EasyChair Preprint

№ 439

Pointing to Private Names

Adrian Francalanza, Marco Giunti and António Ravara

August 17, 2018

Pointing to Private Names

Adrian Francalanza
Departement of Computer Science, ICT
University of Malta, Malta

Marco Giunti
Department of Informatics, Faculty of Sciences
University of Lisbon
NOVA Laboratory for Computer Science and Informatics

António Ravara
Department of Informatics, Faculty of Sciences and Technology
NOVA University of Lisbon
NOVA Laboratory for Computer Science and Informatics

Scoped channels, in the π-calculus, are not nameable, as they are bound and subject to alpha-renaming. For program analysis purposes, however, to identify properties of these channels, it is necessary to talk about them. We present herein a method for uniquely identifying scoped channels.

1 Introduction

In process calculi like the π-calculus [2, 3, 4], the `new` operator has two roles: it creates a fresh channel and binds its occurrences in a declared scope. The usual semantic rules dealing with binders apply, namely those of the λ-calculus [1].

A basic rule of the operational semantics is α-conversion, i.e., the simultaneous substitution of all occurrences of a bound identifier in a given scope by another one, usually taking into account care to avoid capturing free identifiers.

So, the identities of bound identifiers are actually meaningless, as they can change. However, to develop program analysis methods like particular occurrences of a bound identifier to pinpoint program defects, for instance, require the ability to name such occurrences of the bound identifier, what seems to be a contradiction in terms. We address the problem by associating with each syntactic occurrence of an identifier in a `new` operator

In short, our contribution is the following: a syntactic mechanism, simple to automatise, that generates unique identifiers associated with each scoped name. The uniqueness of these identifiers is preserved by reduction, the usual operational semantics mechanism of the calculus. This mechanism is usefult for program analysis purposes, like detecting deadlocks on scoped names.

2 Syntax and semantics

The syntax of the process language is inductively defined by the grammar in Figure 1. As usual, u, v range over *names* and n, x over name variables. Moreover, h, i, j range over natural numbers.

The distinctive characteristic of our language is the use of labels to uniquely identify private names. This paper is dedicated to show that the reduction semantics of our language indeed guarantees label uniqueness.

Submitted to:
DCM 2018

Definition 2.1 (Process Labels)**.** *Let the following sets be inductively defined by the given rule and by homomorphic rules on the remaining process constructs.*

1. $\mathsf{secLabs}((\mathit{newn} : (h,i))P) = \{i\} \cup \mathsf{secLabs}(P)$
2. $\mathsf{labelPairs}((\mathit{newn} : (h,i))P) = \{(h,i)\} \cup \mathsf{labelPairs}(P)$

We work with well formed processes, where label pairs occur linearly. To define the concept precisely, we need to define the multiset of subprocesses of a process.

Definition 2.2 (Subprocesses)**.** *The multiset of the subprocesses of a process P is inductively generated by the following rules.*

$$\begin{aligned}
\mathsf{subprocs}(\mathit{nil}) &= \{\mathit{nil}\} \\
\mathsf{subprocs}(u!v.P) &= \{u!v.P\} \uplus \mathsf{subprocs}(P) \\
\mathsf{subprocs}(u?x.P) &= \{u?x.P\} \uplus \mathsf{subprocs}(P) \\
\mathsf{subprocs}(*u?x.P) &= \{*u?x.P\} \uplus \mathsf{subprocs}(P) \\
\mathsf{subprocs}((\mathit{newn} : (h,i))P) &= \{(\mathit{newn} : (h,i))P\} \uplus \mathsf{subprocs}(P) \\
\mathsf{subprocs}(P \parallel Q) &= \mathsf{subprocs}(P) \uplus \mathsf{subprocs}(Q)
\end{aligned}$$

We are now ready to define what is a well-formed process.

Definition 2.3 (Well-Formedness)**.** *A process P is* well-formed *(and we write* $\mathsf{wf}(P)$*) if when there is a set* $S \in \mathsf{subprocs}(P)$ *such that* $\{(\mathit{newn} : (h,i))Q, (\mathit{newn} : (h',j))Q'\} \subseteq S$ *then* $i \neq j$*.*

From now on we simply say 'P well-formed' whenever $\mathsf{wf}(P)$ holds. Notice that if a static (defined below) and well-formed process uses labels $(h_1,h_1),\ldots,(h_n,h_n)$, then $h_1,\ldots,h_n$ are all distinct.[1]

Definition 2.4 (Static Processes)**.** *Let a process P be* static *if the predicate below, inductively defined by the two rules and by homomorphic rules on the remaining process constructs, holds.*
$\mathsf{static}(\mathit{nil}) = \mathsf{true}$ *and* $\mathsf{static}((\mathit{newn} : (h,i))P) = (\mathsf{static}(P) \wedge h = i)$

So, in well-formed static processes no label pair occurs more than once – well-formedness implies that labels are used linearly. Therefore, if a process is well-formed, so are all its subprocesses.

Lemma 2.5 (Label freshness)**.** *Let* $\mathsf{wf}(P)$ *hold. Then,*

1. *if* $P = (\mathit{newn} : (h,i))Q$ *then* $i \notin \mathsf{secLabs}(Q)$*;*
2. *if* $P = (Q \parallel R)$ *then* $\mathsf{secLabs}(Q) \cap \mathsf{secLabs}(R) = \emptyset$*;*
3. $\mathsf{wf}(P\sigma)$*, being* σ *be a substitution of a name for a variable;*
4. *for any* $Q \in S$*, for some* $S \in \mathsf{subprocs}(P)$*, it holds that* $\mathsf{wf}(Q)$*;*
5. *for any Q and R such that* $\{Q,R\} \subseteq S$*, for some* $S \in \mathsf{subprocs}(P)$*, it holds that* $\mathsf{wf}(Q \parallel R)$*;*
6. *if* $\mathsf{wf}(Q)$ *and* $\mathsf{secLabs}(P) \cap \mathsf{secLabs}(Q) = \emptyset$ *then* $\mathsf{wf}(P \parallel Q)$*.*

Proof. Immediate, due to the definition of well-formed processes. □

Let π_1 denote the first pair projection function. The set S contains the labels to avoid when renaming the labels of the process.

[1] This result is an immediate consequence of Lemma A.2 in Page 7.

Static Process Syntax

$$
\begin{array}{llll}
P,Q,R \in \textsc{Proc} ::= \mathsf{nil} & \text{(inert)} & \mid (P \parallel Q) & \text{(composition)} \\
\mid u?x.P & \text{(input)} & \mid *u?x.P & \text{(replication)} \\
\mid u!v.P & \text{(output)} & \mid (\mathsf{new}\, n : (h,h))P & \text{(hiding)}
\end{array}
$$

Dynamic Process Syntax Let $i \in \mathbb{N}$.

$$P,Q,R \in \textsc{Proc} ::= \ldots \mid (\mathsf{new}\, n : (h,i))P \qquad \text{(hiding)}$$

Figure 1: The process language: syntax

Definition 2.6 (Process Relabelling)**.** *Let the (partial) binary function* relabelling*, taking a process and a set of labels and returning a process and a set of labels, be inductively defined by the rules below (the remaining cases being homomorphic). Consider* $S \subseteq \mathscr{H}$.

1. *if* $S \supseteq \{i\} \cup \mathsf{secLabs}(P)$ *and* $j \notin S$ *then*
   let $(P',S') = \mathit{relabelling}(P, S \cup \{j\})$ *in*

$$\mathit{relabelling}((\mathit{new}\, n : (h,i))P, S) = ((\mathit{new}\, n : (h,j))P', S')$$

2. *let* $(P',S') = \mathit{relabelling}(P,S)$ *and* $(Q',S'') = \mathit{relabelling}(Q,S')$ *in*

$$\mathit{relabelling}(P \parallel Q, S) = (P' \parallel Q', S'')$$

Note that the first label in label pairs is not affected by relabelling. The relevant results are that labels obtained by relabelling are fresh and relabelling preserves well-formedness. The proofs are in Appendix A.2.

Proposition 2.7 (Relabelling preserves label freshness)**.** *Let P be well-formed. Then, for any set of labels* $S \subseteq \mathscr{H}$ *such that* $S \supseteq \mathsf{secLabs}(P)$*,*

1. $\mathsf{relabelling}(P,S)$ *is defined;*
2. $\mathsf{secLabs}(P) \cap \mathsf{secLabs}(\pi_1(\mathsf{relabelling}(P,S))) = \emptyset$*;*
3. $\pi_1(\mathsf{relabelling}(P,S))$ *is well-formed.*

3 Reduction semantics

Considering, as usual, processes indistinguishable up to α-conversion, the operational semantics of the language is defined with two relations: *structural congruence* and *reduction*. Figure 2 presents the rules inductively defining both relations.

Note that labels, being constants, are not subject to α-conversion (naturally, only variables are). Labels are thus a mechanism to identify places where bound channels (variables) are used.

Structural Congruence

$$\textsc{sNil}\quad P \| \mathsf{nil} \equiv P \qquad \textsc{sCom}\quad P \| Q \equiv Q \| P \qquad \textsc{sAss}\quad P \| (Q \| R) \equiv (P \| Q) \| R$$

$$\textsc{sSwp}\quad (\mathsf{new}\, n : (h,i))(\mathsf{new}\, n' : (h',j))P \equiv (\mathsf{new}\, n' : (h',j))(\mathsf{new}\, n : (h,i))P$$

$$\textsc{sExt}\quad P \| (\mathsf{new}\, n : (h,i))Q \equiv (\mathsf{new}\, n : (h,i))(P \| Q) \text{ if } n \notin \mathbf{fn}(()P)$$

Reduction system

$$\textsc{Com}\ \frac{}{n!v.P \| n?x.Q \xrightarrow{n} P \| Q[^v/_x]} \qquad \textsc{Rep}\ \frac{Q' = \pi_1(\mathsf{relabelling}(Q, \mathsf{secLabs}(P \| Q)))}{n!v.P \| *n?x.Q \xrightarrow{*n} P \| Q[^v/_x] \| *n?x.Q'}$$

$$\textsc{Hid}\ \frac{P \xrightarrow{n} Q}{(\mathsf{new}\, n : (h,i))P \xrightarrow{(h,i)} (\mathsf{new}\, n : (h,i))Q} \qquad \textsc{HidRep}\ \frac{P \xrightarrow{*n} Q}{(\mathsf{new}\, n : (h,i))P \xrightarrow{*(h,i)} (\mathsf{new}\, n : (h,i))Q}$$

$$\textsc{Res}\ \frac{P \xrightarrow{o} Q \qquad o \neq n}{(\mathsf{new}\, n : (h,i))P \xrightarrow{o} (\mathsf{new}\, n : (h,i))Q} \qquad \textsc{Par}\ \frac{P \xrightarrow{o} Q \quad \mathsf{secLabs}(Q) \cap \mathsf{secLabs}(R) = \emptyset}{P \| R \xrightarrow{o} Q \| R}$$

$$\textsc{Str}\ \frac{P \equiv P' \qquad P' \xrightarrow{o} Q' \qquad Q' \equiv Q}{P \xrightarrow{o} Q}$$

Figure 2: The process language: operational semantics

Remark 3.1 (Notation)**.**

- *We write* $P \longrightarrow Q$ *in lieu of* $\exists o\,.\, P \xrightarrow{o} Q$.
- *Let* $\Longrightarrow$ *be the reflexive and transitive closure of* $\longrightarrow$.
- *For simplicity sake, in the examples and in some statements, we just write* $(\mathsf{new}\, n : h)P$*, instead of* $(\mathsf{new}\, n : (h,h))P$.

Notice that reduction preserves the first label in any label pair.

Lemma 3.2 (Label preservation)**.** *Let* $P \longrightarrow^* Q$*. For any* $(h,j) \in \mathsf{labelPairs}(Q)$ *there is an* i *such that* $(h,i) \in \mathsf{labelPairs}(P)$.

Proof. Straightforward. □

Relabelling at work. A simpler mechanism to generate fresh labels would be to increase the second label each time a new thread is spawned. The idea, however, does not guarantee label uniqueness.

Example 3.3 (Why increment doesn't work)**.**

$$\begin{aligned}
*a?.*b?.(\mathsf{new}\, n : (l,1))\mathit{nil} \| a!.\mathit{nil} \| a!.\mathit{nil} \| b!.\mathit{nil} &\longrightarrow \\
*b?.(\mathsf{new}\, n : (l,1))\mathit{nil} \| *a?.*b?.(\mathsf{new}\, n : (l,2))\mathit{nil} \| a!.\mathit{nil} \| b!.\mathit{nil} &\longrightarrow \\
*b?.(\mathsf{new}\, n : (l,1))\mathit{nil} \| *b?.(\mathsf{new}\, n : (l,2))\mathit{nil} \| *a?.*b?.(\mathsf{new}\, n : (l,3))\mathit{nil} \| b!.\mathit{nil} &\longrightarrow \\
(\mathsf{new}\, n : (l,1))\mathit{nil} \| *b?.(\mathsf{new}\, n : (l,2))\mathit{nil} \| *b?.(\mathsf{new}\, n : (l,2))\mathit{nil} \| *a?.*b?.(\mathsf{new}\, n : (l,3))\mathit{nil}
\end{aligned}$$

□

The relabelling mechanism defined actually guarantees that label uniqueness is preserved by reduction. An elaborate example is below.

Example 3.4 (Relabelling works)**.** *Consider*

$$P = a!.(newn : (l_1, l_1))nil \| Q_0 \qquad \text{with}$$
$$Q_0 = *a?.Q_{00} \qquad \text{and} \quad Q_{00} = *b?.(newn : (l_0, l_0))nil$$

By rule REP*, we have* $P \longrightarrow Q$*, where*

$$Q = (newn : (l_1, l_1))nil \| Q_{00} \| *a?.\mathsf{relabel}(Q_{00})$$

and $\mathsf{relabel}(Q_{00}) = *b?.(newn : (l_0, l_4))nil$ *with a fresh label* l_4*. Notice how*

$$\big(\mathsf{secLabs}((newn : (l_1, l_1))nil \| Q_0) = \{l_0, l_1\}\big) \cap \big(\{l_4\} = \mathsf{secLabs}(\mathsf{relabel}(Q_{00}))\big) = \emptyset$$

Consider now

$$R = a!.(newn : (l_2, l_2))nil \| b!.(newn : (l_3, l_3)nil)$$

Since $\mathsf{secLabs}(Q) = \{l_0, l_1, l_4\}$ *and* $\mathsf{secLabs}(R) = \{l_2, l_3\}$ *(they are disjoint), we conclude, by rule* PAR*,*

$$P \| R = \begin{cases} a!.(newn : (l_1, l_1))nil \| *a?.*b?.(newn : (l_0, l_0))nil \\ \| \, a!.(newn : (l_2, l_2))nil \| \, b!.(newn : (l_3, l_3))nil \end{cases}$$
$$\longrightarrow$$
$$Q \| R = \begin{cases} (newn : (l_1, l_1)nil) \| *b?.(newn : (l_0, l_0))nil \| *a?.*b?.(newn : (l_0, l_4))nil \\ \| \, a!.(newn : (l_2, l_2))nil \| \, b!.(newn : (l_3, l_3))nil \end{cases}$$

The same reasoning applies now for the subsequent reduction step:

- *Assuming* $\mathsf{relabel}(*b?.(newn : (l_0, l_4))nil) = *b?.(newn : (l_0, l_5))nil$ *with* l_5 *fresh, by rule* REP *we have:*

$$*a?.*b?.(newn : (l_0, l_4))nil \| a!.(newn : (l_2, l_2))nil \quad \longrightarrow$$
$$*b?.(newn : (l_0, l_4))nil \| *a?.*b?.(newn : (l_0, l_5))nil \| (newn : (l_2, l_2))nil$$

- *thus, by rule* PAR*,* $P' = Q \| a!.(newn : (l_2, l_2))nil \longrightarrow Q'$*, where*

$$Q' = \begin{cases} (newn : (l_1, l_1))nil \| (newn : (l_2, l_2))nil \| *b?.(newn : (l_0, l_0))nil \\ \| *b?.(newn : (l_0, l_4))nil \| *a?.*b?.(newn : (l_0, l_5))nil \end{cases}$$

So, $Q \| R \longrightarrow Q' \| b!.(newn : (l_3, l_3))nil$*, and again, reasoning as above, we get*

$$Q' \| b!.(newn : (l_3, l_3))nil \quad \longrightarrow$$
$$\begin{cases} (newn : (l_1, l_1))nil \| (newn : (l_2, l_2))nil \| (newn : (l_3, l_3))nil \\ \| *b?.(newn : (l_0, l_6))nil \| *b?.(newn : (l_0, l_4))nil \| *a?.*b?.(newn : (l_0, l_5))nil \end{cases}$$

Notice that all labelled pairs are different. □

4 Label uniqueness.

A crucial property of our language is that the uniqueness of labels is preserved by reduction. The precision of our deadlock detection analysis relies on this fact.

An example of relabelling at work is in Appendix 3. From it it is simple to understand why using only one label (or an indexing mechanism) would not work.

Preservation of label uniqueness by reduction. A key property to ensure the soundness of our deadlock detection algorithm is the preservation of well-formedness by reduction. The proof is in Appendix A.3.

Lemma 4.1. *If P is well-formed and $P \longrightarrow Q$ then Q is well-formed.*

5 A standard reduction semantics.

Notice that, for well-formed processes, our semantics coincides with a standard one. To state this property, consider the auxiliary function $\mathsf{labErasure}$ on processes that removes the label pairs from the hiding constructor (hence producing standard π-calculus processes). The function is inductively defined by homomorphic rules on all process constructs but on hiding, where the function is defined by the following rule:

$$\mathsf{labErasure}((\mathsf{new}n : (h,i))P) = (\mathsf{new}n)\mathsf{labErasure}(P)$$

The usual relation $\rightarrow$ on standard processes is obtained by removing the side condition $\mathsf{secLabs}(Q) \cap \mathsf{secLabs}(R) = \emptyset$ from rule REP in Figure 2 and by replacing rule REP with the following axiom:

$$n!v.P \,\|\, *n?x.Q \xrightarrow{*n} P \,\|\, Q[v/x] \,\|\, *n?x.Q$$

Obviously, $\mathsf{labErasure}(P) \rightarrow \mathsf{labErasure}(Q)$, if $P \longrightarrow Q$. The opposite direction does not work only due to the side condition of the PAR rule.

6 Conclusions

We devised a simple mechanism to uniquely identify scoped names in the π-calculus. This approach is useful to support the analysis of properties of scoped names, an example being identify which ones are leaked.

References

[1] Henk Barendregt (1981 (1st ed.), revised 1984): *The Lambda Calculus - Its Syntax and Semantics*. North-Holland.

[2] Robin Milner (1999): *Communicating and mobile systems - the Pi-calculus*. Cambridge University Press.

[3] Robin Milner, Joachim Parrow & David Walker (1992): *A Calculus of Mobile Processes, parts I and II*. *Information and Computation* 100(1), pp. 1–77.

[4] Davide Sangiorgi & David Walker (2001): *The Pi-Calculus: A Theory of Mobile Processes*. Cambridge University Press.

A On ensuring label uniqueness

A.1 No label clashes

Consider the following function, inductively defined by the given rules.

nLabels $\mathsf{nLabels}(\mathsf{nil}) = 0$, $\mathsf{nLabels}(P \,\|\, Q) = \mathsf{nLabels}(P) + \mathsf{nLabels}(Q)$,
$\mathsf{nLabels}(u?x.P) = \mathsf{nLabels}(*u?x.P) = \mathsf{nLabels}(u!v.P) = \mathsf{nLabels}(P)$, and
$\mathsf{nLabels}((\mathsf{new}\, n : (h,h'))P) = 1 + \mathsf{nLabels}(P)$

Obviously, $\#\mathsf{labelPairs}(P) \leq \mathsf{nLabels}(P)$.

Let σ be a substitution of a name for a variable. One easily sees that the sets labels and nLabels are preserved by substitutions and by alpha-congruence on names and variables (*i.e.*, labels are like constants). Moreover, both sets might increase with reduction (labels are never removed).

Lemma A.1 (Reduction preserves labels)**.**

$$\mathit{nLabels}(P) = \mathit{nLabels}(P\sigma) \tag{1}$$
$$\mathsf{labelPairs}(P) = \mathsf{labelPairs}(P\sigma) \tag{2}$$

$$(P \equiv_\alpha Q) \Rightarrow (\mathit{nLabels}(P) = \mathit{nLabels}(Q) \wedge \mathsf{labelPairs}(P) = \mathsf{labelPairs}(Q)) \tag{3}$$
$$(P \longrightarrow Q) \Rightarrow (\mathit{nLabels}(P) \subseteq \mathsf{labelPairs}(Q) \wedge \mathsf{labelPairs}(P) \subseteq \mathsf{labelPairs}(Q)) \tag{4}$$

Proof. Immediate. □

Consider the following predicate, stating that all pairs of labels in a given process are different.

$$\mathsf{noLabelClashes}(P) = (\#\mathsf{labelPairs}(P) = \mathsf{nLabels}(P))$$

The predicate above provides an alternative characterisation of well-formedness.

Lemma A.2 (No label clashes)**.** $\mathsf{wf}(P)$ *if and only if* $\mathit{noLabelClashes}(P)$

Proof. Immediate, due to the definition of well-formed processes. □

A.2 Relabelling

Let π_2 denote the second pair projection functions.

Lemma A.3 (Monotonicity)**.** *If P is well-formed and $S' = \pi_2(\mathsf{relabelling}(P,S))$ then $S \subseteq S'$.*

Proof. Immediate, due to the definition of well-formed processes. □

Lemma A.4 (Relabelling preserves label freshness)**.** *Let P be well-formed and consider a set of labels $S \supseteq \mathsf{secLabs}(P)$. Then, the following results hold.*

$$\mathit{nLabels}(\pi_1(\mathit{relabelling}(P,S))) = \mathit{nLabels}(P) \tag{5}$$
$$S \cap \mathsf{secLabs}(\pi_1(\mathsf{relabelling}(P,S))) = \emptyset \tag{6}$$

Proof. The proofs are by structural induction on P. The first equation is straightforward to prove – it ensures that rellabelling preserves the number of labels.

In the proof of the second equation, two cases matter. Let first $P = (\mathsf{new}\, n : (h,i))Q$ Since by hypothesis $\mathsf{wf}(P)$, Lemma 2.5.2 ensures $i \notin \mathsf{secLabs}(Q)$. Take $j \neq i$ such that $j \notin \mathsf{secLabs}(Q)$. Then, as $j \notin (\{i\} \cup \mathsf{secLabs}(Q)) = \mathsf{secLabs}(P)$, taking a set $S \supseteq \{i\} \cup \mathsf{secLabs}(Q)$ where $j \notin S$, the function relabelling gives the following result.

let $(Q', S') = \mathsf{relabelling}(Q, S \cup \{j\})$ in

$$\mathsf{relabelling}(P,S) = ((\mathsf{new}\, n : (h,j))Q', S') \ .$$

So, as $S \supseteq \mathsf{secLabs}(P)$, we have

$$\mathsf{secLabs}(\pi_1(\mathsf{relabelling}(P, S \cup \{j\}))) = \{j\} \cup \mathsf{secLabs}(Q') \ .$$

Since $\mathsf{wf}(Q)$ by Lemma 2.5.4, by induction hypothesis,

$$S \cup \{j\} \cap \mathsf{secLabs}(\pi_1(\mathsf{relabelling}(Q, S \cup \{j\}))) = \emptyset,$$

so, as $S \supseteq \{i\} \cup \mathsf{secLabs}(Q)$ and $Q' = \pi_1(\mathsf{relabelling}(Q, S \cup \{j\}))$ and furthermore $j \neq i$, we conclude $S \cap (\{j\} \cup \mathsf{secLabs}(Q')) = \emptyset$ as required.

Consider now $P = (Q \| R)$. Since by hypothesis $\mathsf{wf}(P)$, Lemma 2.5.2 ensures $\mathsf{secLabs}(Q) \cap \mathsf{secLabs}(R) = \emptyset$. As both $\mathsf{wf}(Q)$ and $\mathsf{wf}(R)$ by Lemma 2.5.4, by induction hypothesis, we have $S \cap \mathsf{secLabs}(Q') = \emptyset$ and $S' \cap \mathsf{secLabs}(R') = \emptyset$ where $(Q', S') = \mathsf{relabelling}(Q, S)$ and $R' = \pi_1(\mathsf{relabelling}(R, S'))$. So, since $S \subseteq S'$ by Lemma A.3, we conclude

$$\begin{aligned} S \cap \mathsf{secLabs}(Q' \| R') &= \\ S \cap (\mathsf{secLabs}(Q') \cup \mathsf{secLabs}(R')) &= \\ (S \cap \mathsf{secLabs}(Q')) \cup (S \cap \mathsf{secLabs}(R')) &= \\ \emptyset \cup \emptyset &= \emptyset \end{aligned}$$

as required. □

Lemma A.5 (Relabelling preserves well-formedness)**.** *Let P be well-formed and consider a set of labels $S \supseteq \mathsf{secLabs}(P)$. Then, $\pi_1(\mathsf{relabelling}(P,S))$ is well-formed.*

Proof. The proof is by structural induction on P. All homomorphic cases in the definition of relabelling are either straightforward or following by the induction hypothesis, using Lemma 2.5.4. So, two cases matter. Let first $P = (\mathsf{new}\, n : (h,i))Q$. As P is well-formed, so is Q (again, by the previous lemma). By definition,

$$\mathsf{relabelling}((\mathsf{new}\, n : (h,i))Q, S) = ((\mathsf{new}\, n : (h,j))Q', S')$$

where $(Q', S') = \mathsf{relabelling}(Q, S \cup \{j\})$, considering $i \in S$ and $j \notin (S \cup \mathsf{secLabs}(P))$. By induction hypothesis, Q' is well-formed. Since by hypothesis, $S \supseteq \mathsf{secLabs}(P)$, obviously $j \notin S$ and $i \neq j$, so $(\mathsf{new}\, n : (h,j))Q'$ is also well-formed.

Consider now $P = (Q \| R)$. As P is well-formed, by the same lemma, so are Q and R. By definition,

$$\mathsf{relabelling}(Q \| R, S) = (Q' \| R', S'')$$

where $(Q', S') = \mathsf{relabelling}(Q, S)$ and $(R', S'') = \mathsf{relabelling}(R, S')$. Since by hypothesis $S \supseteq \mathsf{secLabs}(P)$, obviously $S \supseteq \mathsf{secLabs}(Q)$, so by induction hypothesis, Q' is well-formed. It is also the case that $S \supseteq \mathsf{secLabs}(R)$, and since by Lemma A.3, $S' \supseteq S$, by induction hypothesis, R' is also well-formed. Since Lemma A.4.6 ensures that $S \cap \mathsf{secLabs}(Q') = \emptyset$ and $S' \cap \mathsf{secLabs}(R') = \emptyset$, we have $\mathsf{secLabs}(Q') \cap \mathsf{secLabs}(R') = \emptyset$, thus by Lemma 2.5.6 we conclude that $Q' \parallel R'$ is well-formed. □

A.3 Reduction preserves label uniqueness

Lemma A.6. *If P is well-formed and $P \longrightarrow Q$ then Q is well-formed.*

Proof. Notice first that structural congruence preserves label uniqueness, as no relabelling happens. To prove that well-formedness is preserved by reduction, we proceed by induction of the derivation of $P \longrightarrow Q$.

Base cases. The only base case that changes the labels is the REP rule:

$$\textsc{Rep}\frac{P_2' = \pi_1(\mathsf{relabelling}(P_2, \mathsf{secLabs}(P_1)))}{n!v.P_1 \parallel *n?x.P_2 \xrightarrow{*n} P_1 \parallel P_2[v/x] \parallel *n?x.P_2'}$$

As $P = n!v.P_1 \parallel *n?x.P_2$, let $P' = P_1 \parallel P_2[v/x]$. By hypothesis P is well-formed, thus by Lemmas 2.5.4, 2.5.3 2.5.5, and A.4.3, both P' and Q'' are well-formed. Moreover, by definition of well-formedness, $*c?x.Q''$ is also well-formed.

Since by Lemmas 2.5.2 and A.4.2 we conclude that $\mathsf{secLabs}(P' \parallel Q'[v/x]) \cap \mathsf{secLabs}(Q'') = \emptyset$, and obviously $\mathsf{secLabs}(Q'') = \mathsf{secLabs}(*c?x.Q'')$, we attain the result using Lemma 2.5.6.

Inductive steps. The only relevant case is the PAR rule.

Let $P = P_1 \parallel P_2$ and $Q = P_1' \parallel P_2$. By hypothesis,

$$P_1 \longrightarrow P_1' \text{ and } \mathsf{secLabs}(P_1') \cap \mathsf{secLabs}(P_2) = \emptyset$$

Since by induction hypothesis, P_1' is well-formed, we attain the result – Q is well-formed – using again Lemma 2.5.6.

□