3 Prefix ACP with Relational Renaming

The language that I will use for my expressiveness results is a variant of ACP [3] that could be called *prefix ACP with relational renaming*. Like ACP this language has two parameters: an alphabet A of *actions* and a partial *communication function* $|: A^2 \to A$, which is commutative and associative, i.e.

- $a|b = b|a$ (commutativity)
- $(a|b)|c = a|(b|c)$ (associativity)

for all $a, b, c \in A$ (and each side of these equations is defined just when the other side is). I will denote this language as $\mathrm{aprACP}_R(A, |)$.

Its signature contains a constant 0 denoting inaction, two binary operators $+$ and $\|$ denoting *alternative* and *parallel composition* respectively, a unary operator a for any action $a \in A$, a unary *encapsulation* operator ∂_H for any $H \subseteq A$ and a *relational renaming* operator ρ_R for any binary relation $R \subseteq A \times A$.

$p\|q$ represents the independent execution of the processes p and q, partly synchronised by the communication function $|$. If $a|b$ is defined, an occurrence of a in p can synchronise with an occurrence of b in q into a communication action $a|b$ between p and q. If $a|b$ is not defined, no such communication is possible. The action a of p can, instead of synchronising with an action of q, (also) appear independent of q, and likewise can b occur independently of p.

The process ap first performs the action $a \in A$ and then behaves like p. $\partial_H(p)$ behaves like p, but without the possibility of performing actions from H. The operator ρ_R is a slight generalisation of the relabelling and (inverse) image operators of CCS and CSP. Process $\rho_R(p)$ behaves just like process p, except that if p has the possibility of doing an a, $\rho_R(p)$ can do any one action b that is related to a via R.

In $\mathrm{aprACP}_R(A, |)$-expressions brackets are omitted under the convention that a binds strongest and $+$ weakest. Besides aprACP with relation renaming I also consider aprACP with functional renaming, denoted $\mathrm{aprACP}_F(A, |)$. This is the same language, but with a renaming operator ρ_f only for every *function* $f : A \to A$ instead of for every relation.

The action rules for $\mathrm{aprACP}_R(A, |)$ are given in Table 1, thereby completing the formalisation of this language as a TSS. These rules determine an interpretation of the $\mathrm{aprACP}_R(A, |)$-expressions in $\mathbb{G}(A)$. This interpretation agrees, up to bisimulation equivalence, with the more denotational interpretation of $\mathrm{ACP}(A, |)$ in $\mathbb{G}(A)$ given in [Baeten,] Bergstra & Klop [3, 2].

It is common to regard the entries in tables like 1 as schemata, each of which denotes a rule for any proper instantiation of the metavariables a, b, c by real actions from A. Thus in case A is infinite, there are infinitely many rules for every operator. In an attempt at "finitisation" I will in this paper regard each of the first six entries as single rules of an abstract TSS. The rule $\dfrac{x \xrightarrow{a} x'}{x\|y \xrightarrow{a} x'\|y}$ for instance, should be read as $\dfrac{x \longrightarrow x',\ Id}{x\|y \longrightarrow x'\|y}$ where $Id \subseteq A \times A$ is the identity relation on A. In the rules were Pr is not the identity, it is explicitly given. The last three entries in Table 1 remain schemata even when interpreted as abstract action rules. There is namely one rule for every encapsulation operator, one for every relational renaming, and one for every pair $\langle X|S\rangle$ with S a recursive specification and $X \in V_S$. But at least there are now finitely many rules for every operator, even if A is infinite. This will turn out to be a useful property.

$$ax \xrightarrow{a} x \qquad \frac{x \xrightarrow{a} x'}{x+y \xrightarrow{a} x'} \qquad \frac{y \xrightarrow{a} y'}{x+y \xrightarrow{a} y'}$$

$$\frac{x \xrightarrow{a} x'}{x\|y \xrightarrow{a} x'\|y} \qquad \frac{x \xrightarrow{a} x',\ y \xrightarrow{b} y',\ a|b=c}{x\|y \xrightarrow{c} x'\|y'} \qquad \frac{y \xrightarrow{a} y'}{x\|y \xrightarrow{a} x\|y'}$$

$$\frac{x \xrightarrow{a} x',\ a \notin H}{\partial_H(x) \xrightarrow{a} \partial_H(x')} \qquad \frac{S_x[\langle Y|S\rangle/Y]_{Y\in V_S} \xrightarrow{a} z}{\langle X|S\rangle \xrightarrow{a} z} \qquad \frac{x \xrightarrow{a} x',\ R(a,b)}{\rho_R(x) \xrightarrow{b} \rho_R(x')}$$

Table 1: aprACP$_R$

3.1 CCS

Milner's Calculus of Communicating Systems (CCS) [8] can be regarded as (a sublanguage of) an instantiation of aprACP with functional renaming. CCS is parametrised with a set $\mathcal{A}$ of *names*. The set $\bar{\mathcal{A}}$ of *co-names* is given by $\bar{\mathcal{A}} = \{\bar{a} \mid a \in \mathcal{A}\}$, and $\mathcal{L} = \mathcal{A} \cup \bar{\mathcal{A}}$ is the set of *labels*. The function $\bar{\cdot}$ is extended to $\mathcal{L}$ by declaring $\bar{\bar{a}} = a$. Finally $Act = L \dot{\cup} \{\tau\}$ is the set of *actions*. CCS can now be presented as aprACP$(Act, |)$, where $|$ is the partial function on Act given by $a|\bar{a} = \tau$.

In CCS there is a renaming operator only for every function $f : Act \to Act$ that satisfies $f(\bar{a}) = \overline{f(a)}$ and $f(\tau) = \tau$. This operator (applied on a process p) is written $p[f]$ and called *relabelling*. Also there is an encapsulation operator ∂_H only when $H = A \cup \bar{A}$ with $A \subseteq \mathcal{A}$. This operator is called *restriction* and is written $p\backslash A$. Parallel composition is written $|$ instead of $\|$, but there is no further difference between CCS and aprACP$_F(Act, |)$.

3.2 ACP

There are many methodological differences between the ACP approach to *process algebra* and the CCS approach. I will not address these here. As a language, ACP can be regarded as a modification of CCS in four directions.

- First of all, ACP makes a distinction between deadlock and successful termination. As a consequence, action prefixing can be replaced by action constants and a general sequential composition.
- ACP adds two auxiliary operators, the *left merge* and the *communication merge*, denoted $\between$ and $|$, to enable a finite equational axiomatisation of the parallel composition.
- Whereas CCS combines communication and abstraction from internal actions in one operator, in ACP these activities are separated. In CCS the result of any communication is the unobservable action τ. In ACP it is an observable action, from which (in the extended language ACP$_\tau$) one can abstract by applying an *abstraction* operator, renaming designated actions into τ.

- CCS adheres to a specific communication format, admitting only handshaking communication, whereas ACP allows a variety of communication paradigms, including ternary communication, through the choice of the communication function $|$.

In this paper only the last feature of ACP is of importance. I don't distinguish observable and unobservable actions and therefore work with ACP rather than ACP_τ. As I also don't deal with the distinction between deadlock and successful termination, I restrict attention to the sublanguage aprACP of ACP that doesn't make this distinction and consequently supports prefixing only. Whereas in the original papers on ACP the set A of action constants was required to be finite, I allow it to be infinite. I add the subscript R (or F) to indicate the addition of (functional) renaming operators, which were not included in the syntax of ACP. Subsequently I drop the auxiliary operators $\between$ and $|$ from the language, since these can be expressed in the other operators of aprACP_F, as I will show in Section 3.5. The resulting language extends CCS in only one essential way, namely through the general communication format. This generality greatly enhances the expressiveness of the language.

3.3 Meije

Like CCS, also Boudol's language Meije [1, 4, 13] can be regarded as (a sublanguage of) an instantiation of aprACP with functional renaming. Meije is parametrised with a set $\mathcal{A}$ of *atomic actions* and a set $\mathcal{S}$ of *signals*. The set Act of *actions* is a commutative monoid, namely the free commutative product of the free commutative monoid generated by $\mathcal{A}$ and the free commutative group generated by $\mathcal{S}$. This means that the elements of Act are a kind of multisets over $\mathcal{A}$ and $\mathcal{S}$ with the stipulation that elements of $\mathcal{S}$ may also have negative multiplicities. These can also be seen as ordinary multisets over $\mathcal{A}$, $\mathcal{S}$ and $\mathcal{S}^{-1} = \{s^{-1} \mid s \in \mathcal{S}\}$ in which s and s^{-1} cancel. A typical element of Act is denoted as $a^5b^2s^3t^{-1}$. The product operation . on Act is such that $a^5b^2s^3t^{-1}.a^2c^4s^{-3}t^3u^{-3} = a^7b^2c^4t^2u^{-3}$. Meije can now be presented as a sublanguage of $\text{aprACP}_F(Act, .)$.

In Meije there are two kind of renaming operators. There is a renaming operator ρ_Φ, written $\langle\Phi\rangle(p)$, for any *morphism* $\Phi : Act \to Act$. Here a morphism is a function satisfying $\Phi(a.b) = \Phi(a).\Phi(b)$ for $a, b \in Act$. In addition there is a renaming operator $s * p$, called *ticking*, for any signal $s \in S$. This operator renames any action a into $s.a$. The only type of encapsulation operators permitted in Meije are the restriction operators $p\backslash s$ for any signal $s \in S$. $p\backslash s$ is $\partial_H(p)$ for H the set of all actions containing s, i.e. all "multisets" in which s has a positive or negative multiplicity. Meije also has an operator *triggering* which is expressible in the others, and it lacks the operator $+$, because, as explained in Section 3.6, that operator is expressible in the others as well, but there is no further difference between Meije and $\text{aprACP}_F(Act, .)$.

3.4 A Decidable Signature for aprACP

The language $\text{aprACP}_R(A, |)$ defined so far has an uncountable signature if A is infinite. There are namely uncountably many encapsulation and renaming operators. Computationally it makes sense to restrict attention to a fragment of

aprACP$_R$ with a decidable signature. This can be achieved by requiring the set of actions A to be decidable, and by restricting the permitted encapsulation and renaming operators ∂_H and ρ_R to the ones where $A - H$ and R are recursively enumerable sets. Such sets can be represented by the code of a Turing machine that enumerates them, and it is decidable whether an arbitrary piece of text is the code of a Turing machine enumerating a recursive enumerable set. This makes the signature decidable. As a consequence the set of recursion-free terms will be decidable too.

I could have chosen H to be enumerable instead of its complement $A - H$. However, the choice above has the advantage that the encapsulation operators can (in an obvious way) be regarded as special relational renamings, so that one has one kind of operator less to be concerned about. A more compelling argument will be presented in Section 4.4.

In order to ensure that the set of recursive terms is decidable as well, I have to require recursive specifications, seen as sets of equations, to be recursively enumerable at least. This makes the set of open terms decidable. However, it remains undecidable whether a term is closed. The set of closed terms is not even enumerable.

Therefore one may wish to insist that in terms of the form $\langle X|S\rangle$ the set of recursion variables V_S is decidable as well. This makes S computable. However, it is undecidable whether a piece of text is the code of a Turing machine deciding membership of a set. Thus with computable recursion even the set of open terms becomes undecidable again.

Hence an even more restrictive requirement on the desired kind of recursion is in order. Here I require S to be primitive decidable. This means that there is a primitive recursive function deciding membership of V_S, and in case of a variable $X \in V_S$ returning the term S_X. It is decidable whether a piece of text is the source of a primitive recursive function, thus with this restriction the signature as well as the sets of open and closed terms are decidable.

If moreover the communication function is required to be partial recursive, the resulting variant of aprACP$_R$ will be denoted aprACP$_R^{\text{r.e.}\cdot}$. The other languages I mentioned can be adapted in the same way. In aprACP$_F^{\text{r.e.}\cdot}$ I have to allow partial recursive renaming functions. In the original version of aprACP$_F$, these where expressible in terms of total renamings and encapsulation.

3.5 Expressing the Left- and Communication Merge

The language ACP has two operators, $\between$ and $|$, that I didn't include in the syntax of aprACP$_F$. The reason is that these operators can be expressed in the other operators of the language, and thus need not be introduced as primitives. Here I show how. Table 2 shows the action rules for the two operators. The

$$\frac{x \xrightarrow{a} x'}{x \between y \xrightarrow{a} x' \| y} \qquad \frac{x \xrightarrow{a} x',\ y \xrightarrow{b} y',\ a|b = c}{x \mid y \xrightarrow{c} x' \| y'}$$

Table 2: The left- and communication merge of ACP

left merge, $\between$, behaves exactly like the *merge* or parallel composition, $\|$, except that the first action is required to come from its leftmost argument. The communication merge, $|$, behaves exactly like $\|$, except that its first action is required to be a communication between its two arguments. The operators' most crucial use is in the axiom CM1

$$x\|y = x \between y + y \between x + x \mid y$$

that plays an essential rôle in axiomatising ACP with bisimulation semantics.

Note that the symbol $|$ is used for the communication function as well as the communication merge. This overloading is intentional, as the communication merge can be thought of as an extension of the communication function, which is defined on actions only. The vertical bar in the middle of an expression $\langle X|S\rangle$ is pronounced *where*—and sometimes even written that way—and has nothing to do with the communication function and merge. The vertical bar in a set expression like $\{n \in \mathbb{N} \mid n > 5\}$ is also pronounced *where* and constitutes a fourth use of this symbol. Finally $|$ is used to denote parallel composition in CCS. It is generally easy to determine from the context which $|$ is meant.

In order to express $\between$ and $|$ in $\mathrm{aprACP}_F(A,|)$ I assume that the set of actions A is divided into a set A_0 of actions that may be encountered in applications, and the remainder $H_0 = A - A_0$, which is used as a working space for implementing useful operators, such as $\between$ and $|$. On A_0 the communication function is dictated by the applications, but on H_0 I can choose it in any way that suits me. The cardinality of A_0 should be infinite, and equal to the cardinality of A.

For today's implementation I assume that H_0 contains actions $\mathtt{skip}$, $\mathtt{first}$, $\mathtt{next}$, $a_{\mathtt{first}}$ and $a_{\mathtt{next}}$ (for $a \in A_0$). The communication function given on A_0 is extended to these actions as indicated below (applying the convention that if $a|b$ is not defined it is undefined).

$$\begin{array}{|ll|}\hline a \mid \mathtt{first} = a_{\mathtt{first}} & (a \in A_0)\\ a \mid \mathtt{next} = a_{\mathtt{next}} & (a \in A_0)\\ a_{\mathtt{first}} \mid b_{\mathtt{first}} = a \mid b & (a,b \in A_0)\\ a \mid \mathtt{skip} = a & (a \in A_0)\\ \hline\end{array}$$

Let $H_1 = A_0 \cup \{\mathtt{first}, \mathtt{next}\}$. I will use a renaming operator f_1 that satisfies $f_1(a_{\mathtt{next}}) = a$, $f_1(a_{\mathtt{first}}) = a_{\mathtt{first}}$ and $f_1(a) = a$ for $a \in A_0$.

Let me first introduce the notation a^∞ to denote a process that perpetually performs the action a. This process is obtained as $a^\infty = \langle X \mid X = aX\rangle$. Now suppose p is a process that can do actions from A_0 only. Then

$$\partial_{H_1}(p\|\mathtt{first}(\mathtt{next}^\infty))$$

is a process that behaves exactly like p, except that every initial action has a tag (subscript) $\mathtt{first}$, and every non-initial action has a tag $\mathtt{next}$. Thus $\rho_{f_1}(\partial_{H_1}[p\|\mathtt{first}(\mathtt{next}^\infty)])$ is a process that behaves exactly like p, except that the initial actions are tagged $\mathtt{first}$. It follows that for any two processes p and q with actions from A_0 only

$$p \mid q \leftrightarrows \partial_{H_0}\left(\rho_{f_1}(\partial_{H_1}[p\|\mathtt{first}(\mathtt{next}^\infty)]) \Big\| \rho_{f_1}(\partial_{H_1}[q\|\mathtt{first}(\mathtt{next}^\infty)])\right).$$

In order to extend this result to processes with actions outside A_0 I use a bijective renaming $f_0 : A \to A_0$ and its inverse f_0^{-1}. The communication merge is expressed in aprACP$_F(A, |)$, up to bisimulation equivalence, by

$$x \mid y \leftrightarrows \rho_{f_0^{-1}}\left(\partial_{H_0}\left(\rho_{f_1}(\partial_{H_1}[\rho_{f_0}(x)\|\mathtt{first}(\mathtt{next}^\infty)])\Big|\Big|\rho_{f_1}(\partial_{H_1}[\rho_{f_0}(y)\|\cdots])\right)\right).$$

Finally the left merge is expressed in terms of the communication merge through

$$x \mathbin{\underline{\|}} y \leftrightarrows \rho_{f_0^{-1}}(\mathtt{skip}(\rho_{f_0}(y)) \mid \rho_{f_0}(x)).$$

3.6 Expressing Choice

As remarked in Section 3.3, the language Meije lacks the choice operator + of CCS and ACP. The reason this operator was omitted from the syntax of Meije was that it can be expressed in terms of the other operators. This is true in the setting of aprACP$_F$ as well, so if one likes, this operator can be skipped from the signature.

For the implementation of choice, A is again divided in A_0 and H_0 and in H_0 we put the same actions as in the previous section, together with the action `choose`. The communication function also works as before, except that there is no communication possible between actions of the form $a_{\mathtt{first}}$ and $b_{\mathtt{first}}$. (So maybe one wants to use a different action `first`.) Instead one has the communication $\mathtt{choose} \mid a_{\mathtt{first}} = a$ for $a \in A_0$. Recall that for p a process that can do actions from A_0 only, $\rho_{f_1}(\partial_{H_1}[p\|\mathtt{first}(\mathtt{next}^\infty)])$ is the same process in which the initial actions are tagged with a subscript `first`. This, by the way, is an implementation of the operator *triggering* of Meije. It follows that

$$p+q \leftrightarrows \partial_{H_0}\left(\rho_{f_1}(\partial_{H_1}[p\|\mathtt{first}(\mathtt{next}^\infty)])\Big|\Big|\mathtt{choose}\Big|\Big|\rho_{f_1}(\partial_{H_1}[q\|\mathtt{first}(\mathtt{next}^\infty)])\right)$$

since in the expression on the right `choose` can communicate with an initial action from only one of p or q, so that the other one is blocked forever. Using the renamings ρ_{f_0} and its inverse, just as in the previous section, + is expressed in the rest of aprACP$_F$.

3.7 Expressing the Communication Function

It may be felt as a drawback that the language (apr)ACP, unlike CCS, is parametrised by the choice of a communication function (besides the choice of a set of actions). This, one could argue, makes it into a collection of languages rather then a single one. Personally I do not share this concern. If in different applications different communication functions are used, they can, when desired, all be regarded as different fragments of the same communication function, each considered on only a small subset of the set of actions A. At any given time there is no need to know all actions and the entire communication function to be used in all further applications.

Alternatively one may argue that there are many parallel compositions possible in ACP, namely one for every choice of a communication function. Here I will present one instantiation of aprACP$_F(A, |)$, such that for every other choice of a communication function the resulting parallel composition is expressible in this language.

Let, as in the previous section, $A_0 \subseteq A$ be the actions that are used in applications and $H_0 = A - A_0$ the working space. I fix a bijection $f_0 : A \to A_0$ and abbreviate ρ_{f_0} by ρ_0. For this implementation, H_0 should contain the actions (a, b) for every $a, b \in A_0$, as well as an action δ denoting deadlock. The communication function $|$ is defined by $a|b = (a, b)$ for $a, b \in A_0$ (thus undefined outside A_0). Now for any other communication function $\gamma : A^2 \to A$ let $\bar{\gamma} : A \to A$ be a renaming satisfying $\bar{\gamma}((\rho_0(a), \rho_0(b))) = c$ for those $a, b, c \in A$ with $\gamma(a, b) = c$, and $\bar{\gamma}(a) = f_0^{-1}(a)$ for $a \in A_0$. Furthermore, let H be $\{(a, b) \in A_0 \times A_0 \mid \gamma(a, b) \text{ undefined}\}$. Then the associated parallel composition $\|_\gamma$ is expressible in $\text{aprACP}_F(A, |)$ by

$$x\|_\gamma y \leftrightarrows \rho_{\bar{\gamma}}(\partial_H(\rho_0(x)\|\rho_0(y))).$$

3.8 Expressing Renaming and Encapsulation

The syntax of $\text{aprACP}_F^{\text{r.e.}\cdot}$ allows for a multitude of renaming operators. Here I show that one needs only two, namely the operator ρ_0, introduced earlier, which bijectively maps every action to one in the subset A_0 of A, and the *universal renaming operator* ρ_F. Every other functional renaming is then expressible.

To this end I introduce an action $\bar{f} \in H_0 = A - A_0$ for every partial recursive renaming function $f : A \to A$. I also introduce an action $(\bar{f}, a) \in H_0$ for every such f and every $a \in A_0$. The communication function is enriched by $\bar{f} \mid a = (\bar{f}, a)$ for $a \in A_0$ and the universal renaming F should satisfy $F((\bar{f}, \rho_0(a))) = f(a)$. Let $H_1 = A_0 \cup \{\bar{f} \mid f : A \to A\}$. Then $\partial_{H_1}(\bar{f}^\infty\|\rho_0(x))$ is a process that behaves exactly like x, except that every action a is renamed in $(\bar{f}, \rho_0(a))$. Hence ρ_f is expressible through $\rho_f(x) \leftrightarrows \rho_F(\partial_{H_1}(\bar{f}^\infty\|\rho_0(x)))$.

Note that every co-enumerable encapsulation operator can be regarded as a partial recursive renaming, so also all encapsulation operators can be expressed in aprACP with ρ_0 and ρ_F. The encapsulation ∂_{H_1} used in the construction can be incorporated in ρ_F as well.

In exactly the same way every enumerable relational renaming operator is expressible in aprACP with only ρ_0 and a *universal relational renaming*.

In the preceding section I showed how all operators $\|_\gamma$ with γ a communication function could be expressed in a particular instantiation of aprACP_F, using a multitude of renamings. Here I showed how all renaming operators of $\text{aprACP}_F^{\text{r.e.}\cdot}$ can be expressed in only two of them, using a particular communication function, and similarly for the encapsulations. It is an easy exercise to combine these results, and express all partial recursive renaming operators, all co-enumerable encapsulations, and all parallel compositions $\|_\gamma$ with γ a partial recursive communication function in a particular instantiation of $\text{aprACP}_F^{\text{r.e.}\cdot}$, with only one communication function and two renamings.

One may wonder whether *all* renaming operators can be expressed in aprACP, i.e. if it is possible to get rid of the last two. In general this is not possible. However, if one only cares about the behaviour of all the derived operators on the relevant subset A_0 of A, it is possible to omit the use of ρ_0 from all the constructions, encode the universal renaming in the communication function, and find for any derived operator (such as $\between$ or a renaming) an aprACP-expression with the same behaviour on A_0.

3.9 A Finite Signature for aprACP

Here I show how the syntax of $\mathrm{aprACP}^{\mathrm{r,e.}}_R$ can be reduced from a decidable one to a finite one. In the previous section the set of renaming and encapsulation operators was cut down to two elements, so we are left with an infinity of actions to get rid of. As in $\mathrm{aprACP}^{\mathrm{r,e.}}_R$ the set of actions is decidable, they can be numbered $a_0, a_1, a_2, \ldots$, such that the function $\mathtt{succ} : A \to A$ given by $\mathtt{succ}(a_i) = a_{i+1}$ is partial recursive (even computable). Hence every action is expressible in terms of a_0 and the renaming operator $\rho_{\mathtt{succ}}$.

3.10 Expressing Relational Renaming

In order to express relational renamings in aprACP_F one needs to add just one constant (or a third renaming) to the signature. One has to assume the existence of actions $[a, b]$ in H_0 for $a, b \in A_0$. The desired constant is $\mathtt{all} = \Sigma_{a,b\in A_0}[a, b]0$. Here $\Sigma_{i\in I}$ is an infinite version of choice, to be formally introduced in Section 4.1. $\Sigma_{i\in I}$ is not a standard ingredient in the syntax of aprACP—if it were there would be no reason to add $\mathtt{all}$ as a constant. $\mathtt{all}$ can be expressed as $\rho_R(c0)$ where c is an action chosen from A and R is the relation $\{(c, [a, b]) \mid a, b \in A_0\}$.

Now $\mathtt{allever} = \langle X \mid X = \mathtt{all} \between X\rangle$ is a process that perpetually performs an action of the form $[a, b]$, and at each step has the choice between all such actions. For any relation $R \subseteq A \times A$ the process $\partial_{A\times A-R}(\mathtt{allever})$ has at each step the choice between executing one the actions $[a, b]$ with $R(a, b)$. Let $\mathtt{copy} : A_0 \to A$ be a renaming that sends each action $a \in A_0$ to the (new) action $a_{\mathtt{copy}}$. Define the communication function on the new actions by $a_{\mathtt{copy}} \mid [a, b] = b$. Then for p a process that does actions from A_0 only

$$\rho_R(p) \leftrightarrows \partial_{H_0}(\rho_{\mathtt{copy}}(p) \between \partial_{A\times A-R}(\mathtt{allever}))$$

Thus, by means of ρ_0 and its inverse to deal with action from outside A_0, all relational renaming operators are expressible in aprACP_F with $\mathtt{all}$.

$\mathtt{all}$ can also be expressed in aprACP_F using so-called *unguarded recursion* (Section 4.3) as $\mathtt{all} = \langle X \mid X = [a_0, a_0]0 + \mathtt{succ2}(X)\rangle$ where $\mathtt{succ2}$ is a renaming function enumerating the elements of $A \times A$. Thus, as long as unguarded recursion is permitted, aprACP_F is equally expressive as aprACP_R. When unguarded recursion is banned, however, aprACP_R turns out to be more expressive.

4 Specifying Process Graphs

In this section I will isolate, for each of the classes of process graphs mentioned in the introduction, a corresponding class of process expressions that denote only graphs from that class. When possible, I make these classes of expressions so large that every graph of the appropriate kind can be denoted by an expression in the corresponding class.

Definition 11 (*Kinds of graphs*). A process graph $G = (S, T, I) \in \mathbb{G}(A)$ is

- κ-*bounded* (for an uncountable cardinal κ) if for every state $s \in S$ there are less than κ outgoing transitions $s \xrightarrow{a} s'$,