Chapter 3

Specifying Mobile, Stateful Protocol Behavior

In this chapter, we investigate formal modeling of mobile systems with stateful channels. We propose a principled extension to the π-calculus, namely πZ that integrates the π-calculus mobile channels with stateful abstract types of the Z language to form a novel stateful mobile channel construct. In Section 3.1 we describe the syntax of πZ along with two examples, including modeling of the hidden node problem in wireless networks.

In Section 3.2 we propose reduction and labelled transition semantics for πZ and establish the correspondence between the two styles of semantics. Labelled transition semantics uses the novel notion of action obligations to prevent internal state information from appearing in the transition labels. In Section we study bisimulation as the behavioral equivalence of πZ processes and establish that the bisimulation is closed under important contexts. In Section 3.4 we establish that data refinement induces process simulation.

3.1 πZ syntax

We illustrate the πZ syntax with the help of two examples. The first example is that of a mobile buffered channel and is intended to illustrate the basic πZ syntax. The second is a more elaborate ‘real-world’ example demonstrating the use of mobile stateful channels to model the hidden-node problem occurring in wireless networks.
Processes in \( \pi Z \) are closed terms, i.e., terms containing no free variables, as per the syntax given in Figure 3.1. We keep names and variables separate, as followed by many \( \pi \)-calculus extensions [42, 2]. Terms are identified up to \( \alpha \)-conversion as usual. The syntax of a \( \pi Z \) process extends the \( \pi \)-calculus process syntax in the following ways: (1) input and output actions have associated operation names; (2) stateful \( \pi Z \) channels have associated state-holding terms, whose state structure and operations are defined by abstract type definitions. The output action supplies parameters for an operation; the input action receives results from the operation; the state-holding term holds the current state. The parameters supplied by an output action and the current state (before-state) are used to decide the result values received by the input action and the modified state (after-state) to be held by the state-holding term. In this manner, an abstract type defines a stateful channel.

We use the following naming convention: \( P, Q, R \) range over processes; \( x, y, z \) range over variables; \( V, W \) range over labelled tuples; \( a, b, c, m, n \) range over names; \( u, v, w \) range over values (names, variables, and base values); \( p, q \) range over operation names; and \( A, B, D, T \) range over datatypes. We use \( \bar{x}, \bar{v}, \) and \( \bar{n} \) to mean tuples of variables, values, and names, respectively. Names are identified with channels. Note that, unlike other typed \( \pi \)-calculi, type annotation in \( \pi Z \) is given to the state-holding term rather than the \( \nu \) construct. Variables are not allowed to occur in state-holding terms since variable substitutions may lead to violation of state invariants. Channel names can, however, appear in the state-holding
Figure 3.2 shows a modified version of the bounded buffer abstract type from Figure 2.5. Here, channel mobility is facilitated since we may pass the names of other buffered channels. Channel types (i.e., types of channel names) are introduced as basic types using a minor extension to the Z syntax, viz., the type operator \[ \mathcal{T} \]. Given an abstract type \( T \), the channel type corresponding to \( T \) is denoted by \( \mathcal{T} \). That is, \( \mathcal{T} \) is the name of a channel type whose state structure and operations are defined by abstract type \( T \). From the Z viewpoint, \( \mathcal{T} \) is simply a basic type. Being a basic type, \( \mathcal{T} \) possesses no internal structure, and operations can only store (unmodified) and compare channel names. Fresh names cannot be generated from within abstract type operations; they can only be generated using the ‘\( \nu \)’ construct from within \( \pi Z \) processes.

In Figure 3.2, \( \mathcal{B} \) is the channel type corresponding to abstract type \( \mathcal{B} \). Since elements of type \( \mathcal{D} \) do not depend on the state structure of the abstract type \( D \), we allow \( \mathcal{D} \) to be used from within type \( D \)'s state schema definition. Example \( \pi Z \) processes that use the buffered channel abstract type are shown below.

1. \( \overline{q} \cdot \text{put}(3, p).P | q(\text{items} = \langle \rangle, \text{size} = 100) : \mathcal{B} \cdot q \cdot \text{put}.Q \)

2. \( \overline{q} \cdot \text{get}.P | q(\text{items} = \langle(3, p)\rangle, \text{size} = 100) : \mathcal{B} \cdot q \cdot \text{get}(x, y).\overline{q} \cdot \text{put}(x, q).Q \)

The example shows how the initial state for channel \( q \) is specified using the state-holding term as a part of the process definition, instead of using an initialization schema as a part of the abstract type specification. In order to complete an operation on a stateful channel, three concurrent subterms must synchronize – the subterm executing output action, the subterm executing input action, and the state-holding term. Item (1) above shows a process ready to perform a \( \text{put} \) operation on channel \( q \). The first subterm is ready to execute an output action, supplying values \( (3, p) \) as the operation’s parameters; the second subterm holds the channel state, with \( \text{items} \) set to an empty sequence; the third subterm is ready to execute an input action on the channel. Since the \( \text{put} \) operation has no result values, the input action does not receive any values.

\(^1\)The same symbol is used in the Z language for representing a binary operator on bags. We may distinguish the two based on the usage context.
\[
\text{BufferedChannel}[D] \quad \text{---------}
\]
\[
\text{items} : \text{seq} (D \times \#\text{BufferedChannel}[D]) \\
\text{size}: N \\
\]
\[
\#\text{items} \leq \text{size}
\]

\[
\text{put}[D] \\
\Delta \text{BufferedChannel}[D] \\
\text{msg}?: D \\
\text{ack}?: \#\text{BufferedChannel}[D]
\]
\[
\text{items}' = \text{items} \setminus \langle (\text{msg}?, \text{ack}?) \rangle
\]

\[
\text{get}[D] \\
\Delta \text{BufferedChannel}[D] \\
\text{msg}!: D \\
\text{ack}!: \#\text{BufferedChannel}[D]
\]
\[
\text{items} = \langle (\text{msg}!, \text{ack}!) \rangle \setminus \text{items}'
\]

\[
\text{BufferedChannel}_{\mathbb{N}} = \equiv \text{BufferedChannel}[\mathbb{N}]
\]

Figure 3.2: \(\pi Z\) abstract type definition of a bounded-buffered channel, allowing channel names to be passed in and out of the buffered channel.
Although the unnecessary appearance of the term executing the input action may appear awkward, in the general case the input action may actually consume values for further processing (see the specification of the hidden node problem in the next subsection). Moreover, it is possible to add higher-level constructs that translates to the basic $\pi Z$ syntax, which avoids the unnecessary appearance of terms executing empty input or output actions.

Item (2) above shows a process ready to perform a $get$ operation. Note that channel mobility is illustrated here; the retrieved channel name in bound variable $y$ is used to echo back the retrieved value to the sender by invoking its $put$ operation.

### 3.1.1 $\pi Z$ specification of the hidden node problem

![Diagram of the hidden node problem](image)

Figure 3.3: The hidden node problem in a 3-node wireless network. Diagram at the top shows wireless coverage areas of the three nodes. The bottom diagram shows three cases of signal sequencing.

In order to further illustrate the use of stateful channels in modeling durable communication activities, we consider modeling a 3-node wireless network to demonstrate the hidden node problem and its solution using RTS-CTS exchange [68], depicted in Figure 3.3. The wireless network we wish to model consists of a collection of stations (nodes $a$ and $b$, in our
Figure 3.4: πZ abstract type definition of a wireless channel – State schema and free type definitions.

example) communicating via an access point (the node ap, in our example). Two stations do not communicate with each other directly. The coverage diagram in Figure 3.3 shows that nodes a and b are mutually hidden from each other. Three cases of signal sequencing are shown in Figure 3.3. In Case A there is no hidden node problem – station a transmits a frame, which is received by ap as well as station b. This causes b to sense the carrier and wait till the transmission is complete. In Case B, a’s frame is not seen by b – this causes b to start transmitting a frame while a’s transmission is in progress, resulting in collision and loss of both data frames. In Case C, the hidden node problem is solved by the network using an RTS-CTS (request-to-send/clear-to-send) exchange – this assumes that by the time a requesting node sees the CTS response and begins transmission of its frame, all other nodes would have seen the CTS response and hence will be quiet.

Figures 3.4 and 3.5 together give an abstract type definition for a wireless channel, and Figure 3.6 gives process definitions for wireless network models, borrowing the syntax for process definition and application from [80] for clarity. Wherever a process application appears, a syntactic replacement (along with parameter substitution) of the corresponding process definition may be performed in order to obtain πZ processes as per syntax given in Figure 3.1. For formatting simplicity, we have reused set operator symbols such as ‘∈’ and ‘\’ in place of the special Z bag operator symbols.

The WirelessChannel type represents RF interfaces of wireless nodes. Since RF interfaces are the means of communication in a wireless network, we model them as πZ channels. The following state components are defined for the WirelessChannel type (Figure 3.4): (a) carrier: represents the state of wireless carrier, i.e., busy, idle, or experiencing collision; (b) active:
\begin{align*}
\text{start\_frame} & \quad \Delta \text{WirelessChannel} \\
& \quad \text{src}?: \sharp \text{WirelessChannel} \\
& \quad \text{coverage}?: \text{bag } \sharp \text{WirelessChannel} \\
& \quad \text{carrier} = \text{BUSY} \implies \text{src} \notin \text{coverage} \\
& \quad \text{coverage}^' = \text{coverage} \uplus \text{coverage}? \land \text{active}^' = \text{active} \uplus \{\text{src}\} \\
& \quad \text{if carrier} \neq \text{IDLE} \text{ then carrier}^' = \text{COLLISION} \text{ else carrier}^' = \text{BUSY} \\
\end{align*}

\begin{align*}
\text{end\_frame} & \quad \Delta \text{WirelessChannel} \\
& \quad \text{src}?: \sharp \text{WirelessChannel} \\
& \quad \text{coverage}?: \text{bag } \sharp \text{WirelessChannel} \\
& \quad \text{src}!, \sharp \text{WirelessChannel} \\
& \quad \text{frame}!, \text{Frame} \\
& \quad \text{src}? \in \text{active} \\
& \quad \text{if carrier} = \text{COLLISION} \text{ then frame}! = \text{BAD} \text{ else frame}! = \text{DATA} \\
& \quad \text{if active} = \{\text{src}\} \text{ then carrier}^' = \text{IDLE} \text{ else carrier}^' = \text{carrier} \\
& \quad \text{coverage}^' = \text{coverage} \setminus \text{coverage}? \land \text{active}^' = \text{active} \setminus \{\text{src}\} \land \text{src}! = \text{src}? \\
\end{align*}

\begin{align*}
\text{short\_frame} & \quad \Delta \text{WirelessChannel} \\
& \quad \text{src}?: \sharp \text{WirelessChannel} \\
& \quad \text{dst}?: \sharp \text{WirelessChannel} \\
& \quad \text{frame}? : \text{Frame} \\
& \quad \text{src}!, \sharp \text{WirelessChannel} \\
& \quad \text{dst}!, \sharp \text{WirelessChannel} \\
& \quad \text{frame}!, \text{Frame} \\
& \quad \text{frame}? \in \{\text{RTS, CTS}\} \\
& \quad \text{carrier} = \text{BUSY} \implies \text{src} \notin \text{coverage} \\
& \quad \text{coverage}^' = \text{coverage} \land \text{active}^' = \text{active} \\
& \quad \text{if carrier} \neq \text{IDLE} \text{ then carrier}^' = \text{COLLISION} \land \text{frame}! = \text{BAD} \\
& \quad \text{else frame}! = \text{frame}? \land \text{carrier}^' = \text{carrier} \land \text{src}! = \text{src}? \land \text{dst}! = \text{dst}? \\
\end{align*}

Figure 3.5: \(\pi Z\) abstract type definition of a wireless channel – Operation schema definitions.
the set of RF interfaces currently actively transmitting on the channel; (c) coverage: the collection (modelled as a bag) of RF interfaces covered by the signal being transmitted by active stations.

The nodes themselves are modeled as concurrent processes. A node may transmit two kinds of frames over a wireless channel – normal data frame and short frame. Transmitting a data frame is a durable activity. When a node wants to begin transmitting a data frame to the access point it invokes an output action for the start_frame operation on the ap channel. Even though the frame is being sent to the access point, the RF signal will reach other nodes as well. To model this aspect, we must specify the signal coverage, i.e., the collection (actually, a bag) of RF interfaces covered by the signal when invoking the start_frame operation. Completion of the start_frame operation puts the ap channel in busy state. The carrier-sense behavior is incorporated in the start_frame operation as follows: if the start_frame operation is attempted when the channel is in a busy state and the sender is covered by the RF signal (i.e., src ∈ coverage), then the sender senses the carrier busy and blocks. However, if the sender is not covered by the signal, we have the hidden node problem and collision occurs.
sense and subsequent transmission is modeled as an atomic operation; in reality, there is a very small probability that two stations may simultaneously perform carrier-sense, resulting in collision – this is not captured by the model. The end_frame operation removes the station’s RF interface from active set and also removes its coverage from the channel state. Transmission of a short frame is achieved by invoking the short_frame operation which is assumed to complete instantaneously.

In our specification, we first demonstrate a wireless network with no hidden nodes using the πZ process definition WNet. Here, the two stations simply transmit one frame and then stop; the access point simply logs all frames received by invoking an operation on the log channel, which is left unspecified. The WNext_h process demonstrates the hidden node problem. In the case of WNet, the condition log.frame(BAD) never occurs, whereas this condition occurs in the case of WNet_h. The WNet process models the use of RTS-CTS handshake to avoid the hidden node problem. When a node wants to send a frame to the access point, it first sends an RTS short frame. The access point responds by sending CTS back to the requesting node, which is also received by all other nodes (this is modeled by multiple short_frame invocations by the access point process). The assumption here is that by the time a node receives its CTS, all other nodes would have received the CTS as well.

Three important features of πZ play a crucial role in modeling the hidden node problem: (1) The channel state allows us to model durable nature of frame transmission and the resulting signal collisions. If frame transmissions were instantaneous there would be no collisions. (2) Powerful data abstraction facilities inherited from Z allow us to abstractly specify important aspects of RF interfaces, such as active stations and signal coverage, as a part of the stateful channels representing the interfaces. (3) Channel mobility inherited from the π-calculus allows us to specify communicating wireless nodes that pass around their RF interface addresses as communicating processes that in turn pass around channel names.
3.2 Reductions and Labelled Transitions

3.2.1 Reduction Semantics

Reduction semantics for $\pi Z$ is stated in terms of a collection reduction rules (Figure 3.7). As in the $\pi$-calculus, reductions in $\pi Z$ are defined up to structural equivalence (R-STRUCT rule). Structural equivalence rules for $\pi Z$ processes are identical to that for $\pi$-calculus processes given in Figure 2.2. Of special interest is the R-OP rule, which specifies the reduction rule for stateful channel communication. Other rules are similar to the corresponding $\pi$-calculus reduction rules. In order for communication over a stateful channel to take place, three concurrent subterms must synchronize on the channel. The first term, $\alpha p(\bar{w}).P + R_1$, in the R-OP rule is the output term supplying operation $p$'s parameter values $\bar{w}$. If the output term has a simpler form, $\overline{\alpha} p(\bar{w}).P$, the R-OP rule is still applicable since $\overline{\alpha} p(\bar{w}).P \equiv \overline{\alpha} p(\bar{w}).P + 0$. The second term in the R-OP rule is the state-holding term, the state being of the form of a labelled tuple $W$. The third term, $\alpha' p(\bar{z}).Q + R_2$, in the R-OP rule is the term executing the
input action (input term) and consumes operation \( p \)'s values via \( \varepsilon \).

The \( R\text{-}OP \) rule makes use of the relational interpretation, \( \mathcal{I}(p) \), of operation schemas in order to determine the after-state and result values. This means that a reduction may be performed only if the operation's pre-condition is satisfied by the current state and the supplied input values. By investigating an operation's precondition and ensuring that it evaluates to ‘true’ we may make sure that the pre-condition does not block the operation. After the reduction, the output prefix is removed from the output term, the state-holding term changes its state, and result values are substituted in place of bound input variables in the input term. Notes: (i) Substitution is assumed to be capture-free. (ii) A state-holding term must appear freely in a process (i.e., not under summation or pre-fixing) in order for it to participate in reduction. We illustrate below the reductions of \( \pi \mathbb{Z} \) processes using the buffered channel example from the previous section.

1. \( \overline{q}\text{put}(3,p).P \mid q\langle \text{items} = \{\}, \text{size} = 100 \rangle : \overline{z}\text{BufferedChannel}_k \mid q'\text{put}.Q \rightarrow P \mid q\langle \text{items} = ((3,p)), \text{size} = 100 \rangle : \overline{z}\text{BufferedChannel}_k \mid Q \)

2. \( \overline{q}\text{get}.P \mid q\langle \text{items} = ((3,p)), \text{size} = 100 \rangle : \overline{z}\text{BufferedChannel}_k \mid q'\text{get}(x,y).\overline{y}\text{put}(x,q).Q \rightarrow P \mid q\langle \text{items} = \{\}, \text{size} = 100 \rangle : \overline{z}\text{BufferedChannel}_k \mid \overline{p}\text{put}(3,q).Q\{3/x,p/y\} \)

Proposition 3.2.1 states that if a state-holding term satisfies the state invariant, then reductions preserve the invariance satisfaction.

**Proposition 3.2.1.** If \( P \equiv x\langle W \rangle : \overline{z}D | P' \) such that \( W \in \mathcal{I}(D_s) \) where \( D_s \) is the state schema of abstract type \( D \), and \( P \rightarrow^* Q \), then \( Q \) contains a term of the form \( x\langle W' \rangle : \overline{z}D \) such that \( W' \in \mathcal{I}(D_s) \).

**Proof.** We induct on the structure of reduction rules. When considering base cases, the only rule that causes state transformation is \( R\text{-}OP \), which we may apply if the process contains the state-holding term as well as the corresponding input and output terms for some operation \( p \). Moreover, the operation’s pre-condition must be satisfied by the current state (say, \( W \)) and the values supplied by the output term (say, \( \tilde{w} \)). In the \( R\text{-}OP \) rule \( (W, \tilde{w}, W', \tilde{v}) \in \mathcal{I}(p) \), where \( W' \) is the after-state and \( \tilde{v} \) is the values consumed by input term. As per the definition of \( \mathcal{I}(p) \) (Definition 2), we have that \( W' \in \mathcal{I}(D_s) \), which concludes the case. When considering other base cases (e.g., \( R\text{-EQ} \) or \( R\text{-NEQ} \)), an existing free state-holding term is
never modified or removed by any of these rules. The inductive cases (R-PAR and R-RES rules) follow directly from the inductive hypothesis, since the term composed in parallel (for R-PAR) or name being restricted (for R-RES) does not influence state transformation.

3.2.2 Labelled Transition Semantics

In this section, we develop labelled transition semantics for \( \pi Z \). We use the following notations when discussing labelled semantic rules and their properties: \( fn(P) \) and \( fv(P) \) indicate the set of free names and free variables in \( P \) respectively; \( n(\overline{v}) \) indicates the set of names appearing in \( \overline{v} \); the symbol \( \mu \) ranges over transition labels; \( n(\mu) \) indicates the set of names appearing in \( \mu \); and \( bn(\mu) \) indicates the set of bound names appearing in \( \mu \).

Input and output transitions rules (L-OUT and L-IN) given below are similar to the corresponding rules for the \( \pi \)-calculus, with the difference that the actions and their labels have operation names associated with them, in addition to channel names. An input transition is of early style [80], where input values are instantiated along with the transition.

\[
L-OUT: \quad \overline{\pi p(\overline{w}).P} \xrightarrow{\overline{\pi p(\overline{w})}} P
\]

\[
L-IN: \quad a^p(\overline{z}).Q \xrightarrow{a^p(\overline{v})} Q\{\overline{v}/\overline{z}\}
\]

**Action Obligations** A novelty of the semantics is the transition rule for a state-holding term. Since a \( \pi \)-calculus transition label for a term represents an action performed by the term, we are tempted to label the state-holding term’s transition with before-state and after-state labels. For instance, \( a\langle W\rangle;\!\!\!\!\; D \xrightarrow{W;W'} a\langle W'\rangle;\!\!\!\!\; D \), where \( W \) and \( W' \) represent respectively the before-state and after-state associated with the transition. However, this approach has some undesirable consequences. Firstly, bisimilarity induced by the labelled transitions becomes too discriminating, being dependent on the state representation. Secondly, data refinement – an important property for \( Z \) abstract types – will have no interesting effect on the processes since state representation is exposed via transition labels. Thirdly, if state appears
in a label, we must extrude all the restricted names stored as part of the state to the input term, since we do not know which ones the input term is dependent upon. This results in unwanted scope extrusion.

We avoid these issues by introducing the notion of action obligations. A state-holding term transitions to an altered state through a transition labelled with a pair of action obligations, i.e., a pair of input and output actions that the environment must fulfil for the state transition to take place. This approach is presented in the $L-ST$ rule given below. The rule states that a state-holding term for some channel $a$ may change its state if the environment fulfils a pair of action obligations present in the transition label. The first element in the pair, indicated by $\uparrow a'p(\bar{w})$, is the obligation that the environment executes an output action on the channel $a$ for some operation $p$, supplying values $\bar{w}$. The second element in the pair, indicated by $\uparrow a''p(\bar{v})$, is the obligation that the environment executes an input action on the channel $a$ for the operation $p$, consuming values $\bar{v}$. The side condition states that the before-state $W$, the output values $\bar{w}$ (i.e., the operation’s parameters), the after-state $W''$, and the input values $\bar{v}$ (i.e., the operation’s results) must be related as per the relational interpretation $\mathcal{I}(p)$ of $p$’s operation schema.

$$\begin{align*}
L-ST: & \quad (W, \bar{w}, W', \bar{v}) \in \mathcal{I}(p) \\
& \quad a(W) : \exists D \quad \frac{p \in \Sigma(D)}{\uparrow a'p(\bar{w}), \uparrow a''p(\bar{v}) \rightarrow a(W') : \exists D}
\end{align*}$$

**Scope Extrusion** Scope extrusion rules need careful consideration. A name may extrude scope from an output term to a state-holding term and also from a state-holding term to an input term. The $L-OUT-OPEN$ rule deals with the scope extrusion of names from an output term to a state-holding term and is similar to the standard scope extrusion rule for $\pi$-calculus. The $L-ST-OPEN$ rule deals with the scope extrusion of names from a state-holding term to an input term. The extruded names are attached to the input action obligation. A name being extruded must appear in the value tuple of the input action obligation, but must not appear in the value tuple of the output action obligation.
Structurally, the three terms involved in the stateful channel communication – the input term, the output term, and the state-holding term – may appear in any order. Therefore, we need rules that allow transitions of the state-holding term to interact with both the input transitions and the output transitions. The L-ST-IN rule allows interaction between transitions from the state-holding term and the input term. The interaction results in the scope extrusion of names from the state-holding term to the input term as well as fulfillment of the input action obligation. In the transition for the combined term, the input action obligation is removed, indicating that the obligation is fulfilled. The L-ST-OUT rule is similar, allowing interaction between transitions from the state-holding term and the output term. The scope extrusion now occurs from the output term to the state-holding term. The combined term results in a transition with the output action obligation removed, indicating its fulfilment. The symmetric versions of the L-ST-IN and the L-ST-OUT rules are not shown here.

Stateful vs Stateless Communication  Consider the following process: $P \mid Q \mid R$, where $P$ is an input term, $Q$ is the corresponding output term, and $R$ is the corresponding state-holding term. The three terms can interact by first letting $Q$ to interact with $R$ by applying
the rule \textit{L-ST-OUT}. However, suppose that the associativity for the processes are explicitly specified as follows: \((P | Q) | R\). Now \(R\) may interact only with the combined transition of \(P\) and \(Q\). Thus far, our rules do not allow interactions between an input term and an output term. In order to allow processes to be associated in all possible ways, we must add one more rule – the \textit{L-IN-OUT} rule given below (symmetric version not shown). The \textit{L-IN-OUT} rule allows an input term to interact with an output term so that we may infer a combined transition.

\[
\begin{align*}
\text{L-IN-OUT:} & \quad P \xrightarrow{\alpha p(\bar{v})} P' \quad Q \xrightarrow{(\nu \tilde{n}) \pi p(\bar{\omega})} Q' \\
P | Q & \xrightarrow{(\nu \tilde{n}) \pi p(\bar{\omega}), \alpha p(\bar{v})} P' | Q' \\
\tilde{n} \cap fn(P) &= \emptyset
\end{align*}
\]

The \textit{L-IN-OUT} rules brings out the clear difference between the stateful communication in \(\pi Z\) and the stateless communication in the \(\pi\)-calculus. In the former case, we need the state-holding term to complete the communication. Moreover, the input values need not match the output values (typically, they will not match). In the latter case, we would have inferred a \(\tau\)-transition when an input term and the corresponding output term come together. In addition, since output values are simply copied to the input term, the input values and the output values must exactly match.

An output term may lie between the corresponding state-holding term and the corresponding input term. As a result, in order to allow restricted names to be passed from the state-holding term to the input term we need one more scope extrusion rule. This is accomplished by the \textit{L-IN-OB-OPEN} rule, which allows scope extrusion of restricted names present in an input obligation that resulted from the combined transition of the state-holding term and the output term.

\[
\begin{align*}
\text{L-IN-OB-OPEN} & \quad P \xrightarrow{(\nu \tilde{n}) \uparrow \alpha p(\bar{v})} P' \\
& \quad \nu n P \xrightarrow{(\nu n, \tilde{m} \uparrow \alpha p(\bar{v})}, P'
\end{align*}
\]

Similarly, we need to allow scope extrusion of restricted names present in an output action from the combined transition of an output term and the corresponding input term. This is accomplished by the \textit{L-IN-OUT-OPEN} rule.
Finally, the L-OP-1, the L-OP-2, and the L-OP-3 rules allow us to infer τ-transitions for processes that are internally able to perform the matching input, output, and state-holding transitions. The L-OP-1 rule specifies the interaction between the combined transition from the output term and the state-holding term, on the one hand, and the input action, on the other hand. This results in fulfilment of the remaining input action obligation. Names may extrude scope to the input term, via the input action obligation. The L-OP-2 rule specifies the interaction between the combined transition from the input term and the state-holding term, on the one hand, and the output action, on the other hand. This results in the fulfilment of the remaining output action obligation. Names may extrude scope from the output term along with the output values. The L-OP-3 rule specifies the interaction between the combined transition from the input term and the output term and the transition from the state-holding term. Names may extrude scope both ways — from the state-holding term as well as from the output term. The symmetric versions of the rules are not shown here.

To complete the labelled transition semantics, we need a few more additional rules, which are similar to the corresponding π-calculus rules. A complete set of labelled transition rules for πZ are given in Section 3.5. Note that only input-guarded or output-guarded terms under a summation are allowed to participate in transitions (the L-SUM rule, Figure 3.9, Appendix 47

\[
\begin{align*}
\text{L-IN-OUT-OPEN:} & \quad P \xrightarrow{(\nu \bar{m})\pi p(\bar{v}),a'p(\bar{v})} P' \\
& \quad \nu \bar{n} P \xrightarrow{(\nu n, \bar{m})\pi p(\bar{w}),a'p(\bar{v})} P' \\
& \quad n \in \bar{w}, n \notin \bar{m}, n \neq a
\end{align*}
\]

Finally, the L-OP-1, the L-OP-2, and the L-OP-3 rules allow us to infer τ-transitions for processes that are internally able to perform the matching input, output, and state-holding transitions. The L-OP-1 rule specifies the interaction between the combined transition from the output term and the state-holding term, on the one hand, and the input action, on the other hand. This results in fulfilment of the remaining input action obligation. Names may extrude scope to the input term, via the input action obligation. The L-OP-2 rule specifies the interaction between the combined transition from the input term and the state-holding term, on the one hand, and the output action, on the other hand. This results in the fulfilment of the remaining output action obligation. Names may extrude scope from the output term along with the output values. The L-OP-3 rule specifies the interaction between the combined transition from the input term and the output term and the transition from the state-holding term. Names may extrude scope both ways — from the state-holding term as well as from the output term. The symmetric versions of the rules are not shown here.

\[
\begin{align*}
\text{L-OP-1:} & \quad P \xrightarrow{(\nu \bar{m})\pi a'(\bar{v})} P' \quad Q \xrightarrow{a'p(\bar{v})} Q' \\
& \quad P \mid Q \xrightarrow{\tau} \nu \bar{n} P' \mid Q' \\
& \quad \bar{m} \cap \text{fn}(Q) = \emptyset
\end{align*}
\]

\[
\begin{align*}
\text{L-OP-2:} & \quad P \xrightarrow{\pi p(\bar{w})} P' \quad Q \xrightarrow{(\nu \bar{n})\pi p(\bar{w})} Q' \\
& \quad P \mid Q \xrightarrow{\tau} \nu \bar{n} P' \mid Q' \\
& \quad \bar{n} \cap \text{fn}(P) = \emptyset
\end{align*}
\]

\[
\begin{align*}
\text{L-OP-3:} & \quad P \xrightarrow{\pi p(\bar{w}), (\nu \bar{n})\pi a'(\bar{v})} P' \quad Q \xrightarrow{(\nu \bar{n})\pi p(\bar{w}), a'p(\bar{v})} Q' \\
& \quad P \mid Q \xrightarrow{\tau} \nu \bar{m}, \bar{n} P' \mid Q' \\
& \quad \bar{n} \cap \text{fn}(P) = \emptyset, \bar{m} \cap \text{fn}(Q) = \emptyset, \bar{n} \cap \bar{m} = \emptyset
\end{align*}
\]
By using a process syntax with recursion instead of replication, we have avoided the need to define several additional rules to take care of process interactions under replication. The transition system generated by the rules described in this section is an early transition system [80]. Late transitions [67, 80] are problematic since they introduce free variables into the process terms. If a free variable inhabits an output action, stateful communication cannot occur without instantiating the variable since our transitions are based on the relational interpretation of operations. Furthermore, being free the variable would need to be globally instantiated. Therefore, an early transition system is naturally suitable for $\pi Z$.

### 3.2.3 Correspondence between reductions and $\tau$-transitions

We now work towards our first major result, namely, that reductions correspond to $\tau$-transitions. Since reductions rely on structural equivalence to break the rigid process structure, we first prove Lemma 3.2.2, an important property of structural equivalence. The equivalent of Lemma 3.2.2 for the $\pi$-calculus is proved as part of the Harmony Lemma in [80].

**Lemma 3.2.2** (Labelled transitions and structural equivalence). If $P \equiv Q$ the following holds: (a) $P \xrightarrow{\mu} P'$ implies there exists some $Q'$ such that $Q \xrightarrow{\mu} Q'$ and $P' \equiv Q'$, (b) $Q \xrightarrow{\mu} Q'$ implies there exists some $P'$ such that $P \xrightarrow{\mu} P'$ and $P' \equiv Q'$.

**Proof.** By induction on the inference of structural equivalence. We need to consider all the rules for the structural equivalence given in Figure 2.2, along with rules for reflexivity, symmetry, and transitivity. We consider a few representative cases here. For instance, consider the associativity axiom for the operator `|`. Let $P = P_1 \mid (P_2 \mid P_3)$. By applying the rule, $Q = (P_1 \mid P_2) \mid P_3$. Let us consider the case in which all the three subterms participate in $P$'s transition. One possibility is that $P_1$ is an output term, $P_2$ is a state-holding term, and $P_3$ is an input term. $P$'s transition inference tree has $L-IN$, $L-ST$, and $L-OUT$ as leaves, followed by the application of $L-ST-OUT$ and finally $L-OP-1$. $Q$ can match this with a transition having an inference tree with identical leaves (since all the three terms are present in $Q$), followed by an application of $L-ST-IN$ and finally $L-OP-2$.

Now, consider the $S-EXTR$ rule. Let $P = \nu n P_1 \mid P_2$. By applying the $S-EXTR$ axiom,
Lemma 3.2.3.

Consider the last case \( P_1 \) alone, which is the most intricate. The only rule whose conclusion matches the required transition structure is \( L-ST-OUT \). The \( L-ST-OUT \) rule has two premises: one for the state-holding term’s transitions and the other for the output term’s transitions. Since \( \bar{v} \) appears in the input action obligation of the state-holding term’s transition and since \( n \notin fn(P_1) \), it follows that the state-holding term’s transition must come from \( P_2 \). Therefore, we have \( P_1 \xrightarrow{[\nu \bar{b}]p(\bar{w})} P'_1 \) and \( P_2 \xrightarrow{[\nu \bar{w}]a'p(\bar{v})} P'_2 \), for some \( \bar{w} \) and \( \bar{b} \) such that the side conditions of the rule hold. Moreover, by applying \( L-ST-OUT \), we get \( P' = \nu \bar{b} P'_1 | P'_2 \). Now, we need to check whether \( Q = P_1 | \nu n P_2 \) can match this transition. By applying \( L-ST-OPEN \), we have \( \nu n P_2 \xrightarrow{[\nu \bar{w}]a'p(\bar{v})} \nu b P'_1 | P'_2 \). Now, applying the \( L-ST-OUT \) rule, we get \( P_1 | \nu n P_2 \xrightarrow{[\nu \bar{w}]a'p(\bar{v})} \nu b P'_1 | P'_2 \).

The reflexive case \( P \equiv P \) is trivial. The symmetric and transitive cases directly follow from the inductive hypothesis. Other inductive cases are \( S-PAR \) and \( S-RES \), which can be proved by analyzing the applicable transition rules in a similar fashion to the cases analyzed above.

Since \( \tau \)-transitions are inferred from other labelled transitions, we establish some properties regarding the structure of the processes that engage in various labelled transitions in Lemma 3.2.3.

**Lemma 3.2.3.**

1. If \( P \xrightarrow{[\nu \bar{w}]a'p(\bar{v})} P \) then \( P \equiv \nu \bar{m}, \bar{b} a(W), \bar{v}D | R \) and \( P \equiv \nu \bar{b} a(W'), \bar{v}D | R \), where \( (W, \bar{w}, W', \bar{v}) \in I(p), p \in \Sigma(D), a \notin \bar{m}, \bar{b} \) and \( n(\bar{v}) \cap \bar{b} = \emptyset \).
2. If \( P \xrightarrow{(\nu n)\pi p(\bar{w})} P' \) then \( P \equiv \nu \bar{m}, \bar{c} (\pi p(\bar{w}).P' + R_0) \mid R \), and \( P' \equiv \nu \bar{c} P' \mid R \), where \( a \notin \bar{m}, \bar{c} \) and \( \bar{c} \cap n(\bar{w}) = \emptyset \).

3. If \( P \xrightarrow{a p(\bar{v})} P' \), then \( P \equiv \nu \bar{c} (a(\bar{z}).P' + R_0) \mid R \), and \( P' \equiv \nu \bar{c} P' \{ \bar{v}/\bar{z} \} \mid R \), where \( a \notin \bar{c} \) and \( \bar{c} \cap n(\bar{v}) = \emptyset \).

4. If \( P \xrightarrow{(\nu \bar{m})a p(\bar{v})} P' \) then \( P \equiv \nu \bar{m}, \bar{c} (\pi p(\bar{w}).P' + R_0) \mid a(W) \mid R \) and \( P' \equiv \nu \bar{c} P' \mid a(W') \mid R \), where \((W, \bar{w}, W', \bar{v}) \in I(p), p \in \Sigma(D), a \notin \bar{m}, \bar{c} \) and \( \bar{c} \cap n(\bar{v}) = \emptyset \).

Proof. All the cases may be proved by straightforward induction on the derivation of the corresponding labelled transition. Consider case (1): The base case is the application of \( L-ST \) rule and the inductive cases are applications of \( L-ST-OPEN, L-PAR, \) and \( L-RES \) rules. Case (4) needs to use results from cases (1) and (2).

Theorem 3.2.4 states that reductions and labelled transitions can match each other. Both Lemma 3.2.3 and 3.2.2 are required to complete the proof of Theorem 3.2.4. Since reductions identify terms up to structural equivalence, reductions may alter the structure of the reduced terms in all possible ways, whereas \( \tau \)-transitions cannot make such unnecessary structural changes to the term. As a result, in the forward direction we have \( P \rightarrow Q \iff P \xrightarrow{\tau} Q' \) such that \( Q \equiv Q' \), i.e., a \( \tau \)-transition can only match the reduced term up to structural equivalence.

**Theorem 3.2.4** (Reductions match \( \tau \)-transitions). \( P \rightarrow Q \iff P \xrightarrow{\tau} Q' \) such that \( Q \equiv Q' \).

Proof. The proof is done in two parts: (1) \( P \rightarrow Q \) implies \( P \xrightarrow{\tau} Q' \) such that \( Q \equiv Q' \), and (2) \( P \xrightarrow{\tau} Q \) implies \( P \rightarrow Q \).

(1) \( P \rightarrow Q \) implies \( P \xrightarrow{\tau} Q' \) such that \( Q \equiv Q' \). By induction on the derivation of the reduction relation. Base cases except \( R-OP \): The rules \( R-EQ, R-NEQ, \) and \( R-REC \) directly correspond to their labelled counterparts, i.e., \( L-EQ, L-NEQ, \) and \( L-REC \). For \( R-OP \), we can infer a \( \tau \)-transition with the inference tree having \( L-OUT, L-ST, \) and \( L-IN \) as the leaves. The inductive cases \( R-PAR \) and \( R-RES \) are proved using \( L-PAR \) and \( L-RES \). The last inductive case of \( R-STRUCT \) is proved by appealing to Lemma 3.2.2.

(2) \( P \xrightarrow{\tau} Q \) implies \( P \rightarrow Q \). By induction on the derivation of \( \tau \) transitions. We consider one case here, that of \( L-OP-1 \) rule. The transition has the form: \( P \mid Q \xrightarrow{\tau} \nu \bar{m} P' \mid Q' \), where
By Lemma 3.2.3 (3) and Lemma 3.2.3 (4), we have:

\[
P \xrightarrow{(\nu \tilde{m})!a'\tilde{p}(\tilde{w})} P' \quad \text{and} \quad Q \xrightarrow{a'\tilde{p}(\tilde{v})} Q'.
\]

By \(\alpha\)-conversion we can make sure that bound names of \(P\) and \(Q\) are different and also different from each other’s free names. Therefore, we may re-arrange the sub-terms to get:

\[
P \mid Q = (\nu \tilde{m}, \tilde{c} \pi \tilde{p}(\tilde{w}).P'' + R_0 \mid a\langle W\rangle : \tilde{z}D \mid R_1) \mid (\nu \tilde{b} a'\tilde{p}(\tilde{z}).Q'' + R'_0 \mid R_2), \text{ and,}
\]

\[
\nu \tilde{m} P' \mid Q' = \nu \tilde{m} (\nu \tilde{c} P'' \mid a\langle W'\rangle : \tilde{z}D \mid R_1) \mid (\nu \tilde{b} Q''\{\tilde{v}/\tilde{z}\} \mid R_2), \text{ where,}
\]

\((W, \tilde{w}, W', \tilde{v}) \in I(p)\).

Now, applying \(R\)-OP and \(R\)-STRUCT rules we get the following reduction:

\[
P \mid Q \rightarrow \nu \tilde{m}, \tilde{c}, \tilde{b} P'' \mid a\langle W'\rangle : \tilde{z}D \mid Q''\{\tilde{v}/\tilde{z}\} \mid R_1 \mid R_2. \text{ Let the reduced term be called } M. \text{ It is easy to see that } M \equiv \nu \tilde{m} P' \mid Q'.
\]

The cases of \(L\)-OP-2 and \(L\)-OP-3 rules can be similarly analyzed. The rules \(L\)-EQ, \(L\)-NEQ, and \(L\)-REC directly correspond to their unlabelled counterparts, i.e., \(R\)-EQ, \(R\)-NEQ, and \(R\)-REC. The inductive cases of \(L\)-PAR and \(L\)-RES directly follows from the corresponding \(R\)-PAR and \(R\)-RES rules.

\[\square\]

### 3.3 Bisimilarity

The main result that we wish to show is that bisimilarity is closed under parallel composition, name restriction, and a restricted form of recursion. Definition 3 recollects the standard definition of bisimulation and bisimilarity found in the \(\pi\)-calculus literature.

**Definition 3** (Bisimulation and Bisimilarity). A binary relation \(\mathcal{R}\) on processes is a bisimulation if whenever \(P \mathcal{R} Q\):

1. \(P \xrightarrow{\mu} P'\) implies \(Q \xrightarrow{\mu} Q'\) and \(P' \mathcal{R} Q'\)
2. \(Q \xrightarrow{\mu} Q'\) implies \(P \xrightarrow{\mu} P'\) and \(P' \mathcal{R} Q'\)

Bisimilarity \((\sim)\) is the largest bisimulation.

Since bisimilarity is the largest bisimulation, every other bisimulation is contained in it. As a result, in order to show that two processes are bisimilar we may exhibit a bisimulation containing the two processes \([79]\). However, since bisimulations could be very large relations

---

51
this could be difficult in most cases. The up-to-techniques developed in [78] and further elaborated in [80] provide us with a more tractable method for establishing bisimilarity.

We recapitulate from [80] the following definitions and lemmas related to up-to techniques: Definition 4, Definition 5, Lemma 3.3.1, Corollary 3.3.4, Lemma 3.3.5, Definition 6, Lemma 3.3.2, and Corollary 3.3.6.

**Definition 4** (Strong progression). Relation $\mathcal{R}$ strongly progresses to relation $\mathcal{S}$, written $\mathcal{R} \rightsquigglyleftarrow \mathcal{S}$, if whenever $P \mathcal{R} Q$,

1. $P \xrightarrow{\alpha} P'$ implies $Q \xrightarrow{\alpha} Q'$ and $P' \mathcal{S} Q'$.
2. $Q \xrightarrow{\alpha} Q'$ implies $P \xrightarrow{\alpha} P'$ and $P' \mathcal{S} Q'$.

Naturally, given a relation $\mathcal{R}$ on processes, if $\mathcal{R} \rightsquigglyleftarrow \mathcal{R}$ then $\mathcal{R}$ is a bisimulation and, therefore, if $P \mathcal{R} Q$ then $P \sim Q$. The theory of up-to techniques provides us with a much more general way of establishing bisimilarity by introducing the notion of strongly safe (and strongly quasi-safe) functions on process relations (Definition 5 and Definition 6).

**Definition 5** (Strongly safe function). A function $\mathcal{F}$ on process relations is strongly safe if $\mathcal{R} \subseteq \mathcal{S}$ and $\mathcal{R} \rightsquigglyleftarrow \mathcal{S}$ implies $\mathcal{F}(\mathcal{R}) \subseteq \mathcal{F}(\mathcal{S})$ and $\mathcal{F}(\mathcal{R}) \rightsquigglyleftarrow \mathcal{F}(\mathcal{S})$.

**Definition 6** (Quasi-safe functions). A function $\mathcal{F}$ is strongly quasi-safe if there is a strongly safe function $\mathcal{G}$ such that $\mathcal{F} \subseteq \mathcal{G}$.

Simple examples of strongly safe functions include the identity function ($\mathcal{F}_i$) and the constant function that maps any relation to structural equivalence relation ($\mathcal{F}_\equiv$).

Essentially, if we have $\mathcal{R} \rightsquigglyleftarrow \mathcal{F}(\mathcal{R})$, where $\mathcal{F}$ is a strongly safe (or strongly quasi-safe) function on process relations, then for any $P, Q$ such that $P \mathcal{R} Q$, we have $P \sim Q$ even though $\mathcal{R}$ itself need not be a bisimulation (Lemma 3.3.1 and Lemma 3.3.2).

**Lemma 3.3.1.** If $\mathcal{F}$ is strongly safe and $\mathcal{R} \rightsquigglyleftarrow \mathcal{F}(\mathcal{R})$, then $\mathcal{R}$ and $\mathcal{F}(\mathcal{R})$ are included in $\sim$.

**Lemma 3.3.2.** If $\mathcal{F}$ is strongly quasi-safe and $\mathcal{R} \rightsquigglyleftarrow \mathcal{F}(\mathcal{R})$, then $\mathcal{R}$ and $\mathcal{F}(\mathcal{R})$ are included in $\sim$.

---

2Corollary 3.3.6 is not explicitly mentioned in [80].
Moreover, we could combine strongly safe functions (and strongly quasi-safe functions) using the so-called strongly secure operators (Definition 7 and Lemma 3.3.3) to yield more complex safe functions. Examples of strongly secure operators that are also monotone include union, composition, and chaining.

**Definition 7 (Strongly secure operator).** A operator on functions on process relations is strongly secure if when applied to strongly safe arguments, it yields a strongly safe result.

**Lemma 3.3.3.** Suppose a function $F$ is the result of a combination of functions that are strongly quasi-safe and monotone, and of operators that are strongly secure and monotone. Then $F$ is strongly quasi-safe.

Corollary 3.3.4, Lemma 3.3.5, and Corollary 3.3.6 are some results from [80] related to the notion of strongly safe (and strongly quasi-safe) functions, which are directly used to prove some of our results.

**Corollary 3.3.4.** If $F$ is strongly safe and $\sim \subseteq F(\sim)$, then $F(\sim) = \sim$.

**Lemma 3.3.5.** Suppose $F$ is such that $R \subseteq S$ and $R \sim S$ implies $F(R) \subseteq F^*(S)$ and $F(R) \sim F^*(S)$. Then $F^*$ is strongly safe.

**Corollary 3.3.6.** If $F$ is strongly quasi-safe and $\sim \subseteq F(\sim)$, then $F(\sim) = \sim$.

We now define extended static contexts (Definition 8) by extending the definition of static contexts given in [42] and prove two lemmas (Lemma 3.3.8 and Lemma 3.3.9) which allow us to establish bisimilarity of $\pi Z$ processes using the “up-to-context” technique.

**Definition 8 (Extended static context).** An extended static context is defined as follows:

1. $\_\_$ is an extended static context.

2. if $E$ is an extended static context then $E \mid Q$ and $Q \mid E$ are extended static contexts for any process $Q$.

3. if $E$ is an extended static context then $\nu n\ E$ is a extended static context for any name $n$. 

53
4. if $E$ is an extended static context then $\mu x.(E \mid x)$ is an extended static context.

**Proposition 3.3.7.** If $E$ is an extended static context, then $fv(E) = \emptyset$.

**Proof.** By simple induction on the structure of extended static context. □

We may extend Definition 8 to contexts containing multiple indexed holes. If $E$ is a multi-hole context, we use the syntax $E[P]$, where $P$ is a list of processes, to indicate the process we obtain after replacing the indexed holes in $E$ with corresponding processes in $P$. It is easily seen that Proposition 3.3.7 is also applicable to multi-hole contexts.

**Definition 9.** The function $F_{es1}$ is defined as:

$$F_{es1}(R) = \{ E[P], E[Q] \mid E \text{ is a 1-hole extended static context and } P \mathbin{R} Q \}$$

The function $F_{es}$ is defined in terms of multi-hole extended static contexts as $F_{es1}$ is defined in terms of single-hole contexts.

**Lemma 3.3.8.** $F_{es} \subseteq (F_{es1})^* \cup F_{id}$

**Proof.** Let $\mathcal{R}$ be a relation on processes. Then, any pair $P' F_{es}(\mathcal{R}) Q'$ can be written as $E[P] F_{es}(\mathcal{R}) E[Q]$ where $E$ is a multi-hole context, and $P_i \mathbin{\mathcal{R}} Q_i$ for all $P_i \in P$, $Q_i \in Q$. We want to show that any pair of the form $E[P] F_{es}(\mathcal{R}) E[Q]$ can be written using a composition of the $F_{es1}(\mathcal{R})$ relation, i.e., $E[P] F_{es}(\mathcal{R}) E[Q]$ implies $E[P] F_{es1}(\mathcal{R}) E[Q]$ for some $j$. We first consider multi-hole contexts having one or more holes without any repetition of indexed holes and argue by induction on the number of holes. As base case we consider contexts with one hole, which trivially satisfies the requirement. Consider a pair formed by instantiating an $n + 1$ hole context: $E[P_1, \ldots, P_{n+1}] F_{es}(\mathcal{R}) E[Q_1, \ldots, Q_{n+1}]$, where $P_i \mathbin{\mathcal{R}} Q_i$ for $i = 1..n+1$. Now, $E[P_1, P_2, \ldots, P_n, -]$ is a single-hole context, hence $E[P_1, \ldots, P_n, P_{n+1}] F_{es1}(\mathcal{R}) E[P_1, \ldots, P_n, Q_{n+1}]$ (A). $E[-, \ldots, -, Q_{n+1}]$ is an $n$-hole context, hence $E[P_1, \ldots, P_n, Q_{n+1}] F_{es}(\mathcal{R}) E[Q_1, \ldots, Q_{n+1}]$. By inductive hypothesis, pairs instantiated using $n$-hole contexts can be written using composition of the one-hole context function: $E[P_1, \ldots, P_n, Q_{n+1}] F_{es1}(\mathcal{R}) E[Q_1, \ldots, Q_{n+1}]$ (B), for some $k$. Composing (A) and (B), we get $E[P_1, \ldots, P_n, P_{n+1}] F_{es1}(\mathcal{R}) E[Q_1, \ldots, Q_{n+1}]$, which concludes the inductive proof. Any instance $E[P]$ of a multi-hole context containing repeating indexed holes can be
Hence we have $F_{es} \subseteq (F_{es1})^* \cup F_{id}$.}

**Corollary 3.3.9.** $F_{es^+} \subseteq (F_{es1})^*$, where $F_{es^+}$ uses multi-hole contexts with at least one hole.

**Lemma 3.3.10.** $(F_{es1})^*$ is strongly safe.

*Proof.* Let $R \subseteq S$ and $R \leadsto S$. By Lemma 3.3.5 and Corollary 3.3.9, we need to show that:
(A) $F_{es1}(R) \subseteq F_{es^+}(S)$, and (B) $F_{es1}(R) \leadsto F_{es^+}(S)$. The result (A) directly follows from definitions of the two functions and the condition that $R \subseteq S$. To prove (B), we need to establish that if $E$ is a 1-hole context, $P \leadsto Q$, and $E[P] \xrightarrow{\mu} P'$, then $E[Q] \xrightarrow{\mu} Q'$ such that $P' F_{es^+}(S) Q'$.

We proceed by induction on the structure of $E$.

1. $E$ is $\mu x.(E' | x)$. Since $E'$ does not contain the free variable $x$ (Proposition 3.3.7), $E[P]$ can only make a $\tau$-transition, $E[P] \xrightarrow{\tau} E'[P] | \mu x.(E'[P] | x)$. $E[Q]$ can make a matching $\tau$-transition: $E[Q] \xrightarrow{\tau} E'[Q] | \mu x.(E'[Q] | x)$. Let $E'' = E' | \mu x.(E' | x)$. Note that $E''$ is a multi-hole context. So, $E[P] \xrightarrow{\tau} E''[P]$ and $E[Q] \xrightarrow{\tau} E''[Q]$. Since $R \subseteq S$, we have $P S Q$. Hence $E''[P] F_{es^+}(S) E''[Q]$.

2. $E$ is $E' | R$. In any move by $E[P] \xrightarrow{\mu} P'$, either $E'[P]$ or $R$ alone participates, or both the sub-processes participate. We consider the last case, i.e., either $L-OUT-ST$, $L-IN-ST$, $L-IN-OUT$, $L-OP-1$, $L-OP-2$, $L-OP-3$ is applied to infer the transition. Let $E'[P] \xrightarrow{\mu_1} P''$ and $R \xrightarrow{\mu_2} R'$, for some appropriate $\mu_1$ and $\mu_2$ such that the combined transition has $\mu$ as its label. Therefore $E[P]$’s transition has the form: $E[P] \xrightarrow{\mu} \nu \tilde{n} P'' | R'$, for some $\tilde{n}$. Using the inductive hypothesis, $E'[Q] \xrightarrow{\mu} Q''$ such that $P'' F_{es^+}(S) Q''$. Therefore $P'' = E''[S]$ and $Q'' = E''[S']$ such that $S_i S S_i'$ where $E''$ is some multi-hole context. Let $E''' = \nu \tilde{n} E'' | R'$. Therefore, $E[P] \xrightarrow{\mu} E'''[S]$ and $E[Q] \xrightarrow{\mu} E'''[S']$, i.e., $E[P] \xrightarrow{\mu} P'$ and $E[Q] \xrightarrow{\mu} Q'$ such that $P' F_{es^+}(S) Q'$.

3. $E$ is $\nu n E'$. Five applicable rules are $L-OUT-OPEN$, $L-ST-OPEN$, $L-IN-OB-OPEN$, $L-IN-OUT-OPEN$, and $L-RES$. In the first four cases, the restricted name is extruded via the transition. The cases are easy to show. 

\[\Box\]
Corollary 3.3.11. $\mathcal{F}_{es}$ is quasi-safe.

Theorem 3.3.12 (Bisimilarity is closed under extended static contexts). $P \sim Q$ implies $E[P] \sim E[Q]$.

Proof. The result follows from Corollary 3.3.11 and Corollary 3.3.6.

Lemma 3.3.13 demonstrates the application of up-to techniques for establishing bisimilarity of $\pi Z$ processes.

Lemma 3.3.13. $\mu x. P \mid x \sim (\mu x. P \mid x) \mid (\mu x. P \mid x)$

Proof. Let $\mathcal{R} = \{(\mu x. P \mid x, (\mu x. P \mid x)) \mid (\mu x. P \mid x) \mid P \text{ is any process}\}$. Let $\mathcal{F}_1 = \mathcal{F}_{=}\mathcal{F}_{es}\mathcal{F}_{=}$.

$\mathcal{F}_1$ is quasi-safe since the constituent functions are quasi-safe and monotonous [80]. The result follows if we show that $\mathcal{R} \sim \mathcal{F}_1(\mathcal{R})$. Essentially we are trying to show that if $Q \mathcal{R} R$ and $Q \xrightarrow{\mu} Q'$ then $R \xrightarrow{\mu} R'$ such that $Q' \equiv E[Q]$ and $R' \equiv E[R]$ for some multi-hole context $E$.

If $P$ is a process and $\mu x. P \mid x \xrightarrow{\mu} P \mid \mu x. P \mid x$ (the only action possible since $P$ has no free variables), then we have the matching transition, $(\mu x. P \mid x) \mid (\mu x. P \mid x) \xrightarrow{\mu} P \mid (\mu x. P \mid x) \mid (\mu x. P \mid x)$. Let $E$ be the context $P \mid \_$. Then we have $\mu x. P \mid x \xrightarrow{\mu} E[\mu x. P \mid x]$ and $(\mu x. P \mid x) \mid (\mu x. P \mid x) \xrightarrow{\mu} E[(\mu x. P \mid x) \mid (\mu x. P \mid x)]$. The cases in the reverse direction can also be similarly shown.

3.4 Data refinement and Process Simulation

In this section we discuss how data refinement induces a process simulation. Definition 10 diagrammatically summarizes the notion of upward simulation as the criterion for data refinement [96]. The meaning of this diagram is as follows: if $\mathcal{U}$ is an upward simulation relating the states of abstract types $D$ and $A$, then a move by $D$ can be matched by $A$ up to the upward simulation, provided their starting states are related. We assume that the operations are total, i.e., their preconditions evaluate to true. Upward simulation is also called $L^{-1}$ simulation. Although it is sound, it is not a complete criterion for data refinement. Other criteria include $L$-simulation, $U$-simulation, and $U^{-1}$-simulation [26]. We consider upward simulation in this paper; results for other forms of simulations can be similarly established.
In Definition 10, \( \mathcal{I}^A(p) \) (respectively, \( \mathcal{I}^D(p) \)) stands for the relational interpretation of operation \( p \) for abstract type \( A \) (abstract type \( D \)). Moreover, \( \mathcal{I}^A(p)[\tilde{w},\tilde{v}] \) \( (\mathcal{I}^D(p)[\tilde{w},\tilde{v}]) \) stands for the binary relation on \( S(A) \) \( (S(D)) \) that we obtain by fixing input values \( \tilde{w} \) and output values \( \tilde{v} \) for \( \mathcal{I}^A(p) \) \( (\mathcal{I}^D(p)) \). Essentially, \( \mathcal{I}^A(p)[\tilde{w},\tilde{v}] \) \( (\mathcal{I}^D(p)[\tilde{w},\tilde{v}]) \) relates before-state and after-state for the operation \( p \) of abstract type \( A \) \( (D) \), if we supply the input values \( \tilde{w} \) and expect the output values \( \tilde{v} \). Since \( \pi Z \) does not use initialization schemas, the initialization condition for the upward simulation that any initial state of \( D \) must be related to some initial state of \( A \) is not specified in Definition 10. Instead, this condition is implicitly treated when we consider the process relation induced by data refinement (Definition 12).

**Definition 10** (Upward simulation). Given two abstract datatypes \( A \) and \( D \) with identical operation signatures and having state spaces \( S(A) \) and \( S(D) \), \( D \) refines \( A \) if there exists a relation \( \mathcal{U} \subseteq S(D) \times S(A) \), called upward simulation, such that the following condition holds for each operation \( p \) belonging to abstract types \( D \) and \( A \), and possible input (output) values \( \tilde{w} \) \( (\tilde{v}) \), where \( W_D, W'_D \in S(D) \) and \( W_A, W'_A \in S(A) \).

\[
W_D \quad \mathcal{U} \quad W_A
\]
\[
\mathcal{I}^D(p)[\tilde{w},\tilde{v}] \quad \text{implies} \quad \mathcal{I}^A(p)[\tilde{w},\tilde{v}]
\]
\[
W'_D \quad \mathcal{U} \quad W'_A
\]

Definition 11 recapitulates the definition of process simulation and similarity [79]. If \( P \preceq Q \), we say that \( Q \) simulates \( P \) or that \( P \) is simulated by \( Q \). As in the case of bisimilarity, in order to show that \( P \preceq Q \), we may exhibit a simulation containing \((P, Q)\).

**Definition 11** (Simulation and Similarity). A binary relation \( \mathcal{R} \) on processes is a simulation if whenever \( P \mathcal{R} Q \): \( P \xrightarrow{\mu} P' \) implies \( Q \xrightarrow{\mu} Q' \) and \( P' \mathcal{R} Q' \). Similarity \( (\preceq) \) is the largest simulation.

Data refinement induces a refinement relation on processes as given in Definition 12. We show that this relation is indeed a process simulation in Theorem 3.4.1.

**Definition 12** (Refinement relation on processes). The refinement relation on processes \( (\succeq) \) is defined inductively as follows:
1. \( a(W_D) \cup D \leq a(W_A) \cup A \), if there exists an upward simulation \( U \) between abstract types \( D \) and \( A \), and \( W_D U W_A \).

2. if \( P_D \leq P_A \) and \( Q_D \leq Q_A \), then \( P_D \mid Q_D \leq P_A \mid Q_A \).

3. if \( P_D \leq P_A \), then \( \nu n P_D \leq \nu n P_A \).

4. if \( P_D \leq P_A \), then \( \mu x.(P_D \mid x) \leq \mu x.(P_A \mid x) \).

5. \( P \leq P \) for any process \( P \).

**Theorem 3.4.1.** If \( P \leq Q \), then \( P \preceq Q \).

**Proof.** We need to show that \( \preceq \) is a simulation, i.e., if \( P \leq Q \), and \( P \vdash P' \), then \( Q \vdash Q' \) such that \( P' \leq Q' \). If \( \preceq \) is a simulation, since similarity (\( \simeq \)) is the largest simulation, we have that \( \preceq \subseteq \simeq \) and we have the result. We proceed by induction on the inference that \( P \leq Q \).

Base case: \( a(W_D) \cup D \leq a(W_A) \cup A \) such that \( W_D U W_A \), where \( U \) is an upward simulation between abstract types \( D \) and \( A \). The only applicable transition rule is \( L-ST \), which we may apply to the lhs to get, \( a(W_D) \cup D \xrightarrow{\Pi p(\bar{w}), \mu y p(\bar{v})} a(W_D') \cup D' \), where \( (W_D, \bar{w}, W_D', \bar{v}) \in \mathcal{I}^D(p) \), or, \( (W_D, W_D') \in \mathcal{I}^D(p)[\bar{w}, \bar{v}] \). By Definition 10 and by the definition of \( L-ST \) rule, the rhs may match this transition with \( a(W_A) \cup A \xrightarrow{\Pi p(\bar{w}), \mu y p(\bar{v})} a(W_A') \cup A' \), where \( (W_A, W_A') \in \mathcal{I}^A(p)[\bar{w}, \bar{v}] \) such that \( W_D U W_A \). Since \( W_D U W_A \), we have: \( a(W_D') \cup D \leq a(W_A') \cup A \). The second base case: \( P \leq P \) is trivially true.

We consider a typical inductive case. Let \( P_D \leq P_A \) and \( Q_D \leq Q_A \). By the inductive hypothesis, if \( P_D \xrightarrow{\mu} P_D' \) then \( P_A \xrightarrow{\mu} P_A' \) such that \( P_D' \leq P_A' \), and, if \( Q_D \xrightarrow{\nu n} Q_D' \) then \( Q_A \xrightarrow{\nu n} Q_A' \) such that \( Q_D' \leq Q_A' \). Transitions for process \( P_D \mid Q_D \) may result from transitions from \( P_D \) or \( Q_D \) alone or from an interaction of transitions from both \( P_D \) and \( Q_D \). We consider the latter case. The applicable transition rules are \( L-ST-OUT \), \( L-ST-IN \), \( L-IN-OUT \), \( L-OP-1 \), \( L-OP-2 \), and \( L-OP-3 \). In all these cases, if \( P_D \mid Q_D \xrightarrow{\mu} R \), then \( R \) has the general structure \( \nu n P_D' \mid Q_D' \) (\( \nu n \) could be empty) such that \( P_D \xrightarrow{\mu} P_D' \) and \( Q_D \xrightarrow{\nu n} Q_D', \) for some appropriate transition labels \( \mu_1 \) and \( \mu_2 \) whose interaction results in transition with label \( \mu \). By applying the inductive hypothesis, this transition can be matched with \( P_A \mid Q_A \xrightarrow{\mu} \nu n P_A' \mid Q_A' \), such
that $P'_D \preceq P'_A$ and $Q'_D \preceq Q'_A$. By applying rules (2) and (3) from Definition 12, we have $\nu\bar{n} P'_D | Q'_D \preceq \nu\bar{n} P'_A | Q'_A$. Hence the result.

\section{\(\pi\mathbb{Z}\) Labelled Transition Rules}

In this section we present the complete labelled transition rules for \(\pi\mathbb{Z}\). The primary difficulty in defining labelled semantics for \(\pi\mathbb{Z}\) lies in defining transitions for the state-holding term. In \(\pi\)-calculi, transition labels correspond to actions performed by a term. Taking this approach, if we label a state-holding term’s transitions with before-state and after-state, then bisimilarity induced by the labelled transitions would become dependent on state representation and thereby too discriminating. Moreover, data refinement, which refines the state structure of abstract types, will not have any interesting impact on processes. In order to prevent the state information from entering transition labels, we have specified labels for state-holding term’s transitions as a pair of action obligations, i.e., a pair of input and output actions that the environment must execute for the transition to take place. The use of action obligations, which is novel from \(\pi\)-calculus viewpoint, results in a bisimilarity that does not depend on the internal state of stateful channels. Moreover, scope extrusion from the state-holding term to the input term is minimized; only restricted names appearing in the input action obligation need to be extruded to the input side.
Figure 3.8: πZ Labeled Transitions (Rule Set 1 of 3). Symmetric versions of L-ST-IN and L-IN-OUT are not shown.
\[ L-OP-1: \quad \frac{P \xrightarrow{(\nu \bar{m}) \alpha' P(\overline{v})} P' \quad Q \xrightarrow{\alpha' P(\overline{v})} Q'}{P \mid Q \xrightarrow{\nu \bar{m}} P' \mid Q'} \quad \bar{m} \cap fn(Q) = \emptyset \]

\[ L-OP-2: \quad \frac{P \xrightarrow{\nu P(\overline{v})} P' \quad Q \xrightarrow{(\nu \bar{m}) \Pi P(\overline{w})} Q'}{P \mid Q \xrightarrow{\nu \bar{m}} P' \mid Q'} \quad \bar{n} \cap fn(P) = \emptyset \]

\[ L-OP-3: \quad \frac{P \xrightarrow{\nu P(\overline{v}), (\nu \bar{m}) \alpha' P(v)} P' \quad Q \xrightarrow{(\nu \bar{m}) \Pi P(\overline{w}), \alpha' P(\overline{v})} Q'}{P \mid Q \xrightarrow{\nu \bar{m}, \bar{n}} P' \mid Q'} \quad \bar{n} \cap fn(P) = \emptyset, \bar{m} \cap fn(Q) = \emptyset \]

\[ L-RES: \quad \frac{P \overset{\mu}{\rightarrow} P'}{\nu m P \overset{\mu}{\rightarrow} \nu m P'} \quad m \notin n(\mu) \]

\[ L-PAR: \quad \frac{P \overset{\mu}{\rightarrow} P'}{P \mid Q \overset{\mu}{\rightarrow} P' \mid Q} \quad bn(\mu) \cap fn(Q) = \emptyset \]

\[ L-SUM: \quad \frac{\alpha.P \overset{\mu}{\rightarrow} P'}{\alpha.P + Q \overset{\mu}{\rightarrow} P'} \quad \alpha \text{ is an input or output action.} \]

\[ L-EQ \quad \frac{\text{if } v = v \text{ then } P \text{ else } Q \overset{\mu}{\rightarrow} P}{v_1 \neq v_2} \]

\[ L-NEQ \quad \frac{\text{if } v_1 = v_2 \text{ then } P \text{ else } Q \overset{\mu}{\rightarrow} Q}{v_1 \neq v_2} \]

\[ L-REC \quad \frac{\mu x.P \overset{\mu}{\rightarrow} P[\mu x.P/x]}{} \]

Figure 3.9: $\pi Z$ Labelled Transitions (Rule Set 2 of 3). Symmetric versions of L-OP-1, L-OP-2, and L-OP-3 are not shown.

\[ \]

\[ \]

\[ \]

Figure 3.10: $\pi Z$ Labelled Transitions (Rule Set 3 of 3). Symmetric versions of L-PAR and L-SUM are not shown.