee \langle K\rangle Z) \stackrel{f}{=} E(tt\ U_K\ \phi)$$

$$AG_K\phi \stackrel{f}{=} \nu(Z)(\phi wedge[K]Z) \stackrel{f}{=} \neg EF_K\neg\phi \stackrel{f}{=} \neg E(tt\ U_K\ \neg\phi)$$

$$AG_K\phi \stackrel{f}{=} (\exists(Z)valid(Z \Rightarrow [K]Z)wedgevalid(Z \Rightarrow \phi)wedgeZ)$$

$$EG_K\phi \stackrel{f}{=} \nu(Z)(\phi wedge\langle K\rangle Z) \stackrel{f}{=} \neg AF_K\neg\phi \stackrel{f}{=} \neg A(tt\ U_K\ \neg\phi)$$

3.5 Satisfaction

3.5.1 Action Modal

3.5.1.1 Axiomatic Semantics

May $\boxed{\langle\alpha\rangle\phi}$

$$(\substack{\geq\\ \leq\\ >})\frac{\exists(Q \in P \cdot \alpha)(Q : \phi)}{P : \langle\alpha\rangle\phi}$$

Must $\boxed{[\alpha]\phi}$

$$(\substack{\geq\\ \leq\\ >})\frac{\forall(Q \in P \cdot \alpha)(Q : \phi)}{P : [\alpha]\phi}$$

Next $\boxed{(\alpha).\phi}$

$$(\begin{smallmatrix}\geq\\ \leq\\ >\end{smallmatrix})\frac{\exists(Q \in P \cdot \alpha)(Q:\phi)}{P:(\alpha).\phi}A(P)=\{\alpha\}$$

3.5.1.2 LTS as Action Possibility Pattern Satisfaction

We express the late LTS transitions as certain action possibility pattern formulas and axiomatize them inductively. The action modality is followed by a special process pattern rather than arbitrary an formula.

$$msg()\frac{}{\bar{a}a' : \langle \bar{a}a' \rangle 0}$$

$$inp()\frac{}{a(x).P : \langle a(x) \rangle P}$$

$$rep(\begin{smallmatrix}>\\ <\end{smallmatrix})\frac{}{!a(x).P : \langle a(x) \rangle (P|!a(x).P)}$$

$$sum_{i\in\{1,2\}}(\begin{smallmatrix}=\\ >\end{smallmatrix})\frac{S_i : \langle a(x) \rangle Q}{S_1 + S_2 : \langle a(x) \rangle Q}$$

$$open(\begin{smallmatrix}>\\ >\end{smallmatrix})\frac{P : \langle \bar{a}x \rangle Q}{(\nu x:\sigma)P : \langle (\nu x:\sigma)\bar{a}x \rangle Q}x \neq a$$

$$clom_{i\in\{1,2\}}(\begin{smallmatrix}\geq\\ \leq\\ >\end{smallmatrix})\frac{P_{3-i} : \langle (\nu y:\sigma)\bar{a}y \rangle P'_{3-i} \quad P_i : \langle a(x) \rangle P'_i}{P_1|P_2 : \langle \tau \rangle (\nu y:\sigma)(P'_i(y/x)|_i P'_{3-i})}y \notin fn(P_i)$$

$$com_{i\in\{1,2\}}(\begin{smallmatrix}\geq\\ \leq\\ >\end{smallmatrix})\frac{P_{3-i} : \langle \bar{a}a' \rangle P'_{3-i} \quad P_i : \langle a(x) \rangle P'_i}{P_1|P_2 : \langle \tau \rangle (P'_i(a'/x)|_i P'_{3-i})}$$

$$par_{i\in\{1,2\}}(\begin{smallmatrix}\geq\\ \leq\\ >\end{smallmatrix})\frac{P_i : \langle \alpha \rangle P'_i}{P_1|P_2 : \langle \alpha \rangle (P'_i|_i P_{3-i})}bn(\alpha)\#fn(P_{3-i})$$

$$res(\begin{smallmatrix}\geq\\ \leq\\ >\end{smallmatrix})\frac{P : \langle \alpha \rangle P'}{(\nu x:\sigma)P : \langle \alpha \rangle (\nu x:\sigma)P'}x \notin na(\alpha)$$

All rules are finitary (at most binary) and are formula decreasing, apart from sum_i, which is formula invariant. But sum_i loops process decreasing, thus finitely till we could only apply one of the other rules.

If $P \xrightarrow{\alpha} Q$ then $P : \langle \alpha \rangle Q$ is derivable by these rules.

This holds by induction on the derivation of $P \xrightarrow{\alpha} Q$, whose rules are expressed modally by these ones.

If $P : \langle \alpha \rangle Q$ by these rules then $P \xrightarrow{\alpha} Q$.

This holds by induction on the derivation of $P : \langle \alpha \rangle Q$.

$P : \langle \alpha \rangle Q$ iff $P \xrightarrow{\alpha} Q$.

So the calculus operational semantics is thus expressible.

3.5.1.3 Action Possibility Satisfaction

Hereafter we axiomatise with generality; here for action possibility modality followed by an arbitrary formula.

Both communication (silent τ action) variants, with or without output scope extrusion, are unified in one rule.

$$(\overset{>}{>})\frac{0 : \phi}{\bar{a}a' : \langle \bar{a}a' \rangle \phi}$$

$$(\overset{>}{<})\frac{(P | !a(x).P) : \phi}{!a(x).P : \langle a(x) \rangle \phi}$$

$$(\overset{>}{>})\frac{P : \phi}{a(x).P : \langle a(x) \rangle \phi}$$

$$(\overset{=}{>})\frac{\vee_{i \in \{1,2\}}(S_i : \langle a(x) \rangle \phi)}{S_1 + S_2 : \langle a(x) \rangle \phi}$$

$$(\overset{>}{>})\frac{P : \langle \bar{a}x \rangle \phi}{(\nu x : \sigma)P : \langle (\nu x : \sigma)\bar{a}x \rangle \phi} x \neq a$$

$$(\overset{>}{>})\frac{\begin{array}{l}\vee_{i \in \{1,2\}}(\exists (a(x) \in A(P_i))\exists (\rho \bar{a}a' \in A(P_{3-i})) \\ \exists (Q \in (P_i \cdot a(x)))\exists (R \in (P_{3-i} \cdot \rho \bar{a}a')) \\ (bn(\rho) \# fn(P_i) wedge(\rho(Q(a'/x)|_i R) : \phi)))\end{array}}{P_1 | P_2 : \langle \tau \rangle \phi}$$

$$(\overset{>}{\leqslant})\frac{\begin{array}{l}\vee_{i \in \{1,2\}}(\exists (Q \in (P_i \cdot \alpha)) \\ (bn(\alpha) \# fn(P_{3-i}) wedge(Q|_i P_{3-i}) : \phi))\end{array}}{P_1 | P_2 : \langle \alpha \rangle \phi}$$

$$(\overset{>}{\leqslant})\frac{\exists (Q \in (P \cdot \alpha))((\nu x : \sigma)Q : \phi)}{(\nu x : \sigma)P : \langle \alpha \rangle \phi} x \notin na(\alpha)$$

All rules are finitary and are formula decreasing, except sum_i, which is formula invariant. But sum_i loops process decreasing, thus finitely till we could only apply one of the other rules.

$P : \langle \alpha \rangle \phi$ iff $P \xrightarrow{\alpha} Q$ and $Q : \phi$ for some Q.

Each direction also holds by induction on the derivation of its antecedent judgement.

Since $Q : Q'$ iff $Q \overset{\alpha}{=} Q'$ then with $\phi = Q'$ follows the next
$P : \langle \alpha \rangle Q'$ iff $P \xrightarrow{\alpha} Q$ and $Q \overset{\alpha}{=} Q'$.

So with these general rules the LTS is up to alpha conversion expressible by action possibility process pattern formulas.

3.5.1.4 Action Necessity Satisfaction

$$()\frac{}{0 : [\alpha]\phi}$$

$$()\frac{}{(\nu x : \sigma).P : [\alpha]\phi} x \in na(\alpha)$$

$$(\begin{smallmatrix}>\\>\end{smallmatrix})\frac{0 : \phi}{\bar{a}a' : [\bar{a}a']\phi}$$

$$(\begin{smallmatrix}>\\<\end{smallmatrix})\frac{(P|!a(x).P) : \phi}{!a(x).P : [a(x)]\phi}$$

$$(\begin{smallmatrix}>\\>\end{smallmatrix})\frac{P : \phi}{a(x).P : [a(x)]\phi}$$

$$(\begin{smallmatrix}=\\>\end{smallmatrix})\frac{wedge_{i\in\{1,2\}}(S_i : [\alpha]\phi)}{S_1 + S_2 : [\alpha]\phi}$$

$$(\begin{smallmatrix}>\\>\end{smallmatrix})\frac{P : [\bar{a}x]\phi}{(\nu x : \sigma)P : [(\nu x : \sigma)\bar{a}x]\phi} x \neq a$$

$$(\begin{smallmatrix}>\\\leqslant\\>\end{smallmatrix})\frac{\begin{array}{l} wedge_{i\in\{1,2\}}(\forall(a(x) \in A(P_i))\forall(\rho\bar{a}a' \in A(P_{3-i})) \\ \forall(Q \in (P_i \cdot a(x)))\forall(R \in (P_{3-i} \cdot \rho\bar{a}a')) \\ (bn(\rho)\#fn(P_i)wedge(\rho(Q(a'/x)|_i R) : \phi))) \end{array}}{P_1|P_2 : [\tau]\phi}$$

$$(\begin{smallmatrix}>\\\leqslant\\>\end{smallmatrix})\frac{\begin{array}{l} wedge_{i\in\{1,2\}}(\forall(Q \in (P_i \cdot \alpha)) \\ (bn(\alpha)\#fn(P_{3-i})wedge(Q|_i P_{3-i}) : \phi)) \end{array}}{P_1|P_2 : [\alpha]\phi}$$

$$(\begin{smallmatrix}>\\\leqslant\\>\end{smallmatrix})\frac{\forall(Q \in (P \cdot \alpha))((\nu x : \sigma)Q : \phi)}{(\nu x : \sigma)P : [\alpha]\phi} x \notin na(\alpha)$$

All rules are finitary and are formula decreasing, except sum_i, which is formula invariant. But sum_i loops process decreasing, thus finitely till we could only apply one of the other rules.

3.5.1.5 Action Exclusivity Satisfaction

$$(\begin{smallmatrix}>\\>\end{smallmatrix})\frac{0 : \phi}{\bar{a}a' : (\bar{a}a').\phi}$$

$$(\begin{smallmatrix}>\\<\end{smallmatrix})\frac{(P|!a(x).P) : \phi}{!a(x).P : (a(x)).\phi}$$

$$(\begin{smallmatrix}>\\>\end{smallmatrix})\frac{P : \phi}{a(x).P : (a(x)).\phi}$$

$$(\begin{smallmatrix}=\\>\end{smallmatrix})\frac{\vee_{i\in\{1,2\}}(A(S_{3-i}=\{a(x)\}wedgeS_i:(a(x)).\phi)}{S_1+S_2:(a(x)).\phi}$$

$$(\begin{smallmatrix}>\\>\end{smallmatrix})\frac{A((\nu x:\sigma)P)=\{(\nu x:\sigma)\bar{a}x\}\quad P:(\bar{a}x).\phi}{(\nu x:\sigma)P:((\nu x:\sigma)\bar{a}x).\phi}x\neq a$$

$$(\begin{smallmatrix}>\\>\end{smallmatrix})\frac{\begin{array}{l}\vee_{i\in\{1,2\}}(\exists(a(x)\in A(P_i))\exists(\rho\bar{a}a'\in A(P_{3-i}))\\ \exists(Q\in(P_i\cdot a(x)))\exists(R\in(P_{3-i}\cdot\rho\bar{a}a'))\\ (bn(\rho)\#fn(P_i)wedge(\rho(Q(a'/x)|_iR):\phi)))\end{array}}{P_1|P_2:(\tau).\phi}A(P_1|P_2)=\{\tau\}$$

$$(\begin{smallmatrix}>\\\leqslant\\>\end{smallmatrix})\frac{\begin{array}{l}\vee_{i\in\{1,2\}}(\exists(Q\in(P_i\cdot\alpha))\\ (bn(\alpha)\#fn(P_{3-i})wedge(Q|_iP_{3-i}):\phi))\end{array}}{P_1|P_2:(\alpha).\phi}A(P_1|P_2)=\{\alpha\}$$

$$(\begin{smallmatrix}>\\\leqslant\\>\end{smallmatrix})\frac{\exists(Q\in(P\cdot\alpha))((\nu x:\sigma)Q:\phi)}{(\nu x:\sigma)P:(\alpha).\phi}x\notin na(\alpha)wedgeA((\nu x:\sigma)P)=\{\alpha\}$$

All rules are finitary and are formula decreasing, except sum_i, which is formula invariant. But sum_i loops process decreasing, thus finitely till we could only apply one of the other rules.

$P:(\alpha).\phi$ iff $P:\langle\alpha\rangle\phi$ and $A(P)=\{\alpha\}$.

Each direction may be proved by induction on the derivation of the respective antecedent.

3.5.2 Action Set (Based Multi-) Modal

3.5.2.1 Axiomatic Semantics

$\boxed{\langle K\rangle\phi}$ for some K may ϕ

$$(\begin{smallmatrix}>\\\leqslant\\>\end{smallmatrix})\frac{\exists(\alpha\in\|K\|_P)\exists(Q\in(P\cdot\alpha))(Q:\phi)}{P:\langle K\rangle\phi}$$

$\boxed{[K]\phi}$ for all K must ϕ

$$(\begin{smallmatrix}>\\\leqslant\\>\end{smallmatrix})\frac{\forall(\alpha\in\|K\|_P)\forall(Q\in(P\cdot\alpha))(Q:\phi)}{P:[K]\phi}$$

$\boxed{(K).\phi}$ for some K next ϕ

$$(\begin{smallmatrix}>\\\leqslant\\>\end{smallmatrix})\frac{\exists(\alpha\in\|K\|_P)\exists(Q\in(P\cdot\alpha))(Q:\phi)}{P:(K).\phi}\|-K\|_P=\emptyset$$

These rules are formula decreasing.

As $\|K\|_P$ and $\|-K\|_P$ are finite and the LTS is image finite, then so is $P\cdot\|K\|_P$. So these rules are finitary.

3.5.3 KCTL

Modal Connectives

$\boxed{AX_K\phi}$

$$(\substack{=\\=})\frac{P : [K]\phi}{P : AX_K\phi}$$

$\boxed{EX_K\phi}$

$$(\substack{=\\=})\frac{P : \langle K\rangle\phi}{P : EX_K\phi}$$

Inductive Connectives

$\boxed{A(\phi U_K\phi')}$ is the inductive set generated by the next rules

$$(\substack{\geq\\=})\frac{P : \phi'}{P : A(\phi U_K\phi')}$$

$$(\substack{\geq\\\leq\leq})\frac{P : \phi wedge\neg\phi' \quad \forall(Q \in (P \cdot \|K\|_P))(Q : A(\phi U_K\phi'))}{P : A(\phi U_K\phi')}$$

$\boxed{E(\phi U_K\phi')}$ is the inductive set generated by the next rules

$$(\substack{\geq\\=})\frac{P : \phi'}{P : E(\phi U_K\phi')}$$

$$(\substack{\geq\\\leq\leq})\frac{P : \phi wedge\neg\phi' \quad \exists(Q \in (P \cdot \|K\|_P))(Q : E(\phi U_K\phi'))}{P : E(\phi U_K\phi')}$$

$\boxed{AF_K\phi}$ is the inductive set generated by the next rules

$$(\substack{\geq\\=})\frac{P : \phi}{P : AF_K\phi}$$

$$(\substack{\geq\\\leq\leq})\frac{P : \neg\phi \quad \forall(Q \in (P \cdot \|K\|_P))(Q : AF_K\phi)}{P : AF_K\phi}$$

$\boxed{EF_K\phi}$ is the inductive set generated by

$$(\substack{\geq\\=})\frac{P : \phi}{P : EF_K\phi}$$

$$(\substack{\geq\\\leq\leq})\frac{P : \neg\phi \quad \exists(Q \in (P \cdot \|K\|_P))(Q : EF_K\phi)}{P : EF_K\phi}$$

Coinductive Connectives

$\boxed{AG_K\phi}$ is the coinductive set consistent with the next rule

$$(\substack{\geq\\\leq\\>})\frac{P:\phi \quad \forall(Q \in (P \cdot \|K\|_P))(Q : AG_K\phi)}{P : AG_K\phi}$$

ie the complement of the inductive set generated by

$$(\substack{>\\=})\frac{P:\neg\phi}{P : EF_K\neg\phi}$$

$$(\substack{\geq\\\leq\\>})\frac{P:\phi \quad \exists(Q \in (P \cdot \|K\|_P))(Q : EF_K\neg\phi)}{P : EF_K\neg\phi}$$

$\boxed{EG_K\phi}$ is the coinductive set consistent with the next rule

$$(\substack{\geq\\\leq\\>})\frac{P:\phi \quad \exists(Q \in (P \cdot \|K\|_P))(Q : EG_K\phi)}{P : EG_K\phi}$$

ie the complement of the inductive set generated by

$$(\substack{>\\=})\frac{P:\neg\phi}{P : AF_K\neg\phi}$$

$$(\substack{\geq\\\leq\\>})\frac{P:\phi \quad \forall(Q \in (P \cdot \|K\|_P))(Q : AF_K\neg\phi)}{P : AF_K\neg\phi}$$

Note that $EG_K\phi$ requires a labelled infinite path (proper run) in the LTS.

3.6 Satisfaction Soundness

3.6.1 Action Modal

[Soundness]

1. If $P : \langle\alpha\rangle\phi$ then $P \in \|\langle\alpha\rangle\phi\|$.
2. If $P : [\alpha]\phi$ then $P \in \|[\alpha]\phi\|$.
3. If $P : (\alpha).\phi$ then $P \in \|(\alpha).\phi\|$.

Each is provable, relatively to other fragments, by induction on the derivation of the antecedent.

3.6.2 Action Set (Based Multi) Modal

[Correspondence]

1. $P : \langle K \rangle \phi$ iff $P \in \|\langle K \rangle \phi\|$.
2. $P : [K]\phi$ iff $P \in \|[K]\phi\|$.
3. $P : (K).\phi$ iff $P \in \|(K).\phi\|$.

Each multi (ie menu) modality formula is sound (and complete), resp. relative to the other fragments, since the axiomatic semantics paraphrases the (denotational) semantics.

3.6.3 KCTL

[Modal Correspondence]

1. $P : EX_K\phi$ iff $P \in \|EX_K\phi\|$.
2. $P : AX_K\phi$ iff $P \in \|AX_K\phi\|$.

[Inductive Correspondence]

1. $P : A(\phi U_K \phi')$ iff $P \in \|A(\phi U_K \phi')\|$.
2. $P : E(\phi U_K \phi')$ iff $P \in \|E(\phi U_K \phi')\|$.
3. $P : AF_K\phi'$ iff $P \in \|AF_K\phi'\|$.
4. $P : EF_K\phi'$ iff $P \in \|EF_K\phi'\|$.

Each inductive formula satisfaction rule is sound wrt the semantics, by the way each rule expresses the respective semantics.

The fixpoint minimality amounts to inductive generation of membership in the process set meaning of the formula. Thus back proving a true membership must be well founded. As the LTS is image finite and finitely branching, each rule is likewise finitary and so the back derivation is finite. Therefore the inductive formula rules are also relative complete.

1. If $P : AG_K\phi'$ then $P \in \|AG_K\phi'\|$.
2. If $P : EG_K\phi'$ then $P \in \|EG_K\phi'\|$.

Each coinductive formula satisfaction rule is sound wrt the semantics, by the way each rule expresses the respective semantics.

Meta checking membership in the process set of the formula, whose fixpoint is maximally consistent with the respective rules, could be non well founded, thus with infinite derivation. As their rule completeness would require infinite derivations, but we consider just finite derivations, coinductive formula satisfaction is incomplete.

3.7 Satisfaction Relative Decidability

3.7.1 Action Modal

Possibility %beginpn $P : \langle\alpha\rangle\phi$ back proving tree is well founded.

$P : \langle\alpha\rangle\phi$ back proving tree is finitely branching.

$P : \langle\alpha\rangle\phi$ back proving tree is finite.

$P : \langle\alpha\rangle\phi$ is relative decidable.

Necessity $P : [\alpha]\phi$ back proving tree is well founded.

$P : [\alpha]\phi$ back proving tree is finitely branching.

$P : [\alpha]\phi$ back proving tree is finite.

$P : [\alpha]\phi$ is relative decidable.

Exclusivity $P : (\alpha).\phi$ back proving tree is well founded.

$P : (\alpha).\phi$ back proving tree is finitely branching.

$P : (\alpha).\phi$ back proving tree is finite.

$P : (\alpha).\phi$ is relative decidable.

3.7.2 Action Set (Based Multi) Modal

Possibility $P : \langle K\rangle\phi$ back proving tree is well founded.

$P : \langle K\rangle\phi$ back proving tree is finitely branching.

$P : \langle K\rangle\phi$ back proving tree is finite.

$P : \langle K\rangle\phi$ is relative decidable.

Necessity $P : [K]\phi$ back proving tree is well founded.

$P : [K]\phi$ back proving tree is finitely branching.

$P : [K]\phi$ back proving tree is finite.

$P : [K]\phi$ is relative decidable.

Exclusivity $P : (K).\phi$ back proving tree is well founded.

$P : (K).\phi$ back proving tree is finitely branching.

$P : (K).\phi$ back proving tree is finite.

$P : (K).\phi$ is relative decidable.

3.7.3 KCTL

$P : AX_K\phi$ and $P : EX_K\phi$ are relative decidable.

$P : E(\phi U_K \phi')$ is relative semidecidable.

$P : EF_K\phi')$ is relative semidecidable.

$P : A(\phi U_K \phi')$ is relative semidecidable.

$P : AF_K\phi')$ is relative semidecidable.

$P : AG_K\phi')$ is relative cosemidecidable.

$P : EG_K\phi')$ is relative cosemidecidable.

For finite processes, namely without replication, even the satisfaction of a KCTL formula is decidable.

Chapter 4

Derived Fragments

4.1 Syntax

$$
\begin{array}{rrl}
(x)\ \phi & ::= & p^m \mid p^t \\
m & ::= & \ldots \mid l \mid r \\
t & ::= & l' \mid r' \mid 0' \mid 1' \mid w' \mid *' \\
(b)\ \phi & ::= & \beta \\
\beta & ::= & \surd \mid \top \mid \text{stable} \mid s_{IA} \mid s_A \mid p_R \\
s & ::= & \bar{a} \mid r \\
r & ::= & a \mid (r + r) \\
(d)\ \phi & ::= & (\phi \triangleleft \phi) \mid (\phi \blacktriangleleft \phi) \\
(g)\ \Gamma & ::= & (\Sigma; \nabla; \phi)
\end{array}
$$

4.1.1 (Replicated) Linear Extended Multiplicity Semilattice

$\boxed{m}$ Extra Order

$$l < 1$$

$$r < w$$

4.1.2 (Temporally Given) Dynamic Multiplicity Semilattice

$\boxed{t}$ Order

$$l', 0' < 1'$$

$$r', 0' < w'$$

$$1', w' < *'$$

4.1.2.1 Intuitive Traces

A (port, dynamic multiplicity) trace $\Theta_{p,t}$ is a set of maximal label sequences θ, running from a process in the late LTS. Let $\Theta^\infty \stackrel{df}{=} \Theta^* \cup \Theta^\omega$.

$$\begin{aligned}
\Theta_{p,*'} &= (p \cup -p)^\infty = (-)^\infty \\
\Theta_{p,0'} &= (-p)^\infty \\
\Theta_{p,l'} &= (-p)^* p (-p)^\infty \\
\Theta_{p,1'} &= \Theta_{p,0'} \cup \Theta_{p,l'} \\
\Theta_{p,w'} &= ((-p)^* p)^\infty (-p)^\infty \\
\Theta_{p,r'} &= ((-p)^* p)\Theta_{p,w'}.
\end{aligned}$$

4.1.3 Behavioral Properties of a Process

Quietness The process cannot reduce; ie has no immediate silent actions.

Stability The process is quiet but can do, ie has immediate, non silent actions.

Strong Termination Every reduction cannot perpetuate; ie it must reach a quiet process.

Immediate Port (Sum) Activeness The process can immediately do an action of the given output port or input port sum, even after doing any number of other actions.

Port (Sum) Activeness The process may only reduce, always eventually reaching a process immediately active on the given output port or input port sum.

Port (Sum) Responsiveness Whenever the process (sends) receives an name on a given port, it will eventually (receive) send on that name, in respective turn.

4.1.4 Dependency / Interface

This fragment allows to express dependencies, of properties of a process, on those of other processes. In particular, we consider dependencies between the local and remote sides of the process interface.

4.1.5 Process Type

This highest level type allows to specify processes with a given binary sorting, as well as properties expressed by a formula using the fragments.

4.2 Derived Formulas

4.2.1 Extended Static Multiplicity Semantics

$$\begin{aligned} \|p^l\| &\stackrel{df}{=} \|p^{l'}\| \\ \|p^r\| &\stackrel{df}{=} \|p^{r'}\|. \end{aligned}$$

4.2.2 Dynamic Multiplicity Temporal Formulas

$\boxed{p^t}$ is a temporal formula from the next ones, characterising the dynamic extended multiplicity p^t.

$$\begin{aligned} p^{*'} &\stackrel{df}{=} tt \\ p^{0'} &\stackrel{df}{=} AG_{-p}[p]ff \\ p^{1'} &\stackrel{df}{=} AG_{-p}[p]p^{0'} \\ p^{l'} &\stackrel{df}{=} A(([p]p^{0'})U_{-p}(([p]p^{0'})wedge(p).tt)) \\ p^{w'} &\stackrel{df}{=} AG_{-}[p]A(([p]ff)U_{-}(\langle p\rangle tt)) \\ p^{r'} &\stackrel{df}{=} A(([p]ff)U_{-}(((p).tt)wedgep^{w'})). \end{aligned}$$

Input Extended Multiplicity

$$\begin{aligned} a^{*'} &= tt \\ a^{0'} &= AG_{-a}\forall x[a(x)]ff \\ a^{1'} &= AG_{-a}\forall x[a(x)]a^{0'} \\ a^{l'} &= A((\forall x[a(x)]a^{0'}) \\ &\quad U_{-a}((\forall x[a(x)]a^{0'})wedge(\exists x\langle a(x)\rangle tt)wedge \\ &\quad\quad \forall\alpha((\forall x(a(x)\neq\alpha)\Rightarrow[\alpha]ff))) \\ a^{w'} &= AG_{-}\forall x[a(x)] \\ &\quad A((\forall y[a(y)]ff) \\ &\quad U_{-}(\exists y\langle a(y)\rangle tt)) \\ a^{,r'} &= A((\forall x[a(x)]ff) \\ &\quad U_{-}((\forall\alpha((\forall x(a(x)\neq\alpha))\Rightarrow[\alpha]ff))wedge \\ &\quad\quad (\exists x\langle a(x)\rangle tt)wedgea^{w'})). \end{aligned}$$

Output Extended Multiplicity

$$\begin{aligned} \bar{a}^{*'} &= tt \\ \bar{a}^{0'} &= AG_{-a}\forall\rho\forall a'[\rho\bar{a}a']ff \end{aligned}$$

$$
\begin{array}{rcl}
\bar{a}^{1'} & = & AG_{-a}\forall\rho\forall a'[\rho\bar{a}a']\bar{a}^{0'} \\
\bar{a}^{l'} & = & A((\forall\rho\forall a'[\rho\bar{a}a']\bar{a}^{0'}) \\
 & & U_{-a}((\forall\rho\forall a'[\rho\bar{a}a']\bar{a}^{0'})wedge(\exists\rho\exists a'\langle\rho\bar{a}a'\rangle tt)wedge \\
 & & \forall\alpha((\forall\rho\forall a'(\alpha \neq \rho\bar{a}a')) \Rightarrow [\alpha]ff))) \\
\bar{a}^{w'} & = & AG_{-}\forall\rho\forall a'[\rho\bar{a}a'] \\
 & & A((\forall\rho'\forall a''[\rho'\bar{a}a'']ff) \\
 & & U_{-}(\exists\rho'\exists a''\langle\rho'\bar{a}a''\rangle tt)) \\
\bar{a}^{r'} & = & A((\forall\rho\forall a'[\rho\bar{a}a']ff) \\
 & & U_{-}((\forall\alpha((\forall\rho\forall a'(\rho\bar{a}a' \neq \alpha)) \Rightarrow [\alpha]ff))wedge \\
 & & (\exists\rho\exists a'\langle\rho\bar{a}a'\rangle tt)wedge\bar{a}^{w'})).
\end{array}
$$

Clearly $\|p^t\| \subseteq \|p^{*'}\|$ for every t.
$\|p^{0'}\| \cup \|p^{l'}\| = \|p^{1'}\|$.
$\|p^{0'}\| \cup \|p^{r'}\| = \|p^{w'}\|$.
$\|p^{1'}\| \cap \|p^{w'}\| = \|p^{0'}\|$.
$\|p^{1'}\| \not\subseteq \|p^{w'}\| \not\subseteq \|p^{1'}\|$.
$\|p^{1'}\| \cup \|p^{w'}\| = \|p^{*'}\|$.

4.2.3 Behavioral

4.2.3.1 Quietness

$$\surd \stackrel{df}{=} [\tau]ff.$$

4.2.3.2 Stability

$$\text{stable} \stackrel{df}{=} (-\tau).tt.$$

4.2.3.3 Strong Termination

$$\top \stackrel{df}{=} AF_{\tau}[\tau]ff.$$

4.2.3.4 Port (Sum) Immediate Activeness

We write $l \sqcup l'$ for the concatenation of l with the greatest sublist of l' whose elements do not appear in l.

$\boxed{l_s}$ $l_{\bar{a}} \stackrel{df}{=} \bar{a}$ and $l_a \stackrel{df}{=} a$ and $l_{(r+r')} \stackrel{df}{=} l_r \sqcup l_{r'}$. Eg $l_{(b+(c+b))} = b \sqcup c, b = b, c$.

$$\boxed{s_{IA} \stackrel{df}{=} AG_{-l_s}\langle l_s\rangle tt}.$$

Output Immediate Activeness

$$\bar{a}_{IA} = AG_{-\bar{a}}\langle\bar{a}\rangle tt.$$

Input Immediate Activeness

$$a_{IA} = AG_{-a}\langle a\rangle tt.$$

Input Sum Immediate Activeness

$$(r + r')_{IA} = AG_{-l_r \sqcup l_{r'}}\langle l_r \sqcup l_{r'}\rangle tt.$$

If $l_r = a_1, \ldots, a_n$ then $\langle l_r\rangle tt$ amounts to $\exists x((\langle a_1(x)\rangle tt) \vee \ldots \vee (\langle a_n(x)\rangle tt))$.

4.2.3.5 Port (Sum) Activeness

$$\boxed{s_A \stackrel{df}{=} A(([-\tau]ff)\ U_-\ AG_{-l_s}\langle l_s\rangle tt)}.$$

Output Activeness

$$\bar{a}_A = A(([-\tau]ff)\ U_-\ AG_{-\bar{a}}\langle\bar{a}\rangle tt.$$

Input Activeness

$$a_A = A(([-\tau]ff)\ U_-\ AG_{-a}\langle a\rangle tt.$$

Input Sum Activeness

$$(r + r')_A = A(([-\tau]ff)\ U_-\ AG_{-l_r \sqcup l_{r'}}\langle l_r \sqcup l_{r'}\rangle tt.$$

4.2.3.6 Port Responsiveness

Unifying input and output responsiveness, using $\bar{p}$ like port $a^{\bar{p}}$, p port responsiveness is defined as follows.

$$\boxed{p_R \stackrel{df}{=} AG_-(\forall a[pa]A(([a^{\bar{p}}]ff)U_-(a^{\bar{p}}).tt))}$$

Input Responsiveness

$$\begin{aligned} a_R &= AG_-(\forall x[a(x)]A(([\bar{x}]ff)U_-(\bar{x}).tt)) \\ &= AG_-(\forall x[a(x)] \\ &\quad A((\forall\rho\forall a'[\rho\bar{x}a']ff) \\ &\quad U_-((\exists\rho\exists a'\langle\rho\bar{x}a'\rangle tt)wedge \\ &\qquad \forall\alpha((\forall\rho'\forall a''(\rho'\bar{x}a' \neq \alpha)) \Rightarrow [\alpha]ff)))). \end{aligned}$$

Output Responsiveness

$$\begin{aligned}\bar{a}_R &= AG_-(\forall a'[\bar{a}a']A(([a']ff)U_-(a').tt))\\ &= AG_-(\forall\rho\forall a'[\rho\bar{a}a']\\ &\quad A((\forall x[a'(x)]ff)\\ &\quad U_-((\exists x\langle a'(x)\rangle tt)wedge\\ &\qquad \forall\alpha((\forall x(a'(x)\neq\alpha))\Rightarrow[\alpha]ff)))).\end{aligned}$$

4.2.4 Dependency / Interface

$$\begin{aligned}\phi_1 \triangleleft \phi_2 &\stackrel{df}{=} [\phi_2](\phi_1|\phi_2)\\ \phi_1 \blacktriangleleft \phi_2 &\stackrel{df}{=} \langle\phi_2\rangle(\phi_1||\phi_2)\end{aligned}$$

[Duality between ϕ'-Relativised $\blacktriangleleft\,\phi'$ and $\triangleleft\,\phi'$]

$$\neg(\phi \blacktriangleleft \phi') \stackrel{f}{=} (\neg\phi) \triangleleft \phi'.$$

4.2.5 Process Type

$$(\Sigma;\nabla;\phi) \stackrel{df}{=} (\Sigma;\nabla)wedge\phi$$

4.3 Formula Equivalence

$$\begin{aligned}\text{stable} &\stackrel{f}{=} \surd wedge\langle-\tau\rangle tt\\ \top &\stackrel{f}{=} AF_\tau\surd\\ s_A &\stackrel{f}{=} A(([-\tau]ff)\ U_-\ s_{IA})\\ \bar{a}_{IA} &\stackrel{f}{=} AG_{-\bar{a}}\exists\rho\exists a'\langle\rho\bar{a}a'\rangle tt\\ a_{IA} &\stackrel{f}{=} AG_{-a}\exists x\langle a(x)\rangle tt\\ (\phi_1 \triangleleft \phi_2) &\stackrel{f}{=} (\phi_1 \vee empty(\phi_2))\\ (\phi_1 \blacktriangleleft \phi_2) &\stackrel{f}{=} (\phi_1 wedgene(\phi_2))\end{aligned}$$

4.4 Satisfaction

4.4.1 (Replicated) Linear Multiplicity Satisfaction

The extra next axioms and rules, though intuitively sound, are not well modelled by the count set static semantics, not even with enhancements tried for that purpose.

Linear $\boxed{Q : p^l}$

$$() \ \frac{}{\bar{a}a' : \bar{a}^l}$$

$$(\overset{=}{>}) \ \frac{Q : a^0}{a(x).Q : a^l}$$

$$(\overset{=}{>}) \ \frac{Q : p^l}{a(x).Q : p^l} a \neq p$$

$$(\overset{=}{>}) \ \frac{Q : p^l}{(\nu x)Q : p^l} x \neq n(p)$$

$$(\overset{=}{>}) \ \frac{S_1 : p^l \quad S_2 : p^l}{S_1 + S_2 : p^l}$$

$$(\overset{\leq}{>}) \ \frac{Q_i : p^l \quad Q_{3-i} : p^0 wedge \bar{p}^0}{Q_1|Q_2 : p^l} i \in \{1, 2\}$$

Replicated (Linear) $\boxed{Q : p^r}$

$$(\overset{\leq}{>}) \ \frac{Q : a^0 wedge \bar{a}^0}{!a(x).Q : a^r}$$

$$(\overset{\leq}{>}) \ \frac{Q : p^l wedge \bar{p}^0}{!a(x).Q : p^r} a \neq p$$

$$(\overset{=}{>}) \ \frac{Q : p^r}{a(x).Q : p^r} a \neq p$$

$$(\overset{=}{>}) \ \frac{Q : p^r}{(\nu x)Q : p^r} x \neq n(p)$$

$$(\overset{=}{>}) \ \frac{S_1 : p^r \quad S_2 : p^r}{S_1 + S_2 : p^r}$$

$$(\overset{\leq}{>}) \ \frac{Q_i : p^r \quad Q_{3-i} : p^0 wedge \bar{p}^0}{Q_1|Q_2 : p^r} i \in \{1, 2\}$$

4.4.2 Process Type Derivable Satisfaction

The following rule is derivable.

$$(\overset{>}{>}) \frac{wedge_{i \in \{1,2\}}(P_i : (\Sigma; \nabla; \phi_i \triangleleft \phi_i'))}{(P_1|P_2) : (\Sigma; \nabla; (\phi_1|\phi_2)) \vee (\vee_{i \in \{1,2\}} empty(\phi_i'))}$$

From it, so is the next one.

$$(\overset{>}{>}) \frac{wedge_{i \in \{1,2\}}(P_i : (\Sigma; \nabla; \phi_i \triangleleft \phi_i') wedge \phi_{3-i}')}{(P_1|P_2) : (\Sigma; \nabla; (\phi_1|\phi_2))}$$

4.5 Satisfaction Soundness

4.5.1 Extended Multiplicity Satisfaction Soundness

If $Q : p^m$ then $Q : p^t$ with $t = m'$.

This is obvious for the any $*$ case. For each other multiplicity, in the order in which the temporal formulas are defined above, we expect it may be proved by induction on the derivation of $Q : p^m$. The zero case should be standalone. The affine case should depend on the zero case. The linear case should depend on the zero case. The extreme case might be standalone. The replicated (linear) case should depend on the extreme one.

Chaining with temporal satisfaction soundness, follows If $Q : p^m$ then $Q \in \|p^t\|$ with $t = m'$.

This may be directly proved by induction on the derivation of $Q : p^m$, according to the temporal fragment semantics.

Extending satisfaction soundness from the static multiplicity lattice to the dynamic semilattice, the lemma could also be proved by chaining the extended satisfaction soundness with $\|p^m\| \subseteq \|p^t\|$ with $t = m'$; ie $p(Q) \subseteq V_m$ implies $Q \in \|p^t\|$ with $t = m'$ for every Q.

This asserts the soundness of the multiplicity count set static semantics extension wrt the dynamic multiplicity temporal formula semantics. It may be proved by induction on Q.

4.6 Satisfaction Decidability

4.6.1 Extended Multiplicity Satisfaction Decidability

This extended multiplicity satisfaction $Q : p^m$ is also decidable, since the extra rules are process decreasing.

Part III

Expressiveness

Chapter 5

Formula Translation

$\boxed{\underline{\phi}}$

5.1 Derived Fragments

5.1.1 Process Type

$$\underline{(\Sigma;\nabla;\phi)} \stackrel{df}{=} \underline{\Sigma;\nabla wedge\phi} \tag{5.1}$$

5.1.2 Dependency / Interface

$$\underline{\phi_1 \triangleleft \phi_2} \stackrel{df}{=} \underline{[\phi_2](\phi_1|\phi_2)} \tag{5.2}$$

$$\underline{\phi_1 \blacktriangleleft \phi_2} \stackrel{df}{=} \underline{\langle\phi_2\rangle(\phi_1|\phi_2)} \tag{5.3}$$

5.1.3 Behavioral

$$\underline{\surd} \stackrel{df}{=} \underline{[\tau]ff} \tag{5.4}$$

$$\underline{\text{stable}} \stackrel{df}{=} \underline{(-\tau).tt} \tag{5.5}$$

$$\underline{\top} \stackrel{df}{=} \underline{AF_\tau[\tau]ff} \tag{5.6}$$

5.1.3.1 Port (Sum) Immediate Activeness

$\boxed{\underline{s_{IA}} \stackrel{df}{=} \underline{AG_{-l_s}\langle l_s\rangle tt}}$ where $l_{\bar{a}} \stackrel{df}{=} \bar{a}$ and $l_a \stackrel{df}{=} a$ and $l_{(r+r')} \stackrel{df}{=} l_r \sqcup l_{r'}$ and $l \sqcup l'$ is the concatenation of l with the greatest sublist of l' whose elements do not appear in l.

$$\underline{\bar{a}_{IA}} \stackrel{df}{=} \underline{AG_{-\bar{a}}\langle\bar{a}\rangle tt} \tag{5.7}$$

$$\underline{a_{IA}} \stackrel{df}{=} \underline{AG_{-a}\langle a\rangle tt} \tag{5.8}$$

$$\underline{(r+r')_{IA}} \stackrel{df}{=} \underline{AG_{-l_r \sqcup l_{r'}}\langle l_r \sqcup l_{r'}\rangle tt} \tag{5.9}$$

5.1.3.2 Port (Sum) Activeness

$$\boxed{\underline{s_A} \stackrel{df}{=} \underline{A(([-\tau]ff)\ U_-\ AG_{-l_s}\langle l_s\rangle tt)}}$$

$$\underline{\bar{a}_A} \stackrel{df}{=} \underline{A(([-\tau]ff)\ U_-\ AG_{-\bar{a}}\langle\bar{a}\rangle tt} \tag{5.10}$$

$$\underline{a_A} \stackrel{df}{=} \underline{A(([-\tau]ff)\ U_-\ AG_{-a}\langle a\rangle tt} \tag{5.11}$$

$$\underline{(r+r')_A} \stackrel{df}{=} \underline{A(([-\tau]ff)\ U_-\ AG_{-l_r \sqcup l_{r'}}\langle l_r \sqcup l_{r'}\rangle tt} \tag{5.12}$$

5.1.3.3 Port Responsiveness

$$\boxed{\underline{p_R} \stackrel{df}{=} \underline{AG_-(\forall a[pa]A(([a^{\bar{p}}]ff)U_-(a^{\bar{p}}).tt))}}$$

$$\underline{a_R} \stackrel{df}{=} \underline{AG_-(\forall x[a(x)]A(([\bar{x}]ff)U_-(\bar{x}).tt))} \tag{5.13}$$

$$\underline{\bar{a}_R} \stackrel{df}{=} \underline{AG_-(\forall a'[\bar{a}a']A(([a']ff)U_-(a').tt))} \tag{5.14}$$

5.1.4 (Extended) Dynamic Multiplicity Semilattice

$\boxed{\underline{p^t}}$ For each dynamic multiplicity t we translate the respective characteristic temporal formula.

$$\underline{p^{*'}} \stackrel{df}{=} tt \tag{5.15}$$

$$\underline{p^{0'}} \stackrel{df}{=} \underline{AG_{-p}[p]ff} \tag{5.16}$$

$$\underline{p^{1'}} \stackrel{df}{=} \underline{AG_{-p}[p]p^{0'}} \tag{5.17}$$

$$\underline{p^{l'}} \stackrel{df}{=} \underline{A(([p]p^{0'})U_{-p}(([p]p^{0'})wedge(p).tt))} \tag{5.18}$$

$$\underline{p^{w'}} \stackrel{df}{=} \underline{AG_-[p]A(([p]ff)U_-(\langle p\rangle tt))} \tag{5.19}$$

$$\underline{p^{r'}} \stackrel{df}{=} \underline{A(([p]ff)U_-(((p).tt)wedgep^{w'}))} \tag{5.20}$$

5.2 Dynamic Fragments

5.2.1 KCTL

$$\underline{AX_K\phi} \stackrel{df}{=} \underline{[K]\phi} \tag{5.21}$$

$$\underline{EX_K\phi} \stackrel{df}{=} \underline{\langle K\rangle\phi} \tag{5.22}$$

$$\underline{A(\phi U_K\phi')} \stackrel{df}{=} \underline{\mu(Z)(\phi' \vee (\phi wedge\neg\phi' wedge[K]Z))} \tag{5.23}$$

$$\underline{E(\phi U_K\phi')} \stackrel{df}{=} \underline{\mu(Z)(\phi' \vee (\phi wedge\neg\phi' wedge\langle K\rangle Z))} \tag{5.24}$$

$$\underline{AF_K\phi} \stackrel{df}{=} \underline{\mu(Z)(\phi \vee [K]Z))} \tag{5.25}$$

$$\underline{EF_K\phi} \stackrel{df}{=} \underline{\mu(Z)(\phi \vee \langle K\rangle Z))} \tag{5.26}$$

$$\underline{AG_K\phi} \stackrel{df}{=} \underline{\nu(Z)(\phi wedge[K]Z))} \tag{5.27}$$

$$\underline{EG_K\phi} \stackrel{df}{=} \underline{\nu(Z)(\phi wedge\langle K\rangle Z))} \tag{5.28}$$

$$\underline{A(\phi R_K\phi')} \stackrel{df}{=} \underline{\nu(Z)(\phi' wedge(\phi \vee \neg\phi' \vee [K]Z))} \tag{5.29}$$

$$\underline{E(\phi R_K\phi')} \stackrel{df}{=} \underline{\nu(Z)(\phi' wedge(\phi \vee \neg\phi' \vee \langle K\rangle Z))}. \tag{5.30}$$

5.3 Static Fragments

5.3.1 Extended Static Multiplicity

$$\underline{p^*} \stackrel{df}{=} tt \tag{5.31}$$

$$\begin{aligned}\underline{p^0} \stackrel{df}{=}\ & \underline{\mu(Z)(0 \vee (\exists a\exists a'(p \neq \bar{a} wedge e\bar{a}a'))\vee} \\ & \underline{(\exists a\exists x(p \neq awedgea(x).Z))\vee} \\ & \underline{(\exists a\exists x(p \neq awedge\ !a(x).Z))\vee} \\ & \underline{(\exists x\exists\sigma((x:\sigma)@(x = n(p) \vee Z)))\vee} \\ & \underline{(\mathrm{sum}wedge(Z+Z)) \vee (Z|Z))}\end{aligned} \tag{5.32}$$

$$\begin{aligned}\underline{p^1} \stackrel{df}{=}\ & \underline{\mu(Z)(p^0 \vee (\exists a\exists a'(p = \bar{a} wedge e\bar{a}a'))\vee} \\ & \underline{(\exists a\exists x\ a(x).((p = awedgep^0) \vee (p \neq awedgeZ)))\vee} \\ & \underline{(\exists x\exists\sigma(x \neq n(p)wedge(x:\sigma)@Z))\vee} \\ & \underline{(\mathrm{sum}wedge(Z+Z))\vee} \\ & \underline{(Z|p^0) \vee (p^0|Z))}\end{aligned} \tag{5.33}$$

$$\begin{aligned}\underline{p^w} \stackrel{df}{=}\ & \underline{\mu(Z)(p^0\vee} \\ & \underline{(\exists a\exists x(!a(x).((p = awedgep^0) \vee (p \neq awedgep^1))))\vee} \\ & \underline{(\exists a\exists x(p \neq awedgea(x).Z))\vee}\end{aligned}$$

$$\begin{array}{rcl} & & \underline{(\exists x \exists \sigma (x \neq n(p) wedge(x:\sigma)@Z))\vee} \\ & & \underline{(\mathrm{sum} wedge(Z+Z))\vee} \\ & & \underline{(Z|p^0)\vee(p^0|Z))} \end{array} \tag{5.34}$$

$$\begin{array}{rcl} \underline{p^l} & \stackrel{df}{=} & \underline{\mu(Z)((\exists a \exists a'(p=\bar{a} wedge e \bar{a} a'))\vee} \\ & & \underline{(\exists a \exists x(p = a wedge ea(x).p^0))\vee} \\ & & \underline{(\exists a \exists x(p \neq a wedge ea(x).Z))\vee} \\ & & \underline{(\exists x \exists \sigma (x \neq n(p) wedge(x:\sigma)@Z)\vee} \\ & & \underline{(\mathrm{sum} wedge(Z+Z))\vee} \\ & & \underline{(Z|(p^0 wedge \bar{p}^0))\vee((p^0 wedge \bar{p}^0)|Z)))} \end{array} \tag{5.35}$$

$$\begin{array}{rcl} \underline{p^r} & = & \underline{\mu(Z)(} \\ & & \underline{(\exists a \exists x(p = a\ wedge\ !a(x).(p^0 wedge \bar{p}^0)))\vee} \\ & & \underline{(\exists a \exists x(p \neq a\ wedge\ !a(x).(p^l wedge \bar{p}^0)))\vee} \\ & & \underline{(\exists a \exists x(p \neq a wedge ea(x).Z))\vee} \\ & & \underline{(\exists x \exists \sigma (x \neq n(p) wedge(x:\sigma)@Z))\vee} \\ & & \underline{(\mathrm{sum} wedge(Z+Z))\vee} \\ & & \underline{((Z|(p^0 wedge \bar{p}^0))\vee((p^0 wedge \bar{p}^0)|Z))}. \end{array} \tag{5.36}$$

$\underline{p^m}$ is unambiguous.

For each possible multiplicity, one can always apply the respective case.

$\underline{p^m}$ terminates.

For every case, each translation recurrence is implicitly process decreasing. As processes expressions are finite, so are all paths in the implicit satisfaction recursion tree, which thus is well founded. Moreover such tree is finitely (at most binary) branching. Therefore such tree is finite and the translation terminates.

$\underline{p^m}$ is well defined; ie unambiguous and terminating.

5.3.2 Shape

We translate each shape formula to pre and pattern fragments as follows. First the simpler homomorphic primed connectives:

$$\underline{'0} \stackrel{df}{=} \langle\equiv\rangle 0 \tag{5.37}$$

$$\underline{'\bar{a}a'} \stackrel{df}{=} \langle\equiv\rangle \bar{a}a' \tag{5.38}$$

$$\underline{'(x:\sigma)@\phi} \stackrel{df}{=} \langle\equiv\rangle(x:\sigma)@\underline{\phi} \tag{5.39}$$

$$\underline{'(x:\sigma)\oslash\phi} \stackrel{df}{=} \langle\equiv\rangle(x:\sigma)\oslash\underline{\phi} \tag{5.40}$$

$$\underline{'(\phi_1|\phi_2)} \stackrel{df}{=} \langle\equiv\rangle(\underline{\phi_1}|\underline{\phi_2}) \tag{5.41}$$

$$\underline{'(\phi_1||\phi_2)} \stackrel{df}{=} [\equiv](\underline{\phi_1}||\underline{\phi_2}) \tag{5.42}$$

$$\underline{\phi \triangleright \phi'} \stackrel{df}{=} [\underline{\phi}]\langle\equiv\rangle\underline{\phi'} \tag{5.43}$$

$$\underline{\phi \blacktriangleright \phi'} \stackrel{df}{=} \langle\underline{\phi}\rangle[\equiv]\underline{\phi'} \tag{5.44}$$

$$\underline{H(x)\phi} \stackrel{df}{=} \langle\equiv\rangle(\nu x:\sigma)\underline{\phi} \tag{5.45}$$

$$\underline{'!a(x).\phi} \stackrel{df}{=} \langle\equiv\rangle!a(x).\underline{\phi} \tag{5.46}$$

$$\underline{'\mathrm{sum}} \stackrel{df}{=} \langle\equiv\rangle\mathrm{sum} \tag{5.47}$$

$$\underline{'a(x).\phi} \stackrel{df}{=} \langle\equiv\rangle a(x).\underline{\phi} \tag{5.48}$$

$$\underline{'(\varphi_1+\varphi_2)} \stackrel{df}{=} \langle\equiv\rangle(\underline{\varphi_1}+\underline{\varphi_2}). \tag{5.49}$$

5.3.3 Fixpoints

$$\underline{\mu(Z)\phi} \stackrel{df}{=} \underline{\forall(Z)((valid(\phi \Rightarrow Z)) \Rightarrow Z)} \tag{5.50}$$

$$\underline{\nu(Z)\phi} \stackrel{df}{=} \underline{\exists(Z)((valid(Z \Rightarrow \phi))wedgeZ)}. \tag{5.51}$$

5.3.4 Validity

We translate to pattern and propositional subfragments.

$$\underline{valid(\phi)} \stackrel{df}{=} [\neg\phi]ff \tag{5.52}$$

$$\underline{ne(\phi)} \stackrel{df}{=} \langle\phi\rangle tt \tag{5.53}$$

$$\underline{empty(\phi)} \stackrel{df}{=} [\phi]ff \tag{5.54}$$

This translation is well defined.

On one hand, these three translation cases unambiguously cover all three possible constructors of this fragment formulas.

On the other hand, each translation case eliminates the respective connective. In the first and second cases the translation recurrence is formula decreasing. In the third one it is non decreasing, recurring on the negated formula. If this and the remaining fragments translation terminates, so does this one.

5.4 Basic Fragments Homomorphic Translation

5.4.1 Propositional

$$\begin{aligned} \underline{ff} &\stackrel{df}{=} ff \\ \underline{tt} &\stackrel{df}{=} tt \\ \underline{\neg\phi} &\stackrel{df}{=} \neg\underline{\phi} \\ &\vdots \end{aligned}$$

5.4.2 FO Quantificational

$$\begin{aligned} \underline{\forall x\phi} &\stackrel{df}{=} \forall x\underline{\phi} \\ &\vdots \end{aligned}$$

5.4.3 SO Quantificational

$$\begin{aligned} \underline{\forall Z\phi} &\stackrel{df}{=} \forall Z\underline{\phi} \\ &\vdots \end{aligned}$$

5.4.4 Pre

$$\begin{aligned} \underline{\langle\equiv\rangle\phi} &\stackrel{df}{=} \langle\equiv\rangle\underline{\phi} \\ \underline{[\equiv]\phi} &\stackrel{df}{=} [\equiv]\underline{\phi} \end{aligned}$$

5.4.5 Pattern

$$\begin{aligned} \underline{\%\phi} &\stackrel{df}{=} \%\underline{\phi} \\ \underline{\phi|\phi'} &\stackrel{df}{=} \underline{\phi}|\underline{\phi'} \\ \underline{[\phi]\phi'} &\stackrel{df}{=} [\underline{\phi}]\underline{\phi'} \\ \underline{\langle\phi\rangle\phi'} &\stackrel{df}{=} \langle\underline{\phi}\rangle\underline{\phi'} \\ &\vdots \end{aligned}$$

5.4.6 Nominal / Primitive

$$\begin{aligned} \underline{p = p'} &\stackrel{df}{=} p = p' \\ &\vdots \end{aligned}$$

5.5 Some Formula Translation Equivalences

5.5.1 Validity

$$\begin{aligned} \underline{valid(\phi)} &\stackrel{f}{=} [tt](ff||\underline{\phi}) \\ \underline{ne(\phi)} &\stackrel{f}{=} \langle tt\rangle(ff||\underline{\phi}) \\ \underline{empty(\phi)} &\stackrel{f}{=} [tt](ff||\underline{\neg\phi}) \end{aligned}$$

5.5.2 Fixpoints

$$\begin{aligned} \underline{\mu(Z)\phi} &\stackrel{f}{=} \underline{\forall(Z)(Z \vee ne(\phi wedge \neg Z))} \\ \underline{\nu(Z)\phi} &\stackrel{f}{=} \underline{\exists(Z)(Zwedgevalid(\phi \vee \neg Z))}. \end{aligned}$$

5.5.3 Behavioral

5.5.3.1 Port (Sum) Immediate Activeness

$$\begin{aligned} \underline{\bar{a}_{IA}} &\stackrel{f}{=} \underline{AG_{-\bar{a}}\exists\rho\exists a'\langle\rho\bar{a}a'\rangle tt} \\ \underline{a_{IA}} &\stackrel{f}{=} \underline{AG_{-a}\exists x\langle a(x)\rangle tt} \\ \underline{(r+r')_{IA}} &\stackrel{f}{=} \underline{AG_{-l_r \sqcup l_{r'}}((\langle l_r\rangle tt) \vee (\langle l_r\rangle tt))} \end{aligned}$$

5.5.3.2 Port Responsiveness

$$\begin{aligned} \underline{a_R} &\stackrel{f}{=} \underline{AG_-(\forall x[a(x)]} \\ &\quad \underline{A((\forall\rho\forall a'[\rho\bar{x}a']ff)} \\ &\quad \underline{U_-((\exists\rho\exists a'\langle\rho\bar{x}a'\rangle tt)wedge} \\ &\quad\quad \underline{\forall\alpha((\forall\rho'\forall a''(\rho'\bar{x}a' \neq \alpha)) \Rightarrow [\alpha]ff))))} \\ \underline{\bar{a}_R} &\stackrel{f}{=} \underline{AG_-(\forall\rho\forall a'[\rho\bar{a}a']} \\ &\quad \underline{A((\forall x[a'(x)]ff)} \\ &\quad \underline{U_-((\exists x\langle a'(x)\rangle tt)wedge} \\ &\quad\quad \underline{\forall\alpha((\forall x(a'(x) \neq \alpha)) \Rightarrow [\alpha]ff))))} \end{aligned}$$

5.6 Formula Translation Correctness

[Denotational Correctness] $\underline{\phi} \stackrel{f}{=} \phi$.

5.6.1 Validity

It may be proved by induction on ϕ.

5.6.2 Fixpoints

Standard.

5.6.3 Shape

If ϕ is a shape formula then $\underline{\phi}$ is not. The shape fragment is thus definable from the pre and pattern ones, as a certain blend of these.

It can be seen by inspecting the RHS of each translation rule.

$\vdash_{ueL} (P : \underline{\phi})$ iff $\vdash_{hL} (P : \phi)$.

This may be proved by induction on the structure of ϕ.

The shape rules are derivable from those of pre and pattern.

5.6.4 Extended Static / Dynamic Multiplicity

$Q : \underline{p^m}$ iff $Q : p^m$.

$Q : \underline{p^t}$ iff $Q : p^t$.

Both are expected to hold by induction on Q.

5.7 (Formula) Translation Satisfaction

From formula translation, we obtain what we dub translation satisfaction $\boxed{P : \underline{\phi}}$

[Axiomatic Correctness] $P : \underline{\phi}$ iff $P : \phi$ for every P.

Chapter 6

Satisfaction Translation

$\boxed{\underline{P : \phi}}$

6.1 Dynamic Fragments

We assume $\vee_{i \in \emptyset} J_i \stackrel{df}{=} ff$ and $wedge_{i \in \emptyset} J_i \stackrel{df}{=} tt$.

6.1.1 Action Set Modal

$$(\stackrel{\geq}{\leq}) \ \underline{P : [K]\phi} \stackrel{df}{=} wedge_{\alpha \in \|K\|_P} wedge_{Q \in (P \cdot \alpha)} \underline{Q : [\alpha]\phi} \tag{6.1}$$

$$(\stackrel{\geq}{\leq}) \ \underline{P : \langle K \rangle \phi} \stackrel{df}{=} \vee_{\alpha \in \|K\|_P} \vee_{Q \in (P \cdot \alpha)} \underline{Q : \langle \alpha \rangle \phi} \tag{6.2}$$

$$(\stackrel{\geq}{\leq}) \ \underline{P : (K).\phi} \stackrel{df}{=} (\|{-}K\|_P = \emptyset) wedge (\vee_{\alpha \in \|K\|_P} \vee_{Q \in (P \cdot \alpha)} \underline{Q : \langle \alpha \rangle \phi}) \tag{6.3}$$

6.1.2 Action Modal

$$(\stackrel{\geq}{\leq}) \ \underline{P : [\alpha]\phi} \stackrel{df}{=} wedge_{Q \in (P \cdot \alpha)} \underline{Q : \phi} \tag{6.4}$$

$$(\stackrel{\geq}{\leq}) \ \underline{P : \langle \alpha \rangle \phi} \stackrel{df}{=} \vee_{Q \in (P \cdot \alpha)} \underline{Q : \phi} \tag{6.5}$$

$$(\stackrel{\geq}{\leq}) \ \underline{P : (\alpha).\phi} \stackrel{df}{=} (A(P) = \{\alpha\}) wedge (\vee_{Q \in (P \cdot \alpha)} \underline{Q : \phi}). \tag{6.6}$$

6.2 Static Fragments

6.2.1 Sorting Judgements Translation

6.2.1.1 Binary Sorting Checking Translation

$\boxed{\underline{\Sigma; \nabla \vdash P}}$ Sequentially try each of the following cases. If none (but the last one) applies, return false.

$$
\begin{array}{rcl}
() \ \underline{\Sigma; \nabla \vdash 0} & \stackrel{df}{=} & tt \\
() \ \underline{\Sigma, a : 1 + \sigma, a' : \sigma; \nabla \vdash \bar{a} a'} & \stackrel{df}{=} & tt \\
(\stackrel{\geq}{>}) \ \underline{\Sigma, a : 1 + \sigma; \nabla \vdash ?a(x).Q} & \stackrel{df}{=} & \underline{\Sigma, a : 1 + \sigma, x : \sigma; \nabla \vdash Q} \\
(\stackrel{\geq}{>}) \ \underline{\Sigma; \nabla, x : \sigma \vdash (\nu x) Q} & \stackrel{df}{=} & \underline{\Sigma, x : \sigma; \nabla \vdash Q} \\
(\stackrel{\geq}{>}) \ \underline{\Sigma; \nabla \vdash (\nu x : \sigma) Q} & \stackrel{df}{=} & \underline{\Sigma, x : \sigma; \nabla \vdash Q} \\
(\stackrel{\geq}{>}) \ \underline{\Sigma; \nabla \vdash P_1 | P_2} & \stackrel{df}{=} & wedge_{j \in \{1,2\}} (\underline{\Sigma; \nabla \vdash P_j}) \\
(\stackrel{\geq}{>}) \ \underline{\Sigma; \nabla \vdash S_1 + S_2} & \stackrel{df}{=} & wedge_{j \in \{1,2\}} (\underline{\Sigma; \nabla \vdash S_j}) \\
\text{otherwise} \ () \ \underline{\Sigma; \nabla \vdash P} & \stackrel{df}{=} & ff.
\end{array}
$$

$\underline{\Sigma; \nabla \vdash P}$ is unambiguous.

Indeed one can always apply a single case.

$\underline{\Sigma; \nabla \vdash P}$ terminates.

For every case, each translation recurrence is process decreasing. As processes expressions are finite, so are all paths in the recursion tree, which thus is well founded. Moreover the recursion tree is at most binary branching. Therefore the recursion tree is finite and the translation terminates.

$\underline{\Sigma; \nabla \vdash P}$ is well defined; ie unambiguous and terminating.

6.2.1.2 Tetra Sorting Satisfaction Translation

$\boxed{\underline{P : F; G; H; I}}$ Sequentially try each of the following cases. If none (but the last one) applies, return false.

$$
\begin{array}{rcl}
() \ \underline{0 : \epsilon; \epsilon; \epsilon; \epsilon} & \stackrel{df}{=} & tt \\
() \ \underline{\bar{a} a' : (a : 1 + \sigma, a' : \sigma); \epsilon; \epsilon; \epsilon} & \stackrel{df}{=} & tt \\
(\stackrel{\geq}{>}) \ \underline{?a(x).Q : F, a : 1 + \sigma; G; H; I, x : \sigma} & \stackrel{df}{=} & \underline{Q : F, a : 1 + \sigma, x : \sigma; G; H; I} \vee \\
 & & (x \notin na(Q) wedge \underline{Q : F, a : 1 + \sigma; G; H; I}) \vee \\
 & & (a \notin na(Q) wedge \underline{Q : F, x : \sigma; G; H; I}) \vee
\end{array}
$$

$$(a, x \notin na(Q) wedge \underline{Q : F; G; H; I})$$

$$(\tfrac{\geq}{>}) \ \underline{(\nu x)Q : F; G, x : \sigma; H; I} \stackrel{df}{=} \underline{Q : F, x : \sigma; G; H; I} \vee (x \notin na(Q) wedge \underline{Q : F; G; H; I})$$

$$(\tfrac{\geq}{>}) \ \underline{(\nu x : \sigma)Q : F; G; H, x : \sigma; I} \stackrel{df}{=} \underline{Q : F, x : \sigma; G; H; I} \vee (x \notin na(Q) wedge \underline{Q : F; G; H; I})$$

$$(\tfrac{\geq}{>}) \ \underline{P_1 | P_2 : F; G; H; I} \stackrel{df}{=} wedge_{j \in \{1,2\}}(\underline{P_j : F|_{fn(P_j)}; G|_{ru(P_j)}; H|_{rs(P_j)}; I|_{bi(P_j)}})$$

$$(\tfrac{\geq}{>}) \ \underline{S_1 + S_2 : F; G; H; I} \stackrel{df}{=} wedge_{j \in \{1,2\}}(\underline{S_j : F|_{fn(S_j)}; G|_{ru(S_j)}; H|_{rs(S_j)}; I|_{bi(S_j)}})$$

$$\text{otherwise } () \ \underline{P : F; G; H; I} \stackrel{df}{=} ff$$

$\underline{P : F; G; H; I}$ is unambiguous.

Indeed one can always apply a single case.

$\underline{P : F; G; H; I}$ terminates.

For every case, each translation recurrence is process decreasing. As processes expressions are finite, so are all paths in the recursion tree, which thus is well founded. Moreover the recursion tree is finitely (actually at most binary) branching. Therefore the recursion tree is finite and the translation terminates.

$\underline{P : F; G; H; I}$ is well defined; ie unambiguous and terminating.

6.2.1.3 Sorting Separation Translation

$\boxed{\underline{\Sigma; \nabla \vdash P : F; G; H; I}}$ Sequentially try each of the following cases. If none (but the last one) applies, return false.

$$() \ \underline{\Sigma; \nabla \vdash 0 : \epsilon; \epsilon; \epsilon} \stackrel{df}{=} tt$$

$$() \ \underline{\Sigma, a : 1 + \sigma, a' : \sigma; \nabla \vdash \bar{a}a' : (a : 1 + \sigma, a' : \sigma); \epsilon; \epsilon} \stackrel{df}{=} tt$$

$$(\tfrac{\geq}{>}) \ \underline{\Sigma, a : 1 + \sigma; \nabla \vdash ?a(x).Q : F, a : 1 + \sigma; G; H; I, x : \sigma} \stackrel{df}{=}$$
$$\underline{\Sigma, a : 1 + \sigma, x : \sigma; \nabla \vdash Q : F, a : 1 + \sigma, x : \sigma; G; H; I} \vee$$
$$(x \notin na(Q) wedge \underline{\Sigma, a : 1 + \sigma, x : \sigma; \nabla \vdash Q : F, a : 1 + \sigma; G; H; I}) \vee$$
$$(a \notin na(Q) wedge \underline{\Sigma, a : 1 + \sigma, x : \sigma; \nabla \vdash Q : F, x : \sigma; G; H; I}) \vee$$
$$(a, x \notin na(Q) wedge \underline{\Sigma, a : 1 + \sigma, x : \sigma; \nabla \vdash Q : F; G; H; I})$$

$$(\tfrac{\geq}{>}) \ \underline{\Sigma; \nabla, x : \sigma \vdash (\nu x)Q : F; G, x : \sigma; H; I} \stackrel{df}{=}$$
$$\underline{\Sigma, x : \sigma; \nabla \vdash Q : F, x : \sigma; G; H; I} \vee$$

$$(x \notin na(Q) wedge \underline{\Sigma, x:\sigma; \nabla \vdash Q : F; G; H; I})$$

$$(\overset{\geq}{>}) \ \underline{\Sigma; \nabla \vdash (\nu x:\sigma)Q : F; G; H, x:\sigma; I} \overset{df}{=}$$

$$\underline{\Sigma, x:\sigma; \nabla \vdash Q : F, x:\sigma; G; H; I} \vee$$

$$(x \notin na(Q) wedge \underline{\Sigma, x:\sigma; \nabla \vdash Q : F; G; H; I})$$

$$(\overset{\geq}{>}) \ \underline{\Sigma; \nabla \vdash P_1|P_2 : F; G; H; I} \overset{df}{=}$$

$$wedge_{j\in\{1,2\}}(\underline{\Sigma; \nabla \vdash P_j : F|_{fn(P_j)}; G|_{ru(P_j)}; H|_{rs(P_j)}; I|_{bi(P_j)}})$$

$$(\overset{\geq}{>}) \ \underline{\Sigma; \nabla \vdash S_1 + S_2 : F; G; H; I} \overset{df}{=}$$

$$wedge_{j\in\{1,2\}}(\underline{\Sigma; \nabla \vdash S_j : F|_{fn(S_j)}; G|_{ru(S_j)}; H|_{rs(S_j)}; I|_{bi(S_j)}})$$

$$\text{otherwise } () \ \underline{\Sigma; \nabla \vdash P : F; G; H; I} \overset{df}{=} ff.$$

$\underline{\Sigma; \nabla \vdash P : F; G; H; I}$ is unambiguous.

Indeed one can always apply a single case.

$\underline{\Sigma; \nabla \vdash P : F; G; H; I}$ terminates.

For every case, each translation recurrence is process decreasing. As processes expressions are finite, so are all paths in the recursion tree, which thus is well founded. Moreover the recursion tree is finitely (actually at most binary) branching. Therefore the recursion tree is finite and the translation terminates.

$\underline{\Sigma; \nabla \vdash P : F; G; H; I}$ is well defined; ie unambiguous and terminating.

6.3 Sorting Judgement Translation Correctness

6.3.1 Binary Sorting Checking

$\underline{\Sigma; \nabla \vdash P}$ iff $\Sigma; \nabla \vdash P$.

This may be proved by induction on P.

6.3.2 Tetra Sorting Satisfaction

$\underline{P : F; G; H; I}$ iff $P : F; G; H; I$.

This may be proved by induction on P.

6.3.3 Sorting Separation

$\underline{\Sigma; \nabla \vdash P : F; G; H; I}$ iff $\Sigma; \nabla \vdash P : F; G; H; I$.

This may be proved by induction on P.

$\underline{\Sigma; \nabla \vdash P : F; G; H; I}$ iff $\underline{\Sigma; \nabla \vdash P}$ and $\underline{P : F; G; H; I}$ and $F \subseteq \Sigma$ and $G \subseteq \nabla$ and $\mathrm{uni}(\Sigma, \nabla, H, I)$.