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. 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, one often needs to refer to particular occurrences of a bound identifier to pinpoint, for instance, program defects. Thus, program analysis methods like those looking for deadlocks, need to inspect bound names. Moreover, to provide meaningful explanations to the reasons of a deadlock, it might be necessary to refer to occurrences of bound identifiers, 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 a pair of unique labels that are treated like constants. We present the syntax and the operational semantics of the calculus, and show that as new names are created, the generated labels associated with them are unique. Therefore, one can refer particular instances of bound names, even if their identifiers occur more than once in a process, free or bound.

In short, our contribution is the following: a syntactic mechanism, simple to automatise, that generates unique identifiers associated with each new name. The uniqueness of these identifiers is preserved by reduction, the usual operational semantics mechanism of the calculus. This mechanism is useful 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. The occurrences in process P of the variable x in process like $u?x.P$ and $*u?x.P$, and n in $(\mathsf{new}\, n : (h,h))P$, are *bound*. As usual, other occurrences are *free*. Let $\mathbf{fn}(P)$ denote the set of free names in process P. Moreover, we use the standard notion of capturing avoiding substitution of a free variable x by a name v in a process P (denoted by $P[^v/_x]$).

Submitted to:
DCM 2018

Furthermore, labels h, i, j range over natural numbers. In the "top-level" (i.e., the "programmers") process syntax — that we call *static* — the labels in a pair are equal; reduction produces dynamic processes, where the pairs of labels become different.

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

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.*

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

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* $\mathscr{P} \subseteq \mathsf{subprocs}(P)$ *such that* $\{(\mathit{newn}:(h,i))Q, (\mathit{newn}:(h',j))Q'\} \subseteq \mathscr{P}$ *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 $(h_1,h_1),\dots,(h_n,h_n)$, then $h_1,\dots,h_n$ 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* $P = (Q \,\|\, R)$ *then* $\mathsf{secLabs}(Q) \cap \mathsf{secLabs}(R) = \emptyset$*;*
3. $\mathsf{wf}(P[^v/_x])$*;*
4. *for any* $Q \in \mathsf{subprocs}(P)$*, it holds that* $\mathsf{wf}(Q)$*;*
5. *for any Q and R such that* $\{Q,R\} \subseteq \mathsf{subprocs}(P)$*, it holds that* $\mathsf{wf}(Q \,\|\, R)$*;*
6. *if* $\mathsf{wf}(Q)$ *and* $\mathsf{secLabs}(P) \cap \mathsf{secLabs}(Q) = \emptyset$ *then* $\mathsf{wf}(P \,\|\, Q)$.

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

Given the definition of well-formed processes, the proofs of the results above are straightforward.

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

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 a set of labels S.*

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

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

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 and S a set of labels such that* $S \supseteq \mathsf{secLabs}(P)$*. Then,*

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.

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

Lemma 3.1 (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. □

Static Process Syntax

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

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

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

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

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

$$\textsc{Str}\,\frac{P \equiv P' \qquad P' \longrightarrow Q' \qquad Q' \equiv Q}{P \longrightarrow Q}$$

Figure 2: The process language: operational semantics

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 3.2 (Why increment doesn't work)**.**

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

□

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

Example 3.3 (Relabelling works)**.** *Consider*

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

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

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

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

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

Consider now

$$R = a!.(\mathit{newn}:(l_2,l_2))\mathit{nil} \parallel b!.(\mathit{newn}:(l_3,l_3)\mathit{nil})$$

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*,*

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

The same reasoning applies now for the subsequent reduction step:

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

$$\begin{aligned} &*a?.*b?.(\mathit{newn}:(l_0,l_4))\mathit{nil} \parallel a!.(\mathit{newn}:(l_2,l_2))\mathit{nil} \quad \longrightarrow \\ &*b?.(\mathit{newn}:(l_0,l_4))\mathit{nil} \parallel *a?.*b?.(\mathit{newn}:(l_0,l_5))\mathit{nil} \parallel (\mathit{newn}:(l_2,l_2))\mathit{nil} \end{aligned}$$

- *thus, by rule* Par*,* $P' = Q \parallel a!.(\mathit{newn}:(l_2,l_2))\mathit{nil} \longrightarrow Q'$*, where*

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

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

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

Notice that all labelled pairs are different. □

Preservation of label uniqueness by reduction. A crucial property of our language is that the uniqueness of labels is preserved by reduction. The precision of defect detection analysis might rely on this fact. The proof of this fact (stated below) is in Appendix A.3.

Lemma 3.4. *If* P *is well-formed and* $P \longrightarrow Q$ *then* Q *is well-formed.*

4 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 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 from rule PAR and by replacing rule REP with the axiom

$$n!v.P \parallel *n?x.Q \longrightarrow P \parallel Q[v/x] \parallel *n?x.Q$$

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.

5 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, examples being the identification of which ones are leaked or deadlocked.

We implemented process relabeling (c.f. Definition 2.6 and Figure 2) in OCaml by using a random generator to replace a positive integer i in a new under replication with a fresh positive integer j, and by throwing a clash exception in rule Par whenever the calculation of the redex of the hypothesis introduces a label that is used by the parallel process.

Acknowledgements

This work was partially supported by COST Actions BETTY (IC1201) and ARVI (IC1401), by the EPSRC programme grant ABCD (EP/K034413/1), and by NOVA LINCS grant UID/CEC/04516/2013.

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)**.** *Let σ be a substitution $[{}^{v}\!/\!{}_{x}]$.*

$$\begin{aligned} \mathit{nLabels}(P) &= \mathit{nLabels}(P\sigma) & (1)\\ \mathsf{labelPairs}(P) &= \mathsf{labelPairs}(P\sigma) & (2) \end{aligned}$$

$$\begin{aligned} (P \equiv_\alpha Q) &\Rightarrow (\mathit{nLabels}(P) = \mathit{nLabels}(Q) \land \mathsf{labelPairs}(P) = \mathsf{labelPairs}(Q)) & (3)\\ (P \longrightarrow Q) &\Rightarrow (\mathit{nLabels}(P) \subseteq \mathsf{labelPairs}(Q) \land \mathsf{labelPairs}(P) \subseteq \mathsf{labelPairs}(Q)) & (4) \end{aligned}$$

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 $S \supseteq \mathsf{secLabs}(P)$. Then, making $(Q,R) = \mathsf{relabelling}(P,S)$, the following results hold.*

$$\begin{aligned} \mathit{nLabels}(Q) &= \mathit{nLabels}(P) & (5)\\ S \cap \mathsf{secLabs}(Q) &= \emptyset & (6)\\ R &\subseteq \mathsf{secLabs}(Q) & (7) \end{aligned}$$

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 2.5.1 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 $P = (Q \,\|\, R)$. As by hypothesis $\mathsf{wf}(P)$, by Lemma 2.5.2, $\mathsf{secLabs}(Q) \cap \mathsf{secLabs}(R) = \emptyset$. 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 $S' \cap \mathsf{secLabs}(R') = \emptyset$, and Lemma A.4.7 ensures that $S' \supseteq \mathsf{secLabs}(Q')$, 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:

$$\textsc{Rep}\,\frac{Q' = \pi_1(\mathsf{relabelling}(Q_1, \mathsf{secLabs}(P_1 \parallel Q_1[{}^{v}\!/\!_{x}])))}{P = (n!v.P_1 \parallel *n?x.Q_1) \longrightarrow (P_1 \parallel Q_1[{}^{v}\!/\!_{x}] \parallel *n?x.Q') = Q}$$

By hypothesis P is well-formed, thus:

- by Lemma 2.5.4, both P_1 and Q_1 are well-formed;
- as Q_1 is well-formed, by Lemma 2.5.3 also $Q_1[{}^{v}\!/\!_{x}]$ is well-formed;
- hence, by Lemma 2.5.6, also $P_1 \parallel Q_1[{}^{v}\!/\!_{x}]$ is well-formed.

Then, by Lemma A.5, Q' is well-formed as well.

Moreover, by definition of well-formedness, $*n?x.Q'$ is also well-formed.

Since by Lemmas 2.5.2 and 2.7.2 we conclude that $\mathsf{secLabs}(P_1 \parallel Q_1[{}^{v}\!/\!_{x}]) \cap \mathsf{secLabs}(Q') = \emptyset$, and obviously $\mathsf{secLabs}(Q') = \mathsf{secLabs}(*n?x.Q')$, we attain the result using again 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.

□