Supplementary Material to: Fuzzing Channel-based Concurrency Runtimes Using Types and Effects

QUENTIN STIÉVENART, Vrije Universiteit Brussel, Belgium
MAGNUS MADSEN, Aarhus University, Denmark

1 PROOFS

1.1 Definitions

We refer the reader to Section 4 of the paper for the definition of final configurations, terminated process, terminating configuration, and terminating effect. We do require one more definition before we proceed:

Definition 1.1 (Spawned-By). Consider $e : \tau \,\&\, \varphi$, an expression with effect φ, where each spawn effect is assigned a unique identifier. Let us denote the set of the process identifiers that appear in φ as $\Delta(\varphi)$.

$$\Delta(\epsilon) = \emptyset \qquad \Delta(\textsc{Put}(c)) = \emptyset \qquad \Delta(\textsc{Get}(c)) = \emptyset \qquad \Delta(\varphi_1 + \varphi_2) = \Delta(\varphi_1) \cup \Delta(\varphi_2)$$

$$\Delta(\varphi_1 ; \varphi_2) = \Delta(\varphi_1) \cup \Delta(\varphi_2) \qquad \Delta(\textsc{Spawn}^p(\varphi)) = \{p\} \cup \Delta(\varphi) \qquad \Delta(\varphi_1^{sr} \oplus \varphi_2^{sr}) = \Delta(\varphi_1^{sr}) \cup \Delta(\varphi_2^{sr})$$

$$\Delta(\textsc{SelGet}(\_, \varphi)) = \Delta(\varphi) \qquad \Delta(\textsc{SelPut}(\_, \varphi)) = \Delta(\varphi)$$

Definition 1.2 (Partially Terminated Predicate). The predicate $\mathcal{T}$ is defined as follows and holds if, when P is final, then all processes in ps are terminated.

$$\mathcal{T} : \ \textit{Configuration} \times \mathcal{P}(\textit{ProcessId}) \rightarrow \textit{Bool}$$
$$\mathcal{T}(P, \text{ps}) \text{ holds if } \textsc{Final}(P) \implies \forall p \in \text{ps}, P(p) \in \text{Value}$$

1.2 Main Theorems from the Paper

Theorem 1.3 (Generated Effect is Terminating). *If $\varphi \in$ Generation, then φ is terminating.*

Proof. This is a direct consequence of Lemma 1.5, with $P = []$.

Theorem 1.4 (Rewrite Rules Preserves Termination). *If φ is terminating, applying any rewrite rule (expansion or reordering) to a sub-effect of φ or φ itself preserves termination.*

Proof. This is a direct consequence of Lemma 1.8 for expansion, Lemma 1.9 for reordering, Lemma **??** *for simplification.*

Authors' addresses: Quentin Stiévenart, Vrije Universiteit Brussel, Pleinlaan 2, Brussels, Belgium, quentin.stievenart@vub.be; Magnus Madsen, Aarhus University, Åbogade 34, Aarhus, Denmark, magnusm@cs.au.dk.

XXXX-XXXX/2020/10-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1.3 Generation Ensures Termination

Lemma 1.5 (Main Theorem, Rephrased). *If $e : \tau \,\&\, \varphi$ where $\varphi \in$ Generation, e is closed, P is a set of processes, p_0 is a fresh process, and $P[p_0 : e] \Rightarrow^\star P'$ then $\mathcal{T}(P', \Delta(\varphi) \cup p_0)$.*

Proof. By induction on the derivation of $\varphi \in$ Generation. We make the assumption that channel names are unique and don't overlap when generated from different rules.

Case (G-Final). We know that the effect is ϵ. We must show that for all expressions e that have effect ϵ it must be the case that:

$$P[p_0 : e] \Rightarrow^\star P' \implies \mathcal{T}(P', \Delta(\epsilon) \cup \{p_0\})$$

We know that $\Delta(\epsilon) = \emptyset$, hence we have to show that $P'(p_0) \in$ Value. By inversion, there are four subcases to consider.

Subcase: e $= x$. Impossible, because e is closed.

Subcase: e = (). Because p_0 is final, it cannot progress further and $P'(p_0) =$ (), which is a value.

Subcase: e = true. Because p_0 is final, it cannot progress further and $P'(p_0) =$ true, which is a value.

Subcase: e = false. Because p_0 is final, it cannot progress further and $P'(p_0) =$ false, which is a value.

Case (G-Seq). We know that the effect is $\varphi_1; \varphi_2$, where both φ_1 and φ_2 are known to be terminating effects, and the channels used by φ_1 and φ_2 do not overlap. By inversion of the effect typing, we know that expression of the effect must be of the form:

$$e_1; e_2$$

where both e_1 and e_2 are known to be terminating. We must show that for all such expressions, it must be the case that :

$$P[p_0 : e_1; e_2] \Rightarrow^\star P' \implies \mathcal{T}(P', \Delta(\varphi_1) \cup \Delta(\varphi_2))$$

We know that e_1 is terminating, hence we know that the configuration will always be reduced to a configuration of the following shape:

$$P''[p_0 : v; e_2]$$

where $\mathcal{T}(P'', \Delta(\varphi_1))$. The evaluation rule (E-Process) and (E-Seq) can be applied, resulting in:

$$P'''[p_0 : e_2]$$

We know that e_2 is terminating (it has effect φ_2, which is terminating), and that P''' is such that $\mathcal{T}(P''', \Delta(\varphi_1)$. Hence, we know that $P'''[p_0 : e_2] \Rightarrow^\star P' \implies \mathcal{T}(P', \Delta(\varphi_1) \cup \Delta(\varphi_2))$.

Case (G-Choice). We know that the effect is $\varphi_1 + \varphi_2$, where both φ_1 and φ_2 are terminating. For the sake of simplicity and without loss of generality, we assume the effect to actually be $\epsilon; (\varphi_1 + \varphi_2)$. By inversion of the effect typing, we know that expressions with that effect must be of the form:

$$\mathit{if}\, v_0 \;\mathit{then}\, e_1 \;\mathit{else}\, e_2$$

Either v_0 is true and E-If-True can apply, or v_0 is false and E-If-False can apply. Consider a configuration evaluating this expression:

$$P[p_0 : \mathit{if}\, v_0 \;\mathit{then}\, e_1 \;\mathit{else}\, e_2]$$

With the rule E-Process, it can therefore reach any of the two following configurations:

$$P[p_0 : e_1] \quad \mathit{or} \quad P[p_0 : e_2]$$

which both terminate, as e_1 has effect φ_1, e_2 has effect φ_2, and both φ_1 and φ_2 are terminating by the induction hypothesis.

Case (G-Spawn). We know that the effect is of the form $\textsc{Spawn}(\varphi)$, and we know by the induction hypothesis that φ is a terminating effect, i.e.:

$$e : \tau \,\&\, \varphi \wedge P[p_0 : e] \Rightarrow^\star P' \implies \mathcal{T}(P', \Delta(\varphi) \cup \{p_0\})$$

We must show that (unfolding the definition of Δ)

$$\forall e'.e' : \tau \,\&\, \textsc{Spawn}^p(\varphi) \wedge P[p_0 : e'] \Rightarrow^\star P' \implies \mathcal{T}(P', \{p, p_0\})$$

By inversion on effects we know that the only expression of effect $\textsc{Spawn}(\varphi)$ is the spawn expression:

$$P[p_0 : \textbf{spawn}\, e] \Rightarrow^\star P' \implies \mathcal{T}(P', \Delta(\textsc{Spawn}(\varphi)^p) \cup \{p, p_0\})$$

Let us consider the configuration:

$$P[p_0 : \textbf{spawn}\, e]$$

for some process map P. In order for p_0 to progress, only the rule (E-Spawn) can be applied, resulting in:

$$P[p_0 : ()][p : e]$$

We see that p_0 has terminated. We can apply the induction hypothesis with $p_0 = p$ to get:

$$P[p_0 : ()][p : e] \Rightarrow^\star P' \implies \mathcal{T}(P', \Delta(\varphi) \cup \{p\})$$

We must show that p_0 is terminated in P' which is true because (a) p_0 was already terminated, and (b) $\Rightarrow$ preserves termination by Process Never Removed Lemma.

Case (G-PingPong). We will prove this case for a less generic version of the rule: $\textsc{Spawn}([\textsc{Get}(c)]_0^n); [\textsc{Put}(c)]_0^n$. It can be generalized to dual effects using exactly the same reasoning. The interleaved effects (denoted $\square$ in the paper) do not influence termination of the generated effect, as they are all assumed to be terminating. We will therefore ignore them in the proof for the sake of simplicity, and without loss of generality.

We need to show that $\textsc{Spawn}([\textsc{Get}(c)]_0^n); [\textsc{Put}(c)]_0^n$ is terminating. By inversion on the effect typing, and assuming – without loss of generality – that only unit values are communicated over channels, expressions with that effects are:

$$spawn^p(\leftarrow c; \ldots; \leftarrow c); c \leftarrow (); \ldots; c \leftarrow ()$$

where the number of put matches the number of get. Consider the following configuration.

$$P[p_0 : spawn^p(\leftarrow c; \ldots; \leftarrow c); c \leftarrow (); \ldots; c \leftarrow ()]$$

By applying the evaluation rule (E-Spawn), we have

$$P[p_0 :\leftarrow c; \ldots; \leftarrow c][p : c \leftarrow (); \ldots; c \leftarrow ()]$$

By induction on n, we can show that the rule (E-Sync) can be applied n times in order to reach the following configuration.

$$P[p_0 : ()][p : ()]$$

The base case for $n = 1$ is proven by directly applying (E-Sync) followed by (E-Seq) to $P[p_0 :\leftarrow c][p : c \leftarrow ()]$ in order to reach $P[p_0 : ()][p : ()]$. The inductive case is proven by first applying (E-Sync) followed by (E-Seq) to $P[p_0 : \underbrace{\leftarrow c; \cdots \leftarrow c}_{n}][p : \underbrace{c \leftarrow (); \ldots; c \leftarrow ()}_{n}]$, resulting in $P[p_0 :$

$\underbrace{\leftarrow c; \dots \leftarrow c}_{n-1}][p : \underbrace{c \leftarrow (); \dots; c \leftarrow ()}_{n-1}]$, *which itself results in* $P[p_0 : ()][p : ()]$ *by the induction hypothesis. From this resulting configuration, we have:*

$$P[p_0 : ()][p : ()] \Rightarrow^\star P' \implies \mathcal{T}(P', \{p, p_0\})$$

Case (G-FanOut). Again, without loss of generality, we ignore the interleaved effects (denoted $\square$ *in the paper), and we fix a direction of communication. Hence, we reason about the following effect:*

$$[\textsc{Spawn}(\textsc{Get}(c_i))]_0^n; [\textsc{Put}(c_i)]_0^n$$

By inversion of the effect typing, expressions with this effect are the following:

$$spawn(\leftarrow c_0); \dots; spawn(\leftarrow c_n); c_0 \leftarrow (); \dots; c_n \leftarrow ()$$

Consider the following configuration:

$$P[p_0 : spawn(\leftarrow c_0); \dots; spawn(\leftarrow c_n); c_0 \leftarrow (); \dots; c_n \leftarrow ()]$$

After n applications of the E-Spawn rule (and E-Process applied with E-Seq), we have the following configuration:

$$P[p_0 : c_0 \leftarrow (); \dots; c_n \leftarrow ()][p_1 :\leftarrow c_0] \dots [p_{n+1} :\leftarrow c_n]$$

At this point, the E-Sync rule can also be applied n times, in order to reach the following configuration:

$$P[p_0 : ()][p_1 : ()] \dots [p_{n+1} : ()]$$

And it is clear that we have:

$$P[p_0 : ()][p_1 : ()] \dots [p_{n+1} : ()] \Rightarrow^\star P' \implies \mathcal{T}(P', \{p_0, p_1 \dots p_{n+1}\})$$

Because $\Delta([\textsc{Spawn}(\textsc{Get}(c_i))]_0^n; [\textsc{Put}(c_i)]_0^n) = \{p_1 \dots p_{n+1}\}$*, this case holds.*

Case (G-Pipeline). The effect generated is the following:

$$\textsc{Spawn}(\textsc{Get}(c_0); \textsc{Put}(c_1)); \dots; \textsc{Spawn}(\textsc{Get}(c_{n-1}); \textsc{Put}(c_n)); \textsc{Put}(c_0); \textsc{Get}(c_n)$$

By inversion of the effect typing rule, an expression with that effect must be the following:

$$spawn \leftarrow c_0; c_1 \leftarrow (); \dots; spawn \leftarrow c_{n-1}; c_n \leftarrow (); c_0 \leftarrow (); \leftarrow c_n$$

Consider a configuration where p_0 *has this expression:*

$$P[p_0 : spawn \leftarrow c_0; c_1 \leftarrow (); \dots; spawn \leftarrow c_{n-1}; c_n \leftarrow (); c_0 \leftarrow (); \leftarrow c_n]$$

We can apply the (E-Spawn) rule n times to get the following configuration:

$$P[p_0 : c_0 \leftarrow (); \leftarrow c_n][p_1 :\leftarrow c_0; c_1 \leftarrow ()] \dots [p_{n-1} :\leftarrow c_{n-1}; c_n \leftarrow ()]$$

At this point, rule (E-Sync) can be applied to synchronize p_0 *with* p_1*, and we get (after applying (E-Process) with (E-Seq) on both* p_0 *and* p_1*):*

$$P[p_0 :\leftarrow c_n][p_1 : c_1 \leftarrow ()][p_2 :\leftarrow c_1; c_2 \leftarrow ()] \dots [p_{n-1} :\leftarrow c_{n-1}; c_n \leftarrow ()]$$

A similar reasoning can be applied until we reach:

$$P[p_0 :\leftarrow c_n][p_1 : ()] \dots [p_{n-1} : c_n \leftarrow ()]$$

and a final application of (E-Sync) results in:

$$P[p_0 : ()][p_1 : ()] \dots [p_{n-1} : ()]$$

where all processes p_0 *to* p_{n-1} *are terminated.*

Case (G-Select). We prove this case for $m = 1$. The extension to an arbitrary value for m is natural: because we prove that one communication over each channel is terminating, a sequence of m communications over each channel will also be terminating, because the generated pattern makes sure that each new sequence of communication starts when the previous one is finished. Similarly, we prove this case for a single branch in the select effect, without loss of generality: if the effect is terminating for a single branch of the ones that can be generated, it is also terminating for multiple branches generated in the same way. Again, we consider φ_i to be $\textsc{Get}(c_i)$, and φ_i to be $\textsc{Put}(c_i)$ for a matter of simplicity, without loss of generality. The generated effect is the following:

$$\textsc{Spawn}(\textsc{Get}(c_0)); \ldots; \textsc{Spawn}(\textsc{Get}(c_n)); \mathit{branch}(\mathit{shuffle}(\textsc{Put}(c_0); \ldots; \textsc{Put}(c_n)))$$

By inversion of the effect typing, the expressions with such an effect are the following:

$$\mathit{spawn}(\leftarrow c_0); \ldots; \mathit{spawn}(\leftarrow c_n); \mathit{select}\{\, \mathit{case}\, c'_0 \leftarrow () \Rightarrow c'_1 \leftarrow (); \ldots; c'_n \leftarrow ()\}$$

where $c'_0 \ldots c'_n$ is a permutation of $c_0 \ldots c_n$. Given a configuration where p_0 evaluates this expression, and after applying n times the rule (E-Spawn), we have the following configuration.

$$P[p_0 : \mathit{select}\{\, \mathit{case}\, c'_0 \leftarrow () \Rightarrow c'_1 \leftarrow (); \ldots; c'_n \leftarrow ()\}][p_1 :\leftarrow c_0] \ldots [p_{n+1} :\leftarrow c_n]$$

At this point, it is clear that the rule (E-Select-Put) can apply for channel c'_0, as all processes p_1 to p_{n+1} are trying to send over channels c_0 to c_n, and c'_0 is one of these channels. Assuming, without loss of generality, that c'_0 is c_0, we reach the following configuration:

$$P[p_0 : c'_1 \leftarrow (); \ldots; c'_n \leftarrow ()][p_1 : ()][p_2 :\leftarrow c_1] \ldots [p_{n+1} :\leftarrow c_n]$$

By the same reasoning, rule (E-Sync) can apply with channel c'_1, followed by (E-Sync) with channel c'_2, until channel c'_n. The resulting configuration is then the following.

$$P[p_0 : ()][p_1 : ()] \ldots [p_{n+1} : ()]$$

For which it is clear that all processes p_0 to p_{n+1} are terminated.

Lemma 1.6 (Process Never Removed Lemma). *If $P(p) = v$, then $\forall P'$ such that $P \Rightarrow^\star P'$, then $P'(p) = v$*

Proof. *Consider a configuration P with $P(p) = v$. There exists no rule that can be applied to process p, hence this process cannot be reduced further. Moreover, there exists no rule that remove a process from the process map. Hence, for any P' such that $P \Rightarrow^\star P'$, we have that $P'(p) = v$.*

Lemma 1.7 (Process-Stays-Terminated). *If P is a configuration, p is a process that is terminated in P, and $P \Rightarrow P'$ then p is terminated with the same value in P'.*

Proof. *By inversion on rules that can be applied to values in a process. (There are none).*

1.4 Expansion Preserves Termination

Lemma 1.8 (Expansion Preserves Termination). *If φ is a terminating effect, and $\varphi \to_E \varphi'$, then φ' is terminating. That is, for all $e : \tau \,\&\, \varphi'$, we have that if $[p_0 : e] \Rightarrow^\star P$ where P is final, then P is terminated.*

Proof. *We prove this by a case analysis on the rule applied. There are five cases to consider.*

Case (E-Choice-Dup). The rewrite rule is

$$\varphi \longrightarrow_E \varphi + \varphi$$

We know that for any expression $e : \tau \mathbin{\&} \varphi$, e must terminate. By inversion of the effect typing, we know that the expression of the rewritten effect must be of the form (for simplicity and without loss of generality, we assume that $\varphi + \varphi$ is equivalent to $\epsilon; (\varphi + \varphi)$):

$$if\, v_0 \; then\, e_1 \; else\, e_2$$

where $v_0 : \tau \mathbin{\&} \epsilon$, $e_1 : \tau \mathbin{\&} \varphi$, and $e_2 : \tau \mathbin{\&} \varphi$.

Given a configuration where p_0:

$$[p_0 : if\, v_0 \; then\, e_1 \; else\, e_2, \cdots]$$

It can take a step with E-Process and either E-If-True or E-If-False, resulting in one of the following configurations

$$[p_0 : e_1, \cdots], [p_0 : e_2, \cdots]$$

which both terminate, as e_1 and e_2 have as effect φ, which is a terminating effect.

Case (E-Select-Dup). We prove this case for $i = 1$. The reasoning is applicable mutatis mutandis *for $i = 2$. The rewrite rule is*

$$\varphi_1^{sr} \oplus \varphi_2^{sr} \longrightarrow_E \varphi_1^{sr} \oplus \varphi_1^{sr} \oplus \varphi_2^{sr}$$

By inversion of the effect typing, we know that the expression of the original effect must be of the form:

$$e = select\,\{\, case\, x \leftarrow c_1 \Rightarrow e_1, case\, y \leftarrow c_2 \Rightarrow e_2\}$$

Given some configuration, which eventually terminates, where the above expression appears in an evaluation context:

$$[p_0 : E^1[e], \cdots]$$

then for p_0 to terminate, it must be the case that the (E-Select) evaluation rule is applied at some point. The configuration at that point must be of the form:

$$[p_0 : E^1[e], P_1, \cdots, p_i : E^2[c_j \leftarrow ()], \cdots P_m]$$

with $j \in \{1, 2\}$. Applying (E-Select) then results in the following configurations.

$$[p_0 : E^1[e_j], P_1, \cdots, p_i : E^2[()], \cdots P_m]$$

which must terminate.

By inversion of the effect typing, the expression of the rewritten effect must be of the form:

$$e' = select\,\{\, case\, x \leftarrow c_1 \Rightarrow e'_1, case\, y \leftarrow c_1 \Rightarrow e'_2, case\, z \leftarrow c_2 \Rightarrow e'_3\}$$

Following the same reasoning as for e, we have the following configuration on which (E-Select) can be applied:

$$[p_0 : E^1[e'], P_1, \cdots, p_i : E^2[c_j \leftarrow ()], \cdots P_m]$$

with $j \in \{1, 2\}$. With $j = 1$, by applying (E-Select) we reach:

$$[p_0 : E^1[e'_1], P_1, \cdots, p_i : E^2[()], \cdots P_m]$$

which terminates because $e'_1 : \tau \mathbin{\&} \varphi_1$ where φ_1 is guaranteed to terminate. With $j = 2$, by applying (E-Select) we reach one of the following configurations:

$$[p_0 : E^1[e'_2], P_1, \cdots, p_i : E^2[()], \cdots P_m], [p_0 : E^1[e'_3], P_1, \cdots, p_i : E^2[()], \cdots P_m]$$

which terminates because $e'_2 : \tau \mathbin{\&} \varphi_2$ and $e'_3 : \tau \mathbin{\&} \varphi_2$ where φ_2 is guaranteed to terminate.

Case (E-GET-SELECT). The rewrite rule is:

$$\textsc{Get}(c) \longrightarrow (\textsc{Get}(c); \epsilon) \oplus (\textsc{Get}(c); \epsilon)$$

By inversion of the effect typing, we know that the expression of the original effect must be of the form:

$$\leftarrow c$$

for some channel c.

Given some configuration, which eventually terminates, where the above expression appears in an evaluation context:

$$[p_0 \mapsto E^1[\leftarrow c], P \cdots]$$

then for p_0 to terminate, it must be the case that the (E-SYNC) evaluation rule is applied at some point. The configuration at this point must be of the form:

$$[p_0 \mapsto E^1[\leftarrow c], P_1, \cdots, P_m]$$

where there must be a process P_i whose evaluation context is performing a put operation. Once this happens, we reach a configuration of the form:

$$[p_0 \mapsto E^1[()], P_1, \cdots, P_i \mapsto E^2[()], \cdots, P_n] \tag{$\star$}$$

which must terminate.

By inversion of the effect typing, the expression of the rewritten effect must be of the form:

$$e_0 = select\{case\, x \leftarrow c \Rightarrow (), case\, x \leftarrow c \Rightarrow ()\}$$

for some variables x and y.

Consider the configuration the above expression appears in an evaluation context:

$$[p_0 \mapsto E^1[e_0], P \cdots]$$

then we know that the original effect could reach the same configuration and this configuration must be able to evaluate to another configuration where there exists a process P_i performing a put on the channel c. Thus, by (E-SELECT), this configuration can take a step to another configuration of the form:

$$[p_0 \mapsto E^1[()], P_1, \cdots, P_i \mapsto E^2[()], \cdots, P_n] \tag{$\star$}$$

which is configuration reachable the by original overall effect, and hence the evaluation must terminate.

In the case where there exists multiple processes ready to perform a put operation on the channel c any of them can be chosen both in the original (overall) expression and in the rewritten (overall) expression.

Case (E-PUT-SELECT). The reasoning of the (E-GET-SELECT) case is applicable mutatis mutandis *to this case.*

Case (E-SEQ). The rewrite rule is $\varphi \rightarrow_E \varphi_1; \varphi; \varphi_2$, where φ, φ_1, and φ_2 are terminating effects. As shown in the proof of Lemma 1.5 for the (G-SEQ) case, sequencing terminating effects results in a terminating effect, and hence $\varphi_1; \varphi; \varphi_2$ is itself a terminating effect.

1.5 Reordering Preserves Termination

LEMMA 1.9 (REORDERING PRESERVES TERMINATION). *If φ is a terminating effect, and $\varphi \rightarrow_R \varphi'$, then φ' is terminating. That is, for all $e : \tau \,\&\, \varphi'$, we have that if $[p_0 : e] \Rightarrow^\star P$ where P is final, then P is terminated.*

Proof. We prove this by case analysis on the rule applied. There are four cases to consider.

Case (R-Select). We have $\varphi = \varphi_1^{sr} \oplus \varphi_2^{sr}$, and $\varphi' = \varphi_2^{sr} \oplus \varphi_1^{sr}$. By inversion of the effect typing, we know that the expression of the original effect φ must be of the form (limiting ourselves to get effects, the reasoning is identical for put effects):

$$select\,\{case\,x := c_1 \Rightarrow e_1, case\,y := c_2 \Rightarrow e_2\}$$

for some variables x and y where the effects of e_1 and e_2 are φ_1 and φ_2, respectively. Similarly, by inversion of the effect typing, the expression of the rewritten effect φ' must be of the form:

$$select\,\{case\,y := c_2 \Rightarrow e_2', case\,x := c_1 \Rightarrow e_1'\}$$

The (E-Select) rule is the only evaluation rule applicable to each expression. But the order of cases in (E-Select) is immaterial. Thus, if the original effect terminates, the effects φ_1 and φ_2 terminate, and the rewritten effect must also terminate.

Case (R-ChoiceSelectGetGet). We have $\varphi = (\textsc{Get}(c_1); \varphi_1)+(\textsc{Get}(c_2); \varphi_2)$ and $\varphi' = (\textsc{SelGet}(c_1, \varphi_1))\oplus (\textsc{SelGet}(c_2, \varphi_2))$. By inversion of the effect typing, the expression of the original effect is of the form:

$$if\,e\ then\,(\leftarrow c_1; e_1)\ else\,(\leftarrow c_2; e_2)$$

where the expression e has type bool (for simplicity, we ignore the effect of e, but it must terminate), and the expressions e_1 and e_2 have effects φ_1 and φ_2, respectively. For this expression to reach a terminated configuration, it must first take a step by (E-If-True) or (E-If-False). Next, either of the two sub-expressions: $\leftarrow c_1; e_1$ or $\leftarrow c_2; e_2$ must take a step. The only applicable evaluation rule is (E-Sync), hence there must exist some other process p which is ready to perform a put operation.

By inversion of the effect typing, the expression of the rewritten effect must be of the form:

$$select\,\{case\,x := c_1 \Rightarrow e_1, case\,y := c_2 \Rightarrow e_2\}$$

for some variables x and y. The only applicable evaluation rule is (E-Select) which requires that another process to perform a put operation on either c_1 or c_2, and we know that such a process must exist for original effect to terminate.

Case (R-Spawn-1). We have $\varphi = \textsc{Spawn}(\varphi_1); \textsc{Spawn}(\varphi_2)$ and $\varphi' = \textsc{Spawn}(\varphi_2); \textsc{Spawn}(\varphi_1)$. By inversion of the effect typing, we know that the expressions of the two effects are of the form:

$$spawn(e_1); spawn(e_1)$$

and

$$spawn(e_2); spawn(e_1)$$

If we consider the configuration:

$$[p_0 \mapsto spawn(e_2); spawn(e_1)]$$

It can take a step by (E-Spawn) to:

$$[p_0 \mapsto (); spawn(e_1), p_1 \mapsto e_2]$$

At this point either p_0 or p_1 can take a step. In either case, possibly through multiple steps, we must reach a configuration of the form:

$$[p_0 \mapsto spawn(e_1), p_1 \mapsto e_2', \Delta(\varphi_2)]$$

where e_2' is some reduction of e_2 and $\Delta(\varphi_2)$ are some processes spawned by e_2. This configuration, possibly throught multiple steps, must reach a configuration of the form:

$$[p_0 \mapsto (), p_1 \mapsto e_2'', p_2 = e_1, \Delta(\varphi_2)']$$

where e_2'' is some reduction of e_2' and $\Delta(\varphi_2)'$ are some processes spawned by e_2 and e_2'. But this configuration is reachable by the original configuration and hence must terminate! Its corresponds to the case where e_1 is spawned, e_2 is spawned, and only the process e_2 has taken some steps.

Case (R-Spawn-2). We have $\varphi = \textsc{Spawn}(\varphi_1); \textsc{Spawn}(\varphi_2)$ and $\varphi' = \textsc{Spawn}(\textsc{Spawn}(\varphi_2); \varphi_1)$. By inversion of the effect typing, we know that the expressions of the two effects are of the forms:

$$spawn(e_1); spawn(e_2)$$

and

$$spawn(spawn(e_2); e_1)$$

If we consider the configuration:

$$[p_0 \mapsto spawn(spawn(e_2); e_1)]$$

It can take a step by (E-Spawn) to:

$$[p_0 \mapsto (), p_1 \mapsto spawn(e_2); e_1]$$

This, again by (E-Spawn), can take a step to:

$$[p_0 \mapsto (), p_1 \mapsto (); e_1, p_2 \mapsto spawn(e_2)]$$

At this point either p_1 or p_2 could take a step.

If p_1 takes a step, we reach:

$$[p_0 \mapsto (), p_1 \mapsto e_1, p_2 \mapsto spawn(e_2)]$$

but this configuration is reachable by the original effect and hence must terminate!

If, on the other hand, p_2 takes one or more steps, we reach:

$$[p_0 \mapsto (), p_1 \mapsto (); e_1, p_2 \mapsto e_2', \Delta(\varphi_2)]$$

where e_2' is some reduction of e_2 and $\Delta(\varphi_2)$ are the processes spawned by φ_2. At some point p_1 takes a step to reach:

$$[p_0 \mapsto (), p_1 \mapsto e_1, p_2 \mapsto e_2'', \Delta(\varphi_2)']$$

where e_2'' is some reduction of e_2' and $\Delta(\varphi_2)'$ is the reduction of the processes $\Delta(\varphi_2)$. But this configuration is reachable by the original effect and hence must terminate! It corresponds to the case where e_1 is spawned, but does not run until e_2 is spawned and has run for a while.