NORTHWESTERN UNIVERSITY

A Constructive Calculus for Esterel

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Computer Science

By

Spencer P. Florence

EVANSTON, ILLINOIS

September 2020

ABSTRACT

A Constructive Calculus for Esterel

Spencer P. Florence

The language Esterel has found success in many safety-critical applications, from aircraft landing gear to digital signal processors. Its unique combination of powerful control operations, deterministic concurrency, and real time execution bounds are indispensable to programmer in these kinds of safety-critical domains. However these features lead to an interesting facet of the language, called Constructivity.

Constructivity is a non-local property of Esterel programs which makes defining semantics for the language subtle. Existing semantics tend to sacrifice some desirable facet of a language semantics to handle this. Many sacrifice locality, and only work on whole programs. Some sacrifice adequacy, allowing them to describe transformations to programs at the cost of being able to actually run programs. Still more decide to work in a domain other than Esterel, such as circuits, making Constructivity easier to capture, but forcing users of these semantics to reason in a domain which they are not programming in.

This dissertation provides the first semantics for Esterel which captures all of the above facets, while still describing Constructivity.

Acknowledgments

These acknowledgements can by no means be complete. Completing a PhD is a challenging prospect, and so to list every name would be to create a list of names akin to book 3 of the Illiad, and no one wants that. So here are some highlights.

A huge thanks to my advisor, Robby, who stuck with me as through this long process. Thank you, Vincent, for providing advice, a sounding board, and a sympathetic ear during these last five years. Thank you, Dan, for making the lab so much more fun to hang out in. Thank you, Ethan, for all the fascinating conversations.

Thank you, Shu-hung, for the tremendous effort that made this project possible. Thank you, Matthias, for setting me on this path and for suggesting this project. That you, Steve and Bill, for all of the conversations and advise during the start of my PhD. Thank you, Christos, for all your help in guiding the math part of this project.

Thank you, Wendy, Ryan, Jonathan, Teresa, Pranav, for being such steadfast friends through all of this, and for giving me an escape when it was needed.

Table of Contents

CHAPTER 1

An Introduction

The language Esterel has found success in many safety-critical applications. It has been used in the creation and verification of the maintenance and test computer, landing gear control computer, and virtual display systems of civilian and military aircraft at Dassault Aviation (Berry et al. 2000) and the specification of part of digital signal processors at Texas Instruments (Benveniste et al. 2002).

Its success can partially be attributed to how its computational model is radically different from other languages. It gives the programmer the ability to use non-local communication mechanisms to coordinate powerful non-local control (like suspension or preemption of whole groups of threads) while maintaining deterministic concurrency. This non-local nature of evaluation leads to unexpected situations. For example the choice of which branch a conditional takes may immediately affect the choice another conditional makes in a different part of the program, without any explicit communication between those parts of the program. The selection of the other branch may render the entire program invalid. This powerful and unique evaluation model makes giving a formal semantics to Esterel a subtle and tricky business, and has lead to a plethora of different semantics suited to different purposes.

Some of these semantics are computationally adequate, giving an evaluator for programs, giving meaning to full programs by running them—such as the Constructive Operational Semantics (COS) (Potop-Butucaru 2002), and the State Behavioral Semantics (SBS) (Berry 2002). Others are allow for compositional, modular reasoning about fragments of full programs (i.e. constant propagation or modular compilation)—such as the Circuit Semantics (Berry 2002) or the Axiomatic Semantics (AS) (Tini 2001). Still others give syntactic reasoning, which reason about programs directly using their syntax, without going through an external domain—Such as the COS, SBS, and AS. This allows for more direct communication with programmers in the domain they already understand. This is useful, for example, when giving good crash reports, explaining program refactorings, or for optimization coaching (St-Amour et al. 2012), which helps explain to programmers why some optimization were not applied and how to fix it. Only one prior semantics

is an equational theory, the AS, which allows one to reason about program fragments like one does terms in algrabra. All of the existing semantics are (and must be) consistent and sound, in that they all describe the exact same language, as opposed to subtly different variations on that language.

Each of Esterel's many semantics do some of these jobs very well. However there is a gap in the these semantics: there are no equational theories for Esterel which are simultaneously consistent, sound, and computationally adequate. Such a semantics is the contribution of this dissertation.

I have shown that my equational theory is consistent, sound, and adequate. I show this using three pieces of evidence: proofs, testing, and prior work. These flavors of evidence is necessary because not all parts of the calculus have proofs for them. The proofs apply only to loop free, pure Esterel programs, and are proven with respect to the circuit semantics for Esterel (Berry 2002). The full calculus, on the other hand, is tested against several different Esterel semantics and implementations. Many parts of the calculus are also borrowed from the prior semantics, helping increase confidence in their correctness.

Equational. Equational theories allow for algebra-like reasoning about programs: that is they can explain why fragments of programs are equal using only the syntax of those program fragments.

The benefits of using only the syntax of the program fragments is primarily human: It allows reasoning about a programming language to be expressed directly in terms of that language, rather than in terms of some external domain. Often developing a semantics which uses *only* the syntax of a language is impractical, or even impossible. See, for instance, the σ and ϱ forms of the Felleisen and Hieb (1992) state calculus which do not appear directly in any language, or evaluation contexts (Felleisen and Friedman 1986) often used to describe non-local control operators (e.g. exceptions, continuations) which while described in terms of existing syntax cannot be written directly by a programmer. However, while these frameworks require extending the syntax of the language, they still map closely to the syntax of the surface language, and the extensions they use are minor and either can be mapped directly to the surface language syntax or require only minor annotations to the surface syntax. Therefore, even in the case of minor syntactic extensions, a syntactic semantics still allows for explanations of program transformations using the notation

users of that language are familiar with, rather than some external domain. In the end, this means that calculi are syntactic *by construction*.

In order to make my calculus sound and adequate I have added two new forms to the syntax of Kernel Esterel: a variant of Felleisen and Hieb (1992)'s ϱ, and a loop variant $\overline{\text{loop}}$. They are discussed in more detail in chapter 3 and chapter 5 respectively.

Reasoning about program fragments is useful for both human and machine reasoning. Reasoning about full programs is difficult, impractical, and in often impossible in the case of libraries. Modular reasoning is essential for working with large programs.

Consistent. Consistency is one of the most essential of these facets. A consistent semantics is one that does not allow contradictions to be derived: for example, by not allowing two programs to be proven equal if they evaluate to different values.

The Consistency of the calculus given by proof and by testing. Details may be found in section 4.4 and section 5.4.

Sound. Soundness is necessary for a semantics which describes an already established language. A sound semantics is one which agrees with an existing, ground truth semantics. In other words, a semantics which is not sound describes a *different* language that the one it is supposed to describe. Thus soundness, like consistency is essential for any semantics.

The soundness of the calculus is also given by proof and testing. Specifically it is proven with respect to the circuit semantics (Berry 2002), for pure, loop free, programs within a single instant. Evidence for the Soundness for multi-instant, loop containing programs is given by random testing. This is discussed more in section 4.2 and section 5.4.

Adequate. Adequacy describes the power of a semantics. If we take the word *semantics* to mean "something which allows for formal reasoning about a language",[1] then we can have semantics which allow for manipulations or transformations of a language, but cannot actually run a complete program. Such semantics are not *adequate* for describing an evaluator for a language. This is not ideal, as it means there is some aspect of the language the semantics does not describe. Therefore, to make sure a semantics has broad coverage of the aspects of a language, an adequate semantics is desirable.

Adequacy is also given by proof and testing. Like soundness, it is proven for pure, loop free, programs for one instant. Evidence for the Adequacy of loop containing programs with host language expressions across multiple instants is also given by random testing. This is discussed more in section 4.3 and section 5.4.

1.1. Overview

The dissertation is divided into six more chapters, and four appendices. Chapter 2 summarizes the background a reader will need to understand this document, as well as pointers to the background reading I assume the reader has an understanding of. Chapter 3 then describes the calculus I have designed on pure, loop free Esterel programs. Then chapter 4 gives the proofs for Consistency, Soundness, and Adequacy on this part of the calculus. Next chapter 5 gives the remainder of the calculus and describes the remainder of the evidence that the calculus is correct. Chapter 6 gives existing work related to my calculus. Finally, chapter 7 gives some final thoughts and future directions.

Appendix A lists definitions for all of the notation I use here. Appendix B gives the proofs of the theorems. Appendix C gives an overview of the implementation of a circuit solver I implemented for my proofs. Appendix D gives examples of using the calculus to prove equalities.

[1] There are many ways to define what *semantics* means. Literally, a semantics is that which gives meaning to a language, but that just shifts the question over to defining "meaning". Therefore, I am intentionally using a very broad definition.

CHAPTER 2

Background

This chapter provides the background material necessary to understand this dissertation. This chapter is meant as a refresher on the material, as well as an introduction to the notation I am going to be using. As such, each section has recommended reading, which the section is designed as a refresher for. Readers who are very familiar with the background work of each area may wish to skim these sections for the notation I use.

Section 2.1 describes the language Esterel. It is meant as refresher on Chapters 1, 2, 4, 5, 7 and 12 of *The Constructive Semantics of Pure Esterel (Draft Version 3)* (Berry 2002), as well as chapters 1 through 4 of *Compiling Esterel* (Potop-Butucaru et al. 2007). Specifically an understanding of Kernel Esterel and the Constructive Behavioral Semantics will be helpful. As my dissertation uses Kernel Esterel, this section only describes that language. For a description of Full Esterel, please see the Chapter 1 and 2, and appendix B of *Compiling Esterel* (Potop-Butucaru et al. 2007).

Section 2.2 gives background on language semantics and calculi. It is meant as a refresher to chapters I.1-5, and I.8-I.9 of *Semantics Engineering with PLT Redex* (Felleisen et al. 2009) and sections 1 and 4 of *The Revised Report on the Syntactic Theories of Sequential Control and State* (Felleisen and Hieb 1992).

Section 2.3 gives a semantics for circuits. It is meant as a refresher for *Analysis of Cyclic Combinational Circuits* (Malik 1994), *Constructive Analysis of Cycle Circuits* (Shiple et al. 1996), and borrows heavily from chapters 10.1 and 10.3 of *The Constructive Semantics of Pure Esterel (Draft Version 3)* (Berry 2002). The description of circuits here also relies on the theorems of *Constructive Boolean Circuits and the Exactness of Timed Ternary Simulation* (Mendler et al. 2012), although that work is in no way required to understand this dissertation.

2.1. Esterel

This section is meant as a refresher on Kernel Esterel and its informal semantics. Specifically it is describes Kernel Esterel as given in section 2 of *Compiling Esterel* (Potop-Butucaru et al. 2007), and section 2 of *The Constructive Semantics of Pure Esterel (Draft Version 3)* (Berry 2002). The informal semantics I describe is roughly given in terms of the descriptions in section 3 and 4 of *Compiling Esterel* (Potop-Butucaru et al. 2007) and sections 4.2, 5, and 7 of *The Constructive Semantics of Pure Esterel (Draft Version 3)* (Berry 2002).

Kernel Esterel is a small subset of Esterel in which, for the most part, Full Esterel is trivially macro expressible. It's grammar is:

$$
\begin{aligned}
\mathbf{p}, \mathbf{q} ::= {} & \text{nothing} \mid (\text{exit } \mathbf{n}) \mid (\text{emit } \mathbf{S}) \mid \text{pause} \\
& \mid (\text{signal } \mathbf{S}\ \mathbf{p}) \mid (\text{seq } \mathbf{p}\ \mathbf{q}) \mid (\text{if } \mathbf{S}\ \mathbf{p}\ \mathbf{q}) \mid (\text{par } \mathbf{p}\ \mathbf{q}) \\
& \mid (\text{loop } \mathbf{p}) \mid (\text{suspend } \mathbf{p}\ \mathbf{S}) \mid (\text{trap } \mathbf{p}) \\
& \mid (\text{shared } \mathbf{s} := \mathbf{e}\ \mathbf{p}) \mid (\mathrel{+}= \mathbf{s}\ \mathbf{e}) \mid (\text{var } \mathbf{x} := \mathbf{e}\ \mathbf{p}) \mid (:= \mathbf{x}\ \mathbf{e}) \mid (\text{if!0 } \mathbf{x}\ \mathbf{p}\ \mathbf{q}) \\
S \in {} & \text{signal variables} \qquad x \in \text{sequential variables} \\
s \in {} & \text{shared variables} \qquad e \in \text{host expressions}
\end{aligned}
$$

The Kernel I am using is adapted from section 2.2 of *Compiling Esterel* (Potop-Butucaru et al. 2007). The translation from Full Esterel to this Kernel is given in appendix B of that book.

2.1.1. Pure Esterel

The first three lines of the grammar give the subset of Esterel called Pure Esterel. Pure Esterel defines a "core" to the language which we can use to introduce and examine the important concepts in the language. Pure here refers to just the fragment of Esterel which contains only Esterel terms, without reference to a host language, not to a fragment of Esterel without state.

Instants. Esterel divides computation into instants. Each instant begins in response to some external stimuli, and each instant is atomic with respect to the outside world: its inputs may not change, nor may its outputs or internal state be observed until the instant is completed.

In addition code within each instant can be thought of as running in zero time. That is to say: to maintain deterministic concurrency Esterel does not allow for the program to observe the order in which expressions are run. Without such a total ordering being visible, there isn't really an internal sense of "time" to an instant in Esterel.

The lack of an internal sense of time combined with the fact that the program doesn't run at all outside of instants means that the full execution of an instant is the only notion time in Esterel. Each instant represents one tick forward on a global, discrete clock.[1] In fact, this external-only view time is what gives "instants" their name. We think of every computation in Esterel taking zero time, and so the entire instant completes in zero time.

Signals. Signals, declared with (signal **S p**), give local broadcast communication channels which carry one bit of information: if the signal is present or absent. A signal may only have one value in a single instant.

The conditional form (if **S p q**) conditions on if a signal is present or absent, running **p** or **q** respectively.

The form (emit **S**) sets a signal to present. There is no way to explicitly set a single absent. This asymmetry ties into Esterel's deterministic concurrency and the fact that signals can only obtain one value in an instant. A signal is present if and only if it has is emitted in the current instant. A signal is absent if and only if it is not emitted and *cannot* be emitted in the current instant. The exact meaning of cannot is discussed in section 2.1.3.

Composition. Esterel terms can either be composed concurrently—(par **p q**)—or sequentially—(seq **p q**). seq behaves, more or less, like the sequential composition from familiar languages. par is akin to a fork/join construct: it determines the state of both of its branches and only finishes execution if they both have.

Pausing. The pause form tells the program to stop the instant at that point, and resume from that point in the next instant. If both branches of a par pause, the next instant resumes at both, concurrently. Another way to see this is that pause is the only expression in Esterel which takes time, and it always takes exactly one unit of time.

Non-local control flow. There are two forms of non-local control in Esterel. The first is a form of named, upward jumps, in the form of (trap **p**) and (exit **n**). The (exit **n**) jumps to the **n**th outermost trap (counting from zero). This form cooperates with (par **p q**) such that if both branchs of the par exit, the outer most trap is jumped to.

[1] It should be noted that each instant may not match up to physical time. The outside environment can impose arbitrary and variable delays between instants as each instant is only run on the outside world's request.

For example (par (exit 0) (exit 3)) will jump to the 4th outer most trap. If one branch exits, and the other either pauses or completes, the whole par exits, preempting the non-exiting branch once it has paused. For example both (par (seq (emit **S**) pause) (exit 3)) and (par (seq (emit **S**) (exit 0)) (exit 3)) emit the signal **S** and jump to the fourth outermost trap. I refer to traps as "named" upward jumps because the numbers in exit are really just de Brujin indices for names that appear in Full Esterel. This representation is more convenient to work with in a semantics.

The other kind of non-local control is (suspend **p** **S**). In the first instant a suspend is reached, the suspend behaves like **p**. However in all future instants where the instant would resume in **p**, it only resumes when **S** is absent. If **S** is present, then the whole form pauses, and continues in the next instant (following the same rules).

Loops. The final construct in Pure Esterel is (loop **p**). It continually repeats **p** in an infinite loop. However, because signals can only take on one value per instant in Esterel, any loop which both begins and ends in a single instant will loop forever, causing the instant to never terminate. Therefore Esterel requires that all loops either pause or exit each instant. This ensures that each instant, in fact, terminates.

Schizophrenia. Reincarnated and Schizophrenic variables are a problem related to improper handling of variables and loops in Esterel semantics, particularly when compiling to circuits. Chapter 12 of *The Constructive Semantics of Pure Esterel (Draft Version 3)* (Berry 2002) goes into this in detail. This section gives only a cursory overview.

Reincarnation occurs when loop which contains the declaration of a variable or signal completes execution and restarts in the same instant. This results in two instances of the same variable, and the variable is said to be reincarnated.

Schizophrenia occurs when a reincarnated variable takes on a different value during the two instances of the loop body. This can seem at odds with the statement that "A signal may only have one value in a single instant", and is why these two instances must be though of as separate variables to have a correct semantics. For instance, circuit compilers must duplicate part of all of a loop body to ensure that there are, in fact, two distinct variables.

2.1.2. The host language and Esterel

The last line of the grammar at the start of section 2.1 (shared through if!0) extends Pure Esterel with forms which can track values, in addition to Boolean signals.

However Esterel does not have any notion of value: instead it borrows the outside world's notion of value. That is, Esterel is meant to be embedded in another programming language. This host language controls when instants in Esterel run, and communicates with Esterel using Esterel's signals and is own values. In turn, values in Esterel are computed using the host language's expressions, which may refer to variables bound by Esterel. Values can be stored in either host language or shared variables.

Host Language Variables Host language variables are like traditional programming variables. They are declared and initialized by the (var **x** := **e p**) form, and written to with the (:= **x e**) form. Kernel Esterel also includes another conditional form (if!0 **x p q**), which conditions on the host languages notion of truth—which for purposes of this model will be akin to C's Booleans, with 0 begin false. To maintain deterministic concurrency these variables must be used in a way where concurrent updates are not observable: for example by never using them in multiple branches of a par. How exactly (and if) this is guaranteed depends on the Esterel implementation.

Shared Variables Shared variables give concurrent access to state that may be shared between branches of a par, such that Esterel guarantees deterministic concurrency. Shared variables are declared with (shared **s** := **e p**) and mutated with (+= **s e**). To ensure determinism shared variables have two restrictions. The first is that they must be paired with an associative, commutative combination function which will be used to combine multiple writes to the variable in a given instant, to ensure the order of writes is not visible. The second is that a host language expression referring to a shared variable cannot be evaluated unless no further writes to that shared variable can occur in a given instant. This ensures that only one value for that variable is observed by the program in a given instant. Tracking if a shared variable cannot be written anymore in the current instant uses the same mechanism as determining absence for a signal.[2]

[2]In fact, in Full Esterel shared variables and signals are combined into a single concept: the value carrying signal, which pairs the absence/presence of a signal with a value that is computed the same way as with a shared variable. In Kernel Esterel value carrying signals are represented as a signal paired with a shared variable.

For simplicity's sake, the Constructive Calculus assumes a host language which always uses the combination function +, and that the host language only contains numerical literals, and the operations + and -.

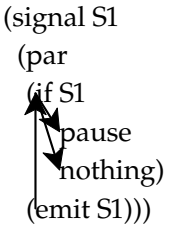
2.1.3. Constructive programs

What is the mechanism used to determine if a signal can be set to absent? Specifically, what kind of reasoning can we perform when showing that a signal cannot be emitted? To show that a signal is emitted or that it cannot be emitted we build a chain of cause and effect which either shows that program Must reach an emit (setting the signal to present) or shows the program Cannot reach an emit (setting the signal to absent).

For a first example, consider the program:

```
(signal S1
 (par
  (if S1
      pause
      nothing)
  (emit S1)))
```

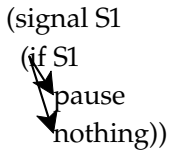
In this example, we can say that the signal S1 is emitted, because we can establish the following chain of cause and effect:



We can read this graph as "emitting the signal S1 on the last line might cause the conditional to take its true branch." We get this interpretation by starting at the entry points of the causality graph and walking forward. In this case the only entry point to the graph is the (emit S1). It points to the conditional that branches on S1, which we can interpret as "this emit running or not running can cause this conditional to take its left or right branch". Because the (emit S1) is at the entry point to the graph we can also conclude that the emit must run. Therefore it must cause the conditional

to take its then branch. This reasoning, where we say "the (emit S1) must happen, therefor the conditional must take this branch" can also be interpreted as "the emit must run before the if", because we are saying that it must cause the present take some action. This prescribing of order to an instant, which is supposed to be timeless, may seem odd. This is because the order we get out of a causality graph is a partial order: there isn't really an internal sense of total time, but rather there are just several possible chains of cause and effect that lead us to a single result. Anything that does not violate the partial ordering goes: including true parallelism.[3]

Now consider this adjustment to the prior program, and its causality graph:



When we walk this causality graph from its entry points as before, we immediately run into the conditional without hitting any (emit S1)s. In addition if we keep walking forward, neither branch can emit S1 either. From this we can safely conclude that nothing can cause S1 to be emitted, therefore it cannot be emitted, therefore it is absent.

Notice that the asymmetry in the syntax—that we have a form for setting a signal to present but not for setting to a signal absent—leads to an asymmetry in our reasoning. To reason about the presence of a signal we consider the causality graph up to the conditional and what it must do. To reason about absence of a signal we look at the entire graph and reason about what it cannot do. However the reasoning about what it cannot do is, itself, restricted by causality. Consider the program in figure 1, which is the same as the previous one but with the pause replaced by an emit. To make the graph easier to read it has been pulled out into a separate image. The darker edges[4] describe the parts of the causality graph that come from the control of the program. The lighter edges[5] come from the data flow. Conditions are represented as nodes labeled `(? S)`, and their branches are labeled `T` and `F`, for the then and else branches respectively. Other control flow edges are labeled with `n` if they pass control on in this instant, `p` if they pass control on in the next instant (e.g. a pause), or a number if they exit with that code.

[3]To quote Gérard Berry: "Everything happens at the same time, just in the right order."
[4]Blue if printed in color.
[5]Pink if printed in color.

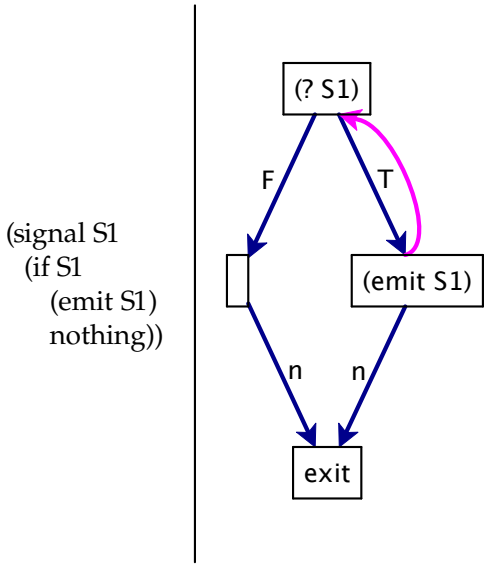


Figure 1: A program with a separate causality graph

As before, we cannot set S1 to present, as there is no emit that must be reached before the conditional. However we cannot set S1 to absent either, as the emit in the then branch might still be reached! One might assume that we could analyze the conditional as if S1 were absent looking at only the else branch, as we know it cannot be present. However this would amount justifying S1 being absent based on the assumption that it was absent. Such self justification doesn't leave a clear chain of cause and effect which result in showing the signal is absent: one of the reasoning steps is just a guess. Esterel considers programs like this one, where some signals cannot be set to either absent or present, to be illegal. Such programs are either rejected statically or raise a runtime error, depending on the Esterel implementation. Programs like this are called non-constructive.[6]

Another way of seeing this is observing that causality graph for that program has a cycle in it: in a timeless world the emit in the then branch could cause the conditional to make a particular choice, which could cause the emit to be reached. Such a cycle in causality does not make sense (and does not give us even a partial ordering on events).

However causality cycles are not always nonsense. In some cases a cycle does not result in a non-constructive program because prior steps in our reasoning may allow the cycle to be broken. Consider the program in figure 2.

[6]This comes from the analogous lack of self-justified reasoning in a constructive logic—that is we may not use the law of the excluded middle to reason about signals.

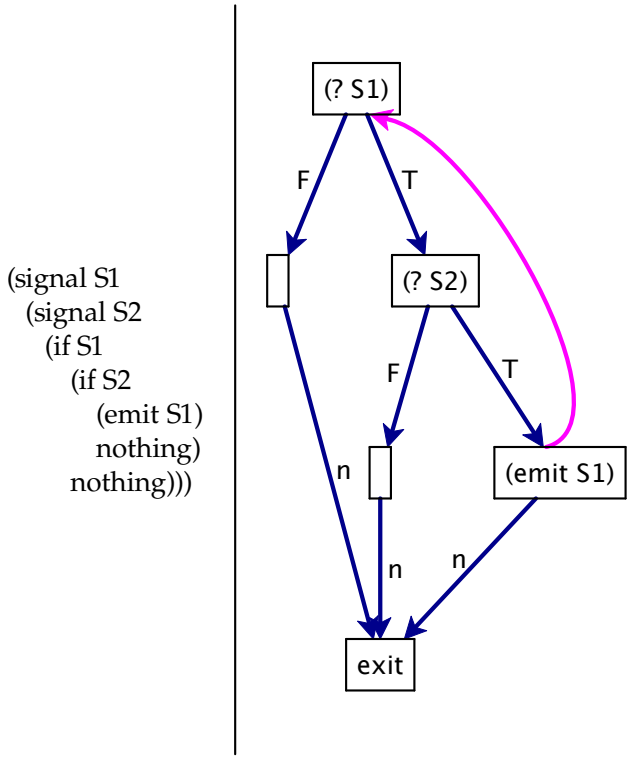


Figure 2: A constructive program with a causality cycle

This program has a causality cycle, because the condition S1 might cause S1 to be emitted. However, we can also see that no emit for S2 is reachable in the causality graph, which means we can set it to absent. But now that we have justified setting S2 to absent, we can justify ignoring any code in that conditionals then branch. This causes the (emit S1) is unreachable, so we can cut any edges in the causality graph leading to or from it. Now we have a causality graph with no cycles that never reaches an (emit S1), so we can set S1 to absent.

Causality graphs also interact with pause. The pause ends an instant (and causes the next instant to pick up from the pause), and the single value restriction for signals only pertains to a single instant. Therefore, pauses essentially cut the causality graph, splitting it in two: one for the instants before the pause is reached, and one for the ones after. For example consider the program and graph in figure 3. This program will emit S1, then pause. In the next instant it will emit S2. This is represented in the graph by introducing a new node start, which has a choice: if it is starting this

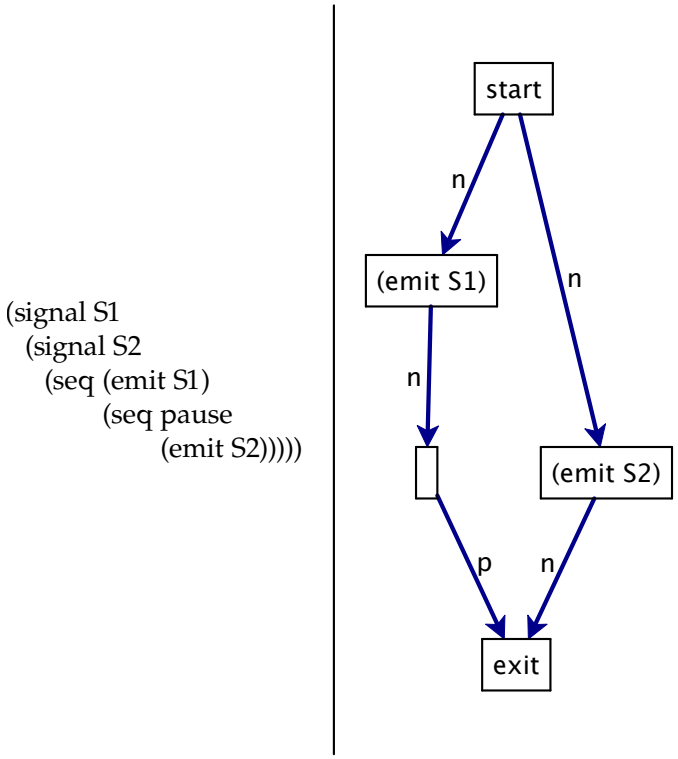


Figure 3: A simple graph split by a pause.

program fresh it will go down the path which emits S1, and pauses, terminating that instant. If it is an instant where it's resuming for the pause, it will take the right hand branch and emit S2.

We can use this to see how pause can break what might otherwise be causality cycles. Look at the differences in the programs and graphs in figure 4 and figure 5. In the first example when we walk the graph from start to finish we find that we need the value for S1 first, but after that condition we might emit S1 so we cannot set it to absent. This cycle renders the program non-constructive. But in the second example, where the last nothing is replaced by a pause we have a different graph with no cycles! Because the emit cannot happen in the same instant as the condition (represented by the choice of where start goes) this program is constructive, and its graph is acyclic.

Must/Cannot and Present/Absent. The description of constructivity in terms of program graphs and what Must and Cannot happen gives a complete semantics for Pure Esterel. This is described by the Constructive Behavioral Semantics (Berry 2002), which defines Esterel in terms of two functions: Can and Must. The first function, Can,

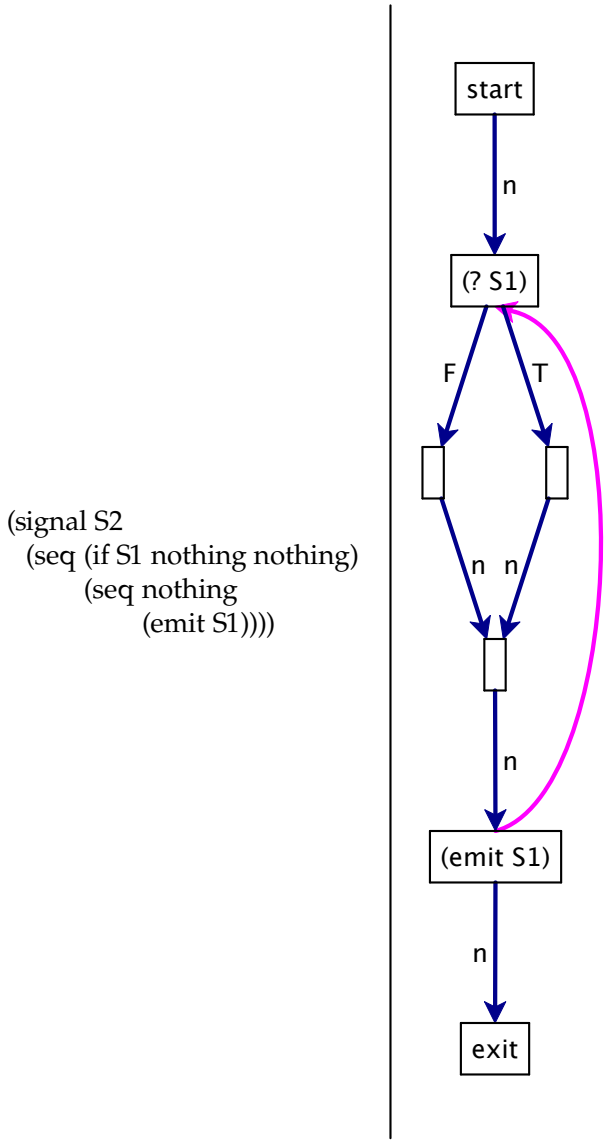


Figure 4: A cycle

returns the set of signals which a program might emit. It's complement, Cannot, gives us the set of signal which it is impossible for a program to emit. The second function, Must, gives us the set of signals which the program will definitely emit. It's complement, Might Not, is the set of signals which the program could potentially not emit.

This divides up program behavior into the chart in figure 6. The upper left corner of this diagram, labeled 1, corresponds to a portion of the program being executed, and to a signal being present. The bottom right, labeled 0,

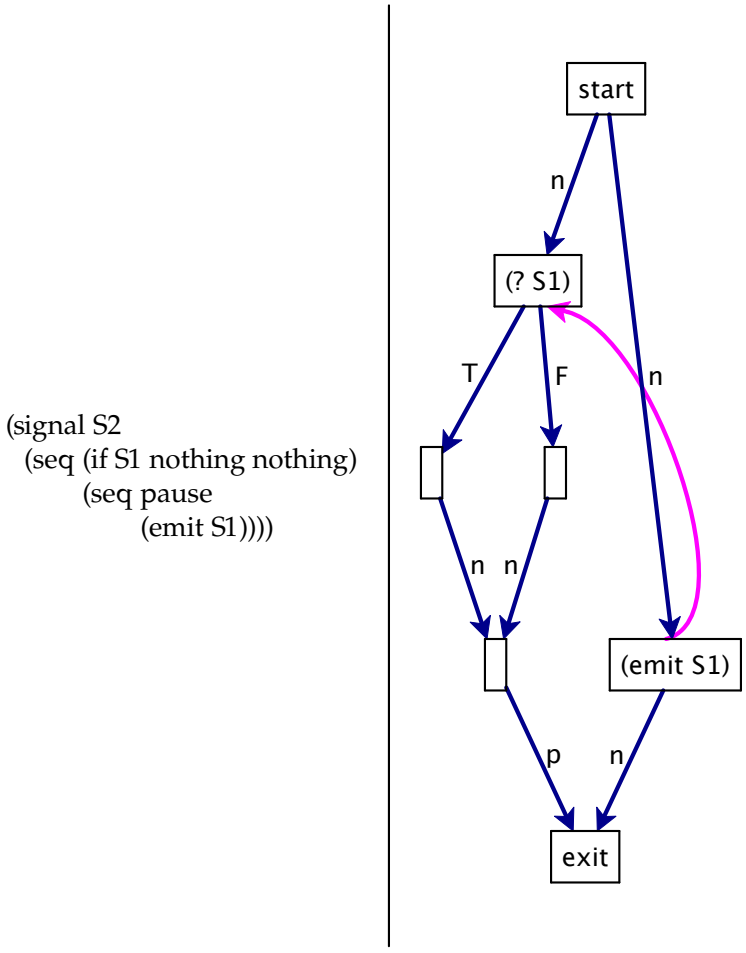


Figure 5: A cycle cut by a pause.

corresponds to a portion of the program not being executed and to a signal being absent. The upper right corner, labeled $\perp$ corresponds to a portion of the program being in an unknown state. The lower left is never realized.

Every edge in the program graph begins in the corner of the chart labeled $\perp$, representing that it could be executed, but might not be. As we execute the program, any control edge can be moved to the 1 corner (it both Must and Can happen), if *all* of the incoming edges to the node it leads from are 1. Conversly, a control edge can be set to 0 if *any* of it's incoming edges are 0. Note that this means that the edges leading from start automatically have 1's as they have no incoming edges. Conversely a Data edge can be set to 1 if *any* of it's incoming edges are 1, but can only be set to 0 if *all* of it's incoming edges are 0. This corresponds to control edges being linked by an extension of $\wedge$ with

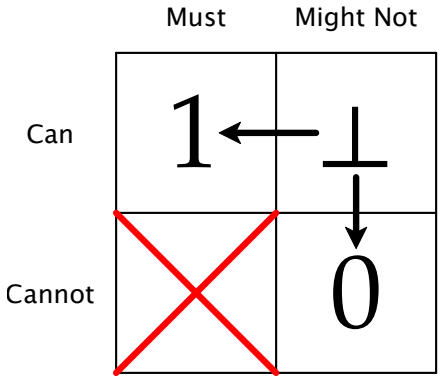


Figure 6: Must/Can Lattice

$\perp$, and data edges being linked to by a similar extension of $\vee$. This is discussed in more detail in section 2.3, and section 4.1.1, and this leads to the circuit semantics of Esterel.

The Must and Cannot corner in the diagram is not reachable, as it is a logical contradiction. No consistent and sound semantics should be able to put programs in this corner. This will motivate some of the design decisions of the Calculus.

It should also be noted that this description leads to a kind of asymmetry between the Must and Cannot corners. The Must corner can be reached only if there is a chain of 1's leading form the top of the program, as the only control node that has zero incoming edges is start. The Cannot corner however can be reached without a connection to the top of the program, as any signal with no emit can automatically get a 0. This leads to an odd fact: Must is less compositional than Cannot. Any program can be put into a context where is wont be executed, therefore Must requires knowledge about where the top of the program is. However if something Cannot happen, then no context can make it happen. This means that given no information about its context, Cannot still gives may give us useful information, whereas Must cannot. This asymmetry will show up several times in the design of the Calculus.

2.1.4. Summary of Notation

The syntax for Kernel Esterel I am using is:

$$
\begin{array}{l}
\mathbf{p}, \mathbf{q} ::= \text{nothing} \mid (\text{exit } \mathbf{n}) \mid (\text{emit } \mathbf{S}) \mid \text{pause} \\
\qquad \mid (\text{signal } \mathbf{S}\ \mathbf{p}) \mid (\text{seq } \mathbf{p}\ \mathbf{q}) \mid (\text{if } \mathbf{S}\ \mathbf{p}\ \mathbf{q}) \mid (\text{par } \mathbf{p}\ \mathbf{q}) \\
\qquad \mid (\text{loop } \mathbf{p}) \mid (\text{suspend } \mathbf{p}\ \mathbf{S}) \mid (\text{trap } \mathbf{p}) \\
\qquad \mid (\text{shared } \mathbf{s} := \mathbf{e}\ \mathbf{p}) \mid (\mathrel{+}= \mathbf{s}\ \mathbf{e}) \mid (\text{var } \mathbf{x} := \mathbf{e}\ \mathbf{p}) \mid (:= \mathbf{x}\ \mathbf{e}) \mid (\text{if!0 } \mathbf{x}\ \mathbf{p}\ \mathbf{q}) \\
\mathrm{S} \in \text{signal variables} \qquad\qquad \mathrm{x} \in \text{sequential variables} \\
\mathrm{s} \in \text{shared variables} \qquad\qquad \mathrm{e} \in \text{host expressions}
\end{array}
$$

I will often refer to the presence of a signal as 1, and the absence of a signal as 0.

2.2. Language Calculi

This section will give the background about language calculi. It covers the call-by-value λ-calculus (Plotkin 1975), small step operational semantics, evaluation contexts (Felleisen and Friedman 1986) and the state calculus (Felleisen and Hieb 1992). The material summarized here can be read about in depth in Chapters I.1-5, and I.8-I.9 of *Semantics Engineering with PLT Redex* (Felleisen et al. 2009)—excluding the subsections that deal only with proofs—and sections 1 and 4 of *The Revised Report on the Syntactic Theories of Sequential Control and State* (Felleisen and Hieb 1992).

A semantics is some mapping from (possibly partial) programs to their meaning. For example: evaluators are functions mapping programs to the result of running them; and denotational semantics give meaning by mapping programs to elements of some external domain, like the circuit semantics for Esterel. The semantics I plan to build will give meaning to terms by mapping them to a set of terms to which they are equivalent—an equivalence class. Specifically I will do this by giving a set of axioms that define an equivalence relation, which will implicitly define this mapping from terms to sets of terms. This set of terms is defined as the set of terms which the given term can be simplified to by the axioms of the calculus.

This equivalence relation will let us reason about programs like we reasoned about arithmetic in grade school: if we can show two terms are equal, then we can safely replace of those terms for another in some larger program without changing its meaning. I refer to kind of equality relation as a calculus (taking the name from Church's λ-calculus).

This section walks through an example of a calculus—the call-by-value λ-calculus (Plotkin 1975), also called λ_v—and the key definitions needed for a calculus. The grammar of this the example language is:

$$\begin{array}{l}\mathbf{e} ::= \mathbf{x} \\ \quad | \, (\lambda \, \mathbf{x} \, \mathbf{e}) \\ \quad | \, (\mathbf{e} \, \mathbf{e} \, ...) \\ \quad | \, \mathbf{c} \\ \mathrm{x} \in \text{Variable Names}\end{array}$$

That is, expressions consist of variables, anonymous functions of one argument, function applications, and built in constants (which may contain primitive functions). Another useful grammatical definition is that of a value:

$$\mathbf{v} ::= (\lambda \, \mathbf{x} \, \mathbf{e}) \mid \mathbf{c}$$

which is to say, just functions and constants. They represent fully evaluated terms.

2.2.1. Notions of Reduction

To build a calculus we first build a small[7] relation called the notions of reduction. This represents the core notions of computation in this language. I will write this relation as $\rightharpoonup$. In general I will add a superscript relations to show which language they refer to. For example the notions of reduction for λ_v will be written as $\rightharpoonup^{\lambda}$ (the superscript drops the $\mathbf{v}$ to avoid a subscript in a superscript). The notions of reduction for λ_v are:

$$\begin{array}{ll}[\beta_{\mathbf{v}}] & ((\lambda \, \mathbf{x} \, \mathbf{e}) \, \mathbf{v}) \rightharpoonup^{\lambda} \mathit{subst}(\mathbf{e}, \mathbf{x}, \mathbf{v}) \\ [\delta] & (\mathbf{c} \, \mathbf{v} \, ...) \rightharpoonup^{\lambda} \delta(\mathbf{c}, \mathbf{v}, ...)\end{array}$$

The left most part of each line is the rule name. Then comes a pattern which describes what goes on the left of the relation, what we might think of as its "input": functions applied to a value for $[\beta_{\mathbf{v}}]$, and primitive constants applied to many values for $[\delta]$. On the right is a pattern which describes what is on the right of the relation, what we might think of at its "output". In this case both right hand sides consist of Metafunctions, that is functions in our metalanguage, rather than functions in λ_v. Metafunction application is written $\mathit{name\text{-}of\text{-}function}(\mathrm{args}, ...)$. The $[\beta_{\mathbf{v}}]$ rule, which describes anonymous function application, relates the application of a function to that functions body,

[7] Well, okay, *technically* the relation is infinite in size, but it has a small number of rules.

but with every occurrence of the variable substituted with the argument. The rule $[\delta]$ handles primitive function application, by calling out to the metafunction δ, which the calculus is parameterized over. In a sense this function represents the "runtime" of the λ_V. So, for example, if $\mathbf{c}$ includes + and numbers, then δ would include a specification of addition.

The relation $\longrightarrow^{\lambda}$ can be called the notions of reduction because, at least so far, each clause of $\longrightarrow^{\lambda}$ is some atomic step in evaluating the program. Since λ_V only contains functions, the only rules in $\longrightarrow^{\lambda}$ handle function application.

2.2.2. Alpha Equivalence

Not all rules may be computationally relevant, but may instead simply describe equivalences we wish to hold. For example, a common rule in λ-calculi is $[\alpha]$:

$$[\alpha]\ (\lambda\ \mathbf{x}\ \mathbf{e}) \longrightarrow^{\lambda} (\lambda\ \mathbf{x}_2\ \mathit{subst}(\mathbf{e}, \mathbf{x}, \mathbf{x}_2))$$
$$\text{if } \mathbf{x}_2 \notin \mathit{FV}(\mathbf{e})$$

This rule describes consistent renaming of terms: one term may step to another if we replace a bound variable with a new name that is not free in the term. We can think of this rule as describing the renaming refactoring found in IDEs. Two terms which can step to each other via only the $[\alpha]$ rule are said to be alpha equivalent.

2.2.3. Equality relation

Using the notions of reduction, a calculus is built as an equality relation which says, essentially, if some part of two programs could be run forwards or backwards to new terms, such that the two programs are become textually equal, then they two programs must be equivalent.

To start with, we must describe what "some part of the programs" means. To do this we use the notion of a context, which lets us split programs into an inner and outer piece. For a calculus we use program contexts, $\mathbf{C}$. In this case of λ_V, these contexts are:

$$\begin{aligned}\mathbf{C} ::= &\bigcirc \\ &\mid (\mathbf{e} \ldots \mathbf{C}\ \mathbf{e} \ldots) \\ &\mid (\lambda\ \mathbf{x}\ \mathbf{C})\end{aligned}$$

This states that we can break a program $\mathbf{e}_1$ into two parts, $\mathbf{C}$ and $\mathbf{e}_2$, written $\mathbf{e}_1 = \mathbf{C}[\mathbf{e}_2]$ by tracing down the top of $\mathbf{e}_1$ following the path laid out by the grammar of $\mathbf{C}$. For instance, any program can be broken into the empty context $\bigcirc$ and itself: $\mathbf{e}_1 = \bigcirc[\mathbf{e}_1]$. The program (+ 3 (+ 1 2)) could be broken into (+ $\bigcirc$ (+ 1 2))[3], (+ 3 $\bigcirc$)[(+ 1 2)], and (+ 3 (+ 1 $\bigcirc$))[2], among others.

Note that the program contexts $\mathbf{C}$ can be generated algorithmically for some non-terminal: they simply go under every single recursive part of that non-terminal. Therefore from here on out I will not write out the definitions of $\mathbf{C}$.

With this in hand we can describe the two axioms of the equality relation which describe evaluating anywhere in the program:

$$\frac{\mathbf{e}_i \rightharpoonup^{\lambda} \mathbf{e}_o}{\mathbf{e}_i \equiv^{\lambda} \mathbf{e}_o}\,[\text{step}] \quad \frac{\mathbf{e}_1 \equiv^{\lambda} \mathbf{e}_2}{\mathbf{C}[\mathbf{e}_1] \equiv^{\lambda} \mathbf{C}[\mathbf{e}_2]}\,[\text{ctx}]$$

The $[\mathbf{step}]$ rule says that two terms are equal if they are related by the notions of reduction. The $[\mathbf{ctx}]$ rule says that our reasoning applies in any program context. From here we turn this into an equality relation: that is we make it transitive, reflexive, and symmetric:

$$\frac{}{\mathbf{e} \equiv^{\lambda} \mathbf{e}}\,[\text{refl}] \quad \frac{\mathbf{e}_1 \equiv^{\lambda} \mathbf{e} \qquad \mathbf{e} \equiv^{\lambda} \mathbf{e}_2}{\mathbf{e}_1 \equiv^{\lambda} \mathbf{e}_2}\,[\text{trans}] \quad \frac{\mathbf{e}_2 \equiv^{\lambda} \mathbf{e}_1}{\mathbf{e}_1 \equiv^{\lambda} \mathbf{e}_2}\,[\text{sym}]$$

The $[\mathbf{refl}]$ rule says that all terms are equal to themselves. The $[\mathbf{trans}]$ rule says that we can chain reasoning steps together, if $\mathbf{A}$ is equal to $\mathbf{B}$, and $\mathbf{B}$ is equal to C, then $\mathbf{A}$ must therefore be equal to C. The final rule, $[\mathbf{sym}]$ says that if $\mathbf{A}$ is equal to $\mathbf{B}$ then $\mathbf{B}$ is equal to $\mathbf{A}$. This rule is the one that allows us to "run" programs backwards.

Sometimes it is valuable to be able to describe stepping forward. This relation is given as closure of the notion of reduction under program contexts—the same as just the $[\mathbf{step}]$ and $[\mathbf{ctx}]$ rules of the equivalence relation. This is noted with $\longrightarrow$, and will be superscripted to show with language it comes from. This closure under program contexts is also called the compatible closure.

2.2.4. Reasoning with a calculus

Now that we have a calculus, what can we do with it? The core idea of how to reason with a calculus is the same as how we reason in our algebra classes from grade school: we "run" our core equalities backwards and forwards until we massage the term into the form we want.

For example, let us say we want to perform something like common subexpression elimination, and prove that:

$$(+\ (+\ 1\ 1)\ (+\ 1\ 1)) \equiv^{\lambda} ((\lambda\ \mathbf{x}\ (+\ \mathbf{x}\ \mathbf{x}))\ (+\ 1\ 1))$$

The reasoning process might go something like this:

(1) By **[step]** and $[\delta]$, $(+\ 1\ 1) \equiv^{\lambda} 2$. That is we can run parts of the program forward one step.

(2) By **[cxt]** and (1), (twice, by **[trans]**) $(+\ (+\ 1\ 1)\ (+\ 1\ 1)) \equiv^{\lambda} (+\ 2\ 2)$. That is we can use **[ctx]** to substitute $\equiv^{\lambda}$ terms in a larger term. Now we are working with $(+\ 2\ 2)$.

(3) By **[step]** and $[\beta_{\mathbf{v}}]$, $((\lambda\ \mathbf{x}\ (+\ \mathbf{x}\ \mathbf{x}))\ 2) \equiv^{\lambda} (+\ 2\ 2)$. We are just running the program forward again.

(4) By **[sym]** and (3), $(+\ 2\ 2) \equiv^{\lambda} ((\lambda\ \mathbf{x}\ (+\ \mathbf{x}\ \mathbf{x}))\ 2)$. **[sym]** lets us take our previous "run forwards" example and use it to actually run backwards. Now we are working with $((\lambda\ \mathbf{x}\ (+\ \mathbf{x}\ \mathbf{x}))\ 2)$.

(5) By **[sym]**, **[ctx]**, (1), and **[trans]**, $((\lambda\ \mathbf{x}\ (+\ \mathbf{x}\ \mathbf{x}))\ 2) \equiv^{\lambda} ((\lambda\ \mathbf{x}\ (+\ \mathbf{x}\ \mathbf{x}))\ (+\ 1\ 1))$. We're running backwards again, and substituting into a larger context. Now we have our final term $((\lambda\ \mathbf{x}\ (+\ \mathbf{x}\ \mathbf{x}))\ (+\ 1\ 1))$.

2.2.5. Evaluators

For a calculus to be adequate, it must be able to define an evaluator for its language. I don't, by this, mean it gives an effective means to compute a program, but rather that it gives a mathematical definition of what the results of such a function should be. For example, the λ_V evaluator might be:

Definition: $eval^{\lambda}(\mathbf{e})$

$$
\begin{aligned}
&\mathit{eval}^{\lambda} : \mathbf{e} \rightarrow \mathbf{c} \text{ or procedure} \\
&\mathit{eval}^{\lambda}(\mathbf{e}) = \mathbf{c} \quad \text{if } \mathbf{e} \equiv^{\lambda} \mathbf{c} \\
&\mathit{eval}^{\lambda}(\mathbf{e}) = \text{procedure} \quad \text{if } \mathbf{e} \equiv^{\lambda} (\lambda\ \mathbf{x}\ \mathbf{e}_1)
\end{aligned}
$$

Which says that if a program is equivalent to a constant, then that program must evaluate to that constant. If the program is equivalent to an anonymous function, then the result is the special symbol procedure. Note that it is not a given that eval^{λ} is a function: it is entirely possible it could map one expression to two different results. This gives us the definition of consistency for a calculus: the evaluator it defines is a function.

2.2.6. State

One more important piece of background is how one can handle state in calculi. State is tricky because it is inherently non-local. The two key pieces for handling state are evaluation contexts (Felleisen and Friedman 1986) and local environments (Felleisen and Hieb 1992). The description I give here is adapted from the state calculus in Felleisen and Hieb (1992). In this section we extend λ_V to support state, and call the extension λ_σ. To start with, we must be able to control the order of evaluation of terms, as state is order sensitive. To do this we need a new kind of context, which only allows holes in specific places depending on how far along the program is in its evaluation. For λ_σ they are:

$$
\begin{aligned}
&\mathbf{E} ::= \bigcirc \\
&\quad\ \mid (\mathbf{v} \ldots \mathbf{E}\ \mathbf{e} \ldots)
\end{aligned}
$$

These contexts limit where holes may be place, so that evaluation can only take place at the first non-value term of a function application. From here we add local state to the syntax of the language, represented by the form, $(\varrho\ \boldsymbol{\theta}.\ \mathbf{e})$, and a form to mutate variables, $(\sigma\ \mathbf{x}\ \mathbf{e})$:

$$
\begin{aligned}
&\mathbf{e} ::= .... \\
&\quad\ \mid (\varrho\ \boldsymbol{\theta}.\ \mathbf{e}) \\
&\quad\ \mid (\sigma\ \mathbf{x}\ \mathbf{e}) \\
&\mathbf{E} ::= .... \mid (\sigma\ \mathbf{x}\ \mathbf{E}) \\
&\boldsymbol{\theta} : \mathbf{x} \rightarrow \mathbf{v}
\end{aligned}
$$

The ϱ form adds a map θ to a term. This map associates bound mutable variables with their current values. From here we can define the notions of reduction:

$$
\begin{array}{ll}
[\beta_\sigma] & ((\lambda\ \mathbf{x}\ \mathbf{e})\ \mathbf{v}) \rightharpoonup^{\sigma} (\varrho\ \{\ \mathbf{x} \mapsto \mathbf{v}\ \}.\ \mathbf{e}) \\
[\sigma] & (\varrho\ \theta.\ \mathbf{E}[(\sigma\ \mathbf{x}\ \mathbf{v})]) \rightharpoonup^{\sigma} (\varrho\ \theta \leftarrow \{\ \mathbf{x} \mapsto \mathbf{v}\ \}.\ \mathbf{E}[42]) \\
 & \quad \text{if } \mathbf{x} \in \mathit{dom}(\theta) \\
[\mathbf{D}] & (\varrho\ \theta.\ \mathbf{E}[\mathbf{x}]) \rightharpoonup^{\sigma} (\varrho\ \theta.\ \mathbf{E}[\theta(\mathbf{x})]) \\
 & \quad \text{if } \mathbf{x} \in \mathit{dom}(\theta) \\
[\mathbf{lift}] & (\varrho\ \theta_1.\ \mathbf{E}[(\varrho\ \theta_2.\ \mathbf{e})]) \rightharpoonup^{\sigma} (\varrho\ (\theta_1 \leftarrow \theta_2).\ \mathbf{E}[\mathbf{e}]) \\
[\delta] & (\mathbf{c}\ \mathbf{v}\ ...) \rightharpoonup^{\sigma} \delta(\mathbf{c}, \mathbf{v}, ...)
\end{array}
$$

The first rule is our β rule, which handles function application by allocating a new local environment for that term. The next two rules handle setting and dereferencing variables. If a σ is within an evaluation context of an environment which contains its variable, that means that the σ is the next term to run with respect to that environment. Therefore it can be run, changing the mapping in the environment to the new value. An arbitrary value is left in place of the σ.[8] Dereferencing works in a similar way: if a variable is within an evaluation context of its environment, then dereferencing that variable is the next step than can be taken with respect to that environment. Environments can be shifted around via the $[\mathbf{lift}]$ rule, exposing new redexs. The final rule is the same $[\delta]$ rule as in λ_v.

From here we define the equality relation $\equiv^{\sigma}$ the same way we defined $\equiv^{\lambda}$: by closing the notions of reduction over program contexts, transitivity, reflexively, and symmetry:

$$
\frac{\mathbf{e}_i \rightharpoonup^{\sigma} \mathbf{e}_o}{\mathbf{e}_i \equiv^{\sigma} \mathbf{e}_o}\,[\text{step}] \quad \frac{\mathbf{e}_1 \equiv^{\sigma} \mathbf{e}_2}{\mathbf{C}[\mathbf{e}_1] \equiv^{\sigma} \mathbf{C}[\mathbf{e}_2]}\,[\text{ctx}]
$$

$$
\frac{}{\mathbf{e} \equiv^{\sigma} \mathbf{e}}\,[\text{refl}] \quad \frac{\mathbf{e}_1 \equiv^{\sigma} \mathbf{e} \qquad \mathbf{e} \equiv^{\sigma} \mathbf{e}_2}{\mathbf{e}_1 \equiv^{\sigma} \mathbf{e}_2}\,[\text{trans}] \quad \frac{\mathbf{e}_2 \equiv^{\sigma} \mathbf{e}_1}{\mathbf{e}_1 \equiv^{\sigma} \mathbf{e}_2}\,[\text{sym}]
$$

[8] In the grand tradition of The Hitchhikers Guide to the Galaxy, the best arbitrary value is 42.

2.2.7. Contextual equivalence

The strongest notion of equivalence between programs we can have is called contextual equivalence (Morris 1963), which I will write as $\simeq$. Contextual equivalence is, generally, defined as a relation between programs which cannot be distinguished in any context. For example, for λ_v, we could define contextual equivalence as

Definition: $\mathbf{e}_1 \simeq^{\lambda} \mathbf{e}_2$

$$\mathbf{e}_1 \simeq^{\lambda} \mathbf{e}_2 \textit{ if and only if, for all contexts } \mathbf{C}, \textit{eval}^{\lambda}(\mathbf{C}[\mathbf{e}_1]) = \textit{eval}^{\lambda}(\mathbf{C}[\mathbf{e}_1]).$$

The definition of contextual equivalence depends on the language in question.

In general contextual equivalence is a "stronger" equivalence relation than the relation defined by a calculus; however for a calculus to be sound it must be that $\equiv^{\mathrm{E}}$ is a subrelation of $\simeq$. That is, if $\mathbf{e}_1 \equiv^{\lambda} \mathbf{e}_2$ then it must be that $\mathbf{e}_1 \simeq^{\lambda} \mathbf{e}_2$, but the converse does not need to hold.

2.2.8. Summary of Notation

- *name-of-function*(args, ...): Metafunction application.
- *eval*: The evaluator for a language.
- $\rightharpoonup$: The notions of reduction for a language.
- $\equiv$: The equivalence relation defined by a calculus. Defined as closure of $\rightharpoonup$ under program contexts, symmetry, reflexivity, and transitivity.
- $\longrightarrow$: The equivalence relation defined by a calculus. Defined as closure of $\rightharpoonup$ under program contexts.
- $\simeq$: Contextual Equivalence.

- $\mathbf{C}$: Program Contexts.

I will, in general, use superscripts to distinguish evaluators and relations from different languages.

2.3. Circuits

This section refreshes background needed to understand the potentially cyclic circuits that Esterel compiles to. It summaries *Analysis of Cyclic Combinational Circuits* (Malik 1994) and *Constructive Analysis of Cycle Circuits* (Shiple et al. 1996), and Chapter 10 of *The Constructive Semantics of Pure Esterel (Draft Version 3)* (Berry 2002).

2.3.1. Circuits as Graphs

Circuits can be thought of as graphs, where each edge represents a wire, and each node represents a gate. This also gives the usual pictorial representation of circuits. As an example, the left of figure 7 is a circuit for the XOr of two bits. This is implemented by taking the nand'ing and or'ing both of the two inputs, and and'ing the outputs of those two gates. The right of figure 7 gives an overview of the notation I am using, which is fairly standard. Buffers here do not impose a delay, but rather just specify the direction voltage propagates through the circuit. Registers save values between cycles in the circuit. The are always initialized to 0 in the first cycle, and then output their input value from the previous cycle afterwards. A small circle on the input or output of a gate represents negating that wire.

A side note on diagrams: Sometime circuit diagrams may refer to wires in a sub-circuits that are not present. For example, see figure 8, which shows the compilation rules for two of Esterel's forms. The first circuit has five input wires: E_i, SEL, RES, SUSP, and KILL. It has two named output wires, SEL and E_o, and some number of output wires labeled K0 through Kn. It has a subcircuit, which for now we can think of as being named $[\![\mathbf{p}^{\mathsf{P}}]\!]$. This subcircuit has the same interface as the overall circuit. All input wires are passed to $[\![\mathbf{p}^{\mathsf{P}}]\!]$ unchanged, except for KILL. The KILL for $[\![\mathbf{p}^{\mathsf{P}}]\!]$ is given as the Or of the KILL of the outer circuit, and the K2 output from $[\![\mathbf{p}^{\mathsf{P}}]\!]$. The outputs of the overall circuit shift Kn to Kn-1, except for K1 which is just the K1 of $[\![\mathbf{p}^{\mathsf{P}}]\!]$, and K0 which is the Or of the K0 from $[\![\mathbf{p}^{\mathsf{P}}]\!]$ and the K2 of $[\![\mathbf{p}^{\mathsf{P}}]\!]$.

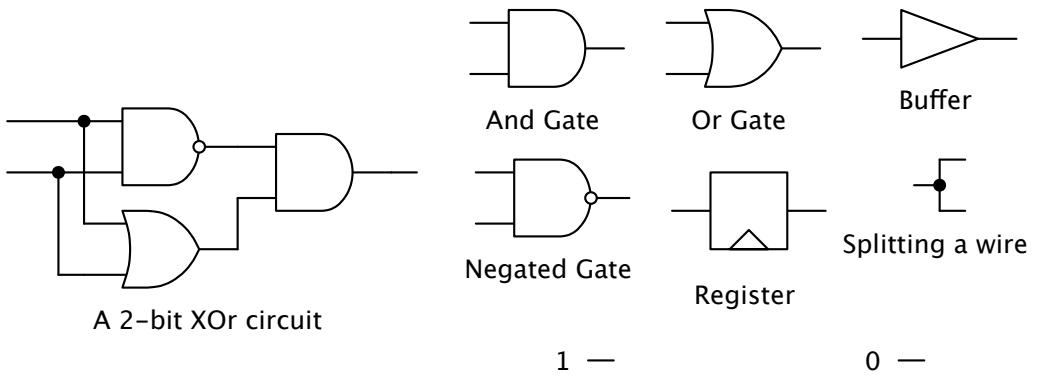


Figure 7: Circuit Diagram Overview

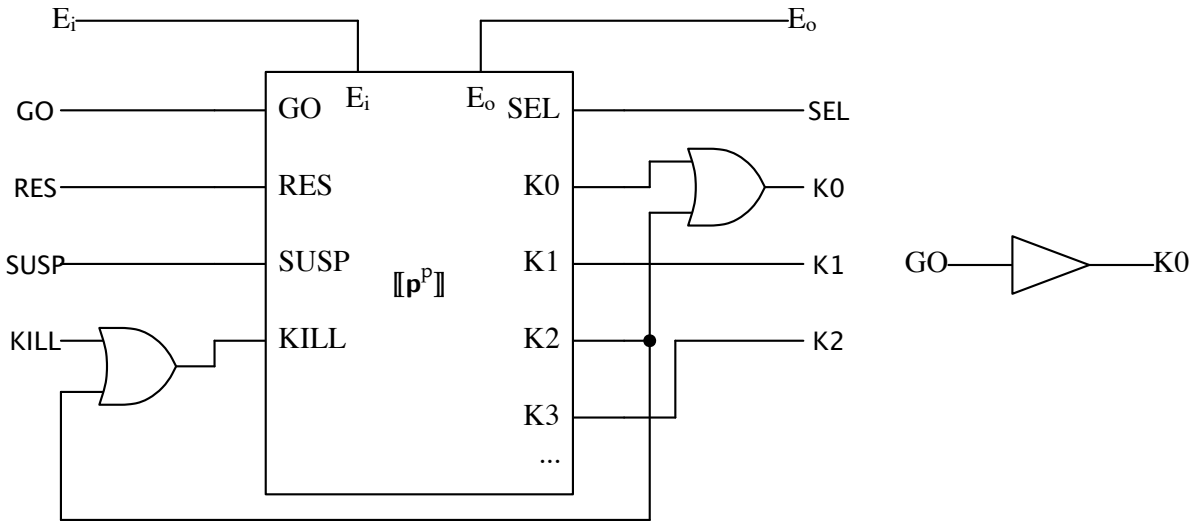


Figure 8: An example in which the sub-circuit may be missing wires

If we take $[\![\mathbf{p}^{\mathrm{P}}]\!]$ to be the second circuit in figure 8, then we have missing wires, as the only input to this circuit is GO, and the only output is K0. In this case we may just ignore the inputs to the inner circuit which are not used. In addition we take the outputs which the outer circuit expects but that the input does not define as begin a constant 0.

Therefore when the second circuit is inserted into the first, we get that every output of the outer circuit except for K1 is a constant 0.[9]

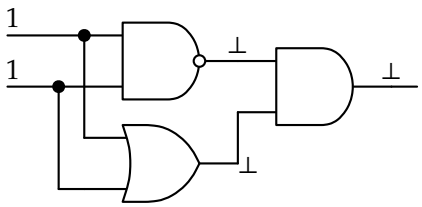
2.3.2. Interpreting a circuit

As an intuition for how to execute a circuit: initialize internal and output wire to the special value $\perp$ and each input wire to the input value given by the environment. This special value $\perp$ represents that we do not yet know the value on that wire. From here iterate through each gate in the circuit, updating the output of each gate if the inputs allow it to change, using the truth tables extended with $\perp$ in figure 9. Continue this until a fixed-point is reached.

The extended truth tables follow the principle that if a value is enough to determine the output of a gate one its own (i.e. would "short circuit" evaluation), then that value controls the output when combined with $\perp$, otherwise the output is $\perp$. This ensures that the output of gates is monotonic: once its output transitions from $\perp$ to 0 or 1 its output will never change.

Once a fixed point has been reached, the input values will have propagated through the entire circuit, with each gate settling on its final value. From here the values of each output wire will we be on those wires.

As and example of this, let us explore the evaluation of the XOr circuit, when run on the values 1 and 1. The initial state looks like:



From here we can either evaluate the Nand or the Or gate. If we evaluate the Nand gate we get:

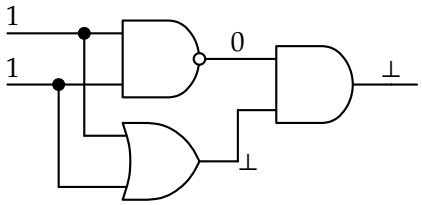
[9]Using this rule for composition we can in fact simplify the combined circuit back to the second circuit.

a ∧ b	= o
1 1	1
1 0	0
1 ⊥	⊥
0 1	0
0 0	0
0 ⊥	0
⊥ 1	⊥
⊥ 0	0
⊥ ⊥	⊥

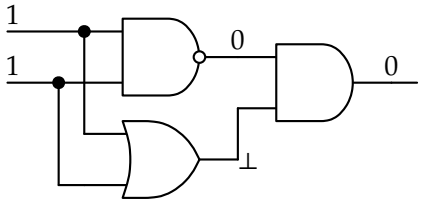
a ∨ b	= o
1 1	1
1 0	1
1 ⊥	1
0 1	1
0 0	0
0 ⊥	⊥
⊥ 1	1
⊥ 0	⊥
⊥ ⊥	⊥

¬a	= o
1	0
0	1
⊥	⊥

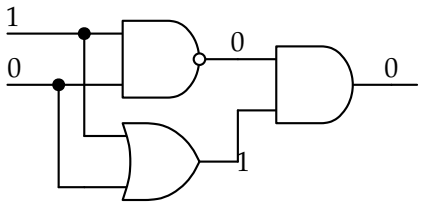
Figure 9: Truth tables for each gate, extended with ⊥



Next we can evaluate either the Or or the And gate, as 0 short circuits And. If we evaluate the And gate we get:



Finally we can evaluate the Or gate:



From here evaluating any gate gives the same result, therefore a fixed-point has been reached and the final result is 0. This, however, may make you wonder: Why do a fixed point? Is it not enough to evaluate the gates in topological order? The answer to this is that, in general, circuits may contain cycles.

2.3.3. Cycles & Constructivity

The circuits generated by the Esterel compiler may contain cycles. The semantics of cyclic circuits I present here is based on Malik (1994), Shiple et al. (1996), Mendler et al. (2012), and Berry (2002).

A circuit with a cycle may or may not be electrically stable. Wires which do not stabilize will have the value $\bot$ when a fixed-point is reached. Initially, all wires are in this state, as we do not yet know their value.

A circuit which does not stabilize is called non-constructive. Like constructivity in Esterel, this term is an allusion to Constructive Logic, a connection which (Mendler et al. 2012) formalizes. But to a first approximation: using three values for Booleans 1, 0, and $\bot$ is one way of formalizing a logic without the law of the excluded middle. That is, the circuit $X \vee \neg X$ may not always produce 1, but can also produce $\bot$.

Any circuit, even ones with a cycle, can be computed in finite time. On the electrical side of things, this is because, for constructive circuits, for any delay time in the computation of a gate there exists some clock time for which the circuit will always stabilize. On the logic side, the functions which define the gates are monotonic: once a value transitions from $\bot$ to 0 or 1 it can never change. This means that there is always a fixed-point when evaluating that circuit, and it should take no more iterations through the whole circuit to find that fixed-point than the number of gates in the circuit.

$$
\begin{aligned}
\mathfrak{c} &::= \mathbf{EQ} \times \mathbf{I} \times \mathbf{O} \\
\mathbf{I}, \mathbf{O} &::= (\mathbf{w} \ldots) \\
\mathbf{EQ} &::= \{(\mathbf{w} = \mathbf{e}) \ldots\} \\
\mathbf{B}_{\perp} &::= 0 \mid 1 \mid \perp \\
\mathbf{e} &::= \mathbf{w} \mid \mathbf{B} \\
&\quad \mid \neg \mathbf{e} \\
&\quad \mid \mathbf{e} \wedge \mathbf{e} \ldots \\
&\quad \mid \mathbf{e} \vee \mathbf{e} \ldots \\
w &\in \text{wire names}
\end{aligned}
$$

Figure 10: A Grammar for Circuits

2.3.4. Circuits, more formally

Now, on to a formal definition of a circuit. A circuit $\mathfrak{c}$ can be defined as a triple $\mathbf{EQ} \times \mathbf{I} \times \mathbf{O}$ where $\mathbf{I}$ is a set of names of input wires, $\mathbf{O}$ is a set of names of output wires, and $\mathbf{EQ}$ is a set of equations $(\mathbf{w} = \mathbf{e})$, which defines the internal wire named $\mathbf{w}$ by the expression $\mathbf{e}$, which is drawn from the grammar given in figure 10. Wire names are assumed to be unique.

Circuit evaluation takes place on a circuit state which is, in essence and environment for the circuit. It takes the form of a mapping from the wire names of the circuit to the current state of the circuit. I will denote a circuit state for a circuit $\mathfrak{c}$ as $\theta^{\mathfrak{c}}$. There is a special circuit state $\theta^{\mathfrak{c}}{}_0$ for every circuit in which every internal wire $\mathbf{w}$ which is not equal to a constant 0 or 1 is assigned the initial value $\perp$. Any wires in the set $\mathbf{I}$ or $\mathbf{O}$ which do not have a corresponding internal wire are given the value 0. For example, the circuit

$$\mathfrak{c} = \langle \{(\text{internal} = \text{input})\}, \{\text{input}\}, \{\text{output1}, \text{internal}\} \rangle$$

would have the initial state

$$\theta^{\mathfrak{c}}{}_0 = \{\{\text{internal} \mapsto \perp\}, \{\text{input} \mapsto 0\}, \{\text{output1} \mapsto \perp\}\}$$

We would write the XOr circuit from before as:

$$\mathfrak{c}_{xor} = \langle \{(a = \neg(i1 \wedge i2)\ ), (b = i1 \vee i2), (out = a \wedge b)\}, \{i1, i2\}, \{out\} \rangle$$

$$[\textbf{eval-wire}]\ \theta^{\mathfrak{C}} \rightharpoonup^{C} (\theta^{\mathfrak{C}} \leftarrow \{\ \mathbf{w} \mapsto \mathbf{B}\ \}) \quad \text{if } \mathbf{w} \in \mathit{dom}(\theta^{\mathfrak{C}}),\ \bot = \theta^{\mathfrak{C}}(\mathbf{w}),\ \theta^{\mathfrak{C}} \vdash \mathfrak{c}(\mathbf{w}) \hookrightarrow \mathbf{B}$$

Figure 11: Reduction relation for circuits

$$\frac{}{\theta^{\mathfrak{C}} \vdash \mathbf{B} \hookrightarrow \mathbf{B}}\,[\text{id}] \quad \frac{}{\theta^{\mathfrak{C}} \vdash \mathbf{w} \hookrightarrow \theta^{\mathfrak{C}}(\mathbf{w})}\,[\text{deref}]$$

$$\frac{\theta^{\mathfrak{C}} \vdash \mathbf{e} \hookrightarrow 0}{\theta^{\mathfrak{C}} \vdash \neg\mathbf{e} \hookrightarrow 1}\,[\text{not-0}] \quad \frac{\theta^{\mathfrak{C}} \vdash \mathbf{e} \hookrightarrow 1}{\theta^{\mathfrak{C}} \vdash \neg\mathbf{e} \hookrightarrow 0}\,[\text{not-1}]$$

$$\frac{\theta^{\mathfrak{C}} \vdash \mathbf{e}_1 \hookrightarrow 0}{\theta^{\mathfrak{C}} \vdash \mathbf{e}_1 \wedge \mathbf{e}_2 \hookrightarrow 0}\,[\text{and-0-left}] \quad \frac{\theta^{\mathfrak{C}} \vdash \mathbf{e}_2 \hookrightarrow 0}{\theta^{\mathfrak{C}} \vdash \mathbf{e}_1 \wedge \mathbf{e}_2 \hookrightarrow 0}\,[\text{and-0-right}] \quad \frac{\theta^{\mathfrak{C}} \vdash \mathbf{e}_1 \hookrightarrow 1 \qquad \theta^{\mathfrak{C}} \vdash \mathbf{e}_2 \hookrightarrow 1}{\theta^{\mathfrak{C}} \vdash \mathbf{e}_1 \wedge \mathbf{e}_2 \hookrightarrow 1}\,[\text{and-1}]$$

$$\frac{\theta^{\mathfrak{C}} \vdash \mathbf{e}_1 \hookrightarrow 1}{\theta^{\mathfrak{C}} \vdash \mathbf{e}_1 \vee \mathbf{e}_2 \hookrightarrow 1}\,[\text{or-1-left}] \quad \frac{\theta^{\mathfrak{C}} \vdash \mathbf{e}_2 \hookrightarrow 1}{\theta^{\mathfrak{C}} \vdash \mathbf{e}_1 \vee \mathbf{e}_2 \hookrightarrow 1}\,[\text{or-1-right}] \quad \frac{\theta^{\mathfrak{C}} \vdash \mathbf{e}_1 \hookrightarrow 0 \qquad \theta^{\mathfrak{C}} \vdash \mathbf{e}_2 \hookrightarrow 0}{\theta^{\mathfrak{C}} \vdash \mathbf{e}_1 \wedge \mathbf{e}_2 \hookrightarrow 0}\,[\text{or-0}]$$

Figure 12: Reduction relation for wire expressions

And we would write its initial state, when run on 1 and 1 to be:

$$\theta^{\mathfrak{C}_{xor}}{}_0 = \{\{\mathsf{a} \mapsto \bot\}, \{\mathsf{b} \mapsto \bot\}, \{\mathsf{out} \mapsto \bot\}, \{\mathsf{i1} \mapsto 1\}, \{\mathsf{i2} \mapsto 1\}\}$$

I will write $\mathfrak{c}(\mathbf{w})$ for accessing the expression in the circuit $\mathfrak{c}$ for the wire $\mathbf{w}$. I will write $\theta^{\mathfrak{C}}(\mathbf{w})$ for accessing the value of $\mathbf{w}$ in $\theta^{\mathfrak{C}}$. I will write $\mathit{dom}(\theta^{\mathfrak{C}})$ for the set of all wires defined in $\theta^{\mathfrak{C}}$.

The circuit can then be evaluated by the reduction relation $\rightharpoonup^{C}$, which is defined in figure 11. This reduction relation has only one rule, which selects one wire in the circuit which currently has the value $\bot$, and attempts to evaluate it using the relation $\theta^{\mathfrak{C}} \vdash \mathbf{e} \hookrightarrow \mathbf{B}$. This relation is adapted from Section 10.3.1 of *The Constructive Semantics of Pure Esterel (Draft Version 3)* (Berry 2002). This relation evaluates Boolean expressions, giving back a Boolean when the expression is constructive. The relation does not hold when the truth table in figure 9 would give back $\bot$.

The evaluator for circuits eval^{C} has the signature:

$$\mathit{eval}^{C} : \mathbf{O}\ \mathfrak{c} \rightarrow \langle \theta\ ,\ \mathbf{bool} \rangle$$

It takes in a set of wires $\mathbf{O}$ we wish to observe and a circuit, and fully evaluates the circuit by calling $\rightharpoonup^{C}$ until it reaches the fixed-point. The result a pair of θ and **bool**. The first part is a map θ that maps the wires in $\mathbf{O}$ to the values they have after evaluating the circuit. The second part is a Boolean which is *tt* if and only if the circuit has no wires which are $\perp$—that is it is true when the circuit is constructive. Otherwise it is *ff*.

Registers. This model can extend to registers by treating each register as a pair of an input and an output wire. Initially these input wires is set to 0, and on each subsequent cycle (e.g. each subsequent call to $eval^{C}$) these input wires are given the value of their corresponding output wire in the previous cycle.

Contextual Equivalence. I take the following to be the definition of contextual equivalence on circuits:

Definition: $\mathfrak{C}_1 \simeq^{C} \mathfrak{C}_2$

For all assignments to the inputs, and all output sets $\mathbf{O}$, $eval^{C}(\mathbf{O}, \mathfrak{C}_1) = \langle \theta , \mathbf{B} \rangle$ *if and only if* $eval^{C}(\mathbf{O}, \mathfrak{C}_2) = \langle \theta , \mathbf{B} \rangle$.

Intuitively, we can understand this to mean that the only observables of a circuit are the values of its output wires and if it is constructive, and the only observation a circuit can make about its context is the state of its input wires.

I base this definition on the procedure given by Malik (1994). This procedure is equivalent to the reduction relation I give here, which is proved by Mendler et al. (2012) and Berry (2002). Specifically Lemma 7 of Berry (2002) gives us that this reduction relation is equivalent to what Mendler et al. (2012) call ternary simulation of the circuits. Corollary 3 of Mendler et al. (2012) tells us that this is equivalent to the algorithm given by Malik (1994) for evaluating a circuit. Theorems 1, 2, 3, and 5 of Mendler et al. (2012) also give us that ternary simulation is equivalent to their UN-delay model of circuits, which is a model of electrical characteristics of circuit (See definition 6 in that paper). This UN-delay model is compositional, and thus can be used when analyzing a circuit without knowing its context.

Other definitions. I write $inputs(\mathfrak{C})$ to access the input set of the circuit, and $outputs(\mathfrak{C})$ for the output set.

At times the expression of a single wire will be equivalent in a particular circuit to some other expression. I will describe this with the notation:

Definition: $\mathfrak{C}(\mathbf{w}) \simeq \mathbf{e}$

$\mathcal{C}$ *is contextually equivalent to a circuit in which the definition of the wire* $\mathbf{w}$ *is replace by* $\mathbf{e}$.

which equates the wire $\mathbf{w}$ of the circuit $\mathcal{C}$ to the expression $\mathbf{e}$. For example $\mathcal{C}_1(\mathbf{w}_1) \simeq 1$ says that in all situations the wire $\mathbf{w}$ in the circuit $\mathcal{C}_1$ will have the value 1, and $\mathcal{C}_2(\mathbf{w}_2) \simeq \mathcal{C}_3(\mathbf{w}_2)$ says that not matter what, the $\mathbf{w}_2$ wire in both circuits will always have the same value.

CHAPTER 3

Loop-free, pure Esterel

With the background out of the way, this section dives directly into describing the calculus for Esterel. Specifically this describes the calculus for single instants of Pure Esterel without loops. This section relies heavily on the background given in section 2.1 and section 2.2.

3.1. The Constructive Calculus

This section will walk through the rules of the calculus to explain their function. The calculus is built around the relation $\rightharpoonup^{\mathrm{E}}$, which defines the notions of reduction for the equality relation $\equiv^{\mathrm{E}}$. These relations work within a single instant of execution, which leads to an evaluator $\mathit{eval}^{\mathrm{E}}$ which evaluates a single instant. Multi-instant evaluation is described in section 5.3.

The rules of $\rightharpoonup^{\mathrm{E}}$, broadly, be broken up into two categories: Administrative reductions which shuffle the term around to find the next redex; and Signal Reductions, which manipulate and read signal states. The description here is incremental, introducing concepts as it goes along. The complete rules and grammars can be found in one place in Appendix A.

3.1.1. Administrative rules

To begin, the administrative rules rely on three categories of programs that represent various ways a program fragment may be in a completed state:

$$
\begin{aligned}
\mathbf{p}^{\mathrm{D}} &::= \mathbf{p}^{\mathrm{S}} \mid \hat{\mathbf{p}} \\
\mathbf{p}^{\mathrm{S}} &::= \mathsf{nothing} \mid (\mathsf{exit}\ \mathbf{n}) \\
\hat{\mathbf{p}} &::= \mathsf{pause} \\
&\quad \mid (\mathsf{seq}\ \hat{\mathbf{p}}\ \mathbf{q}) \\
&\quad \mid (\overline{\mathsf{loop}}\ \hat{\mathbf{p}}\ \mathbf{q}) \\
&\quad \mid (\mathsf{par}\ \hat{\mathbf{p}}\ \hat{\mathbf{p}}) \\
&\quad \mid (\mathsf{suspend}\ \hat{\mathbf{p}}\ \mathbf{S}) \\
&\quad \mid (\mathsf{trap}\ \hat{\mathbf{p}})
\end{aligned}
$$

Stopped terms ($\mathbf{p}^{\mathrm{S}}$) can no longer evaluate and will do nothing further in future instants. Paused terms ($\hat{\mathbf{p}}$) are terms which will not reduce further this instant, but will evaluate further in future instants. Done terms ($\mathbf{p}^{\mathrm{D}}$) are stopped or paused. A $\mathbf{p}^{\mathrm{D}}$ term is analogous to a value in other languages. A $\mathbf{p}^{\mathrm{S}}$ is analogous to a primitive value (e.g. a number), in that it is atomic, and contains no future behaviors. A $\hat{\mathbf{p}}$ term is analogous to a λ term in the λ-calculus in that it is a value which is awaiting input, and once that input is received it can continue reducing. A $\hat{\mathbf{p}}$ term, unlike λ, cannot be immediately be given inputs by the program—rather its inputs are provided by the outer context in the next instant.

The first two rules deal with sequencing:

$$
\begin{aligned}
&[\textbf{seq-done}]\ (\mathsf{seq}\ \mathsf{nothing}\ \mathbf{q}) \rightharpoonup^{\mathrm{E}} \mathbf{q} \\
&[\textbf{seq-exit}]\ \ (\mathsf{seq}\ (\mathsf{exit}\ \mathbf{n})\ \mathbf{q}) \rightharpoonup^{\mathrm{E}} (\mathsf{exit}\ \mathbf{n})
\end{aligned}
$$

The first rule reduces to the second part of the sequence when the first part has completed and will not preempt the whole sequence. The second rule preempts the sequence when the first part reduces to an exit, by discarding the second part of the seq and reducing to the exit.

The next rule handles traps:

$$
[\textbf{trap}]\ (\mathsf{trap}\ \mathbf{p}^{\mathrm{S}}) \rightharpoonup^{\mathrm{E}} \downarrow^{\mathrm{p}} \mathbf{p}^{\mathrm{S}}
$$

This rule reduces when the inner program can reduce no more, via the metafunction:

$$
\begin{aligned}
\downarrow^{\mathrm{p}} &: \mathbf{p}^{\mathrm{S}} \to \mathbf{p}^{\mathrm{S}} \\
\downarrow^{\mathrm{p}} \mathsf{nothing} &= \mathsf{nothing} \\
\downarrow^{\mathrm{p}} (\mathsf{exit}\ 0) &= \mathsf{nothing} \\
\downarrow^{\mathrm{p}} (\mathsf{exit}\ \mathbf{n}) &= (\mathsf{exit}\ \mathbf{n}\text{-}1)
\end{aligned}
$$

which will decrement a exit by one, unless the exit is bound by this trap, in which case it reduces to nothing, allowing execution to continue from this point.

The next rules handle par:

$$
\begin{array}{ll}
[\textbf{par-swap}] & (\text{par } \mathbf{p}\ \mathbf{q}) \rightharpoonup^{E} (\text{par } \mathbf{q}\ \mathbf{p}) \\
[\textbf{par-nothing}] & (\text{par nothing } \mathbf{p}^{D}) \rightharpoonup^{E} \mathbf{p}^{D} \\
[\textbf{par-1exit}] & (\text{par } (\text{exit } \mathbf{n})\ \hat{\mathbf{p}}) \rightharpoonup^{E} (\text{exit } \mathbf{n}) \\
[\textbf{par-2exit}] & (\text{par } (\text{exit } \mathbf{n}_1) (\text{exit } \mathbf{n}_2)) \rightharpoonup^{E} (\text{exit } \mathit{max}(\mathbf{n}_1, \mathbf{n}_2))
\end{array}
$$

The first rule swaps the two branches of a par. This rule is useful for exposing redexes to the next two rules. The second rule allows a par to reduce to its second branch when it is $\mathbf{p}^{D}$ and the other branch has completed. Combined with [**par-swap**], it means that the program will progress with the behavior of one branch if the other is nothing. The last two rules handle exits in pars. In short, an exit will preempt a paused term, and two exits will abort to whichever one is bound higher up.

Note that all of the par administrative reductions only take effect when both branches have completed. This is because pars acts akin to a fork/join, synchronizing the results of both branches, which gives us determinism in that we cannot observe which branch completes first.

Next up is suspend:

$$[\textbf{suspend}]\ (\text{suspend } \mathbf{p}^{S}\ \mathbf{S}) \rightharpoonup^{E} \mathbf{p}^{S}$$

Which just states that the suspension of a $\mathbf{p}^{S}$ term is equivalent to just that term. This is because $\rightharpoonup^{E}$ only works within one instant, and suspend has different behaviors on initial versus future instants. This is the only rule that touches suspend. The rest of suspend's behavior is not handled by $\rightharpoonup^{E}$, but is rather handled by the inter-instant translation function $\mathcal{E}$, which is discussed in section 5.3.

3.1.2. Signal rules

The signal rules are more subtle than the administrative rules. They must reason about state, which is difficult to do in a local way. To handle this, we need to add three new pieces: Environments, Evaluation Contexts, and the metafunction *Can*.

Environments. Like in the state calculus, environments represent local state information. In the constructive calculus environments look like:

$$\begin{aligned}
\mathbf{p}, \mathbf{q} &::= .... \mid (\varrho\ \langle \theta^{\mathrm{r}}, \mathbf{A} \rangle.\ \mathbf{p}) \\
\mathbf{status} &::= 1 \mid 0 \mid \bot \\
\mathbf{status}^{\mathrm{r}} &::= 1 \mid \bot \\
\mathbf{A} &::= \mathrm{GO} \mid \mathrm{WAIT} \\
\theta &: \mathbf{S} \longrightarrow \mathbf{status} \\
\theta^{\mathrm{r}} &: \mathbf{S} \longrightarrow \mathbf{status}^{\mathrm{r}}
\end{aligned}$$

Local environments ϱ contain two parts: a map θ^{r}, and a control variable $\mathbf{A}$. The information contained in these environments is scoped to the program fragment $\mathbf{p}$. The map θ^{r} maps signals that are in scope of the term $\mathbf{p}$ to their status. The maps used for local stores are restricted maps θ^{r}, which only map to a subset of signal statuses. Other parts of the calculus will use full maps θ.[1]

The control variable $\mathbf{A}$ tracks whether or not control has reached this point in the program. This control variable is needed because signal emissions represent what must happen in the program. However, as discussed in section 2.1.3, this is inherently a non-local property. Therefore we add a new piece to the environment, $\mathbf{A}$, which will be GO if control will reach the portion of the program inside the ϱ, or WAIT, if it might not. In essence this tracks a 1 propagating through the control edges of the causality graphs discussed in section 2.1.3.

To understand why this is needed in the setting of the calculus specifically, consider the program in figure 13. This program has a cycle between the test of S1, the test of S2, and the emit of S1. This cycle cannot be broken, therefore this program is non-constructive: evaluation would demand a value for S1 before determining a value for S2, which

[1] You may notice that these three statuses correspond to wire values in Circuits. This is because signals correspond exactly to wire in compilation, and this fact will be crucial in proving soundness of the calculus.

```
(signal S1
  (if S1
     (signal S2
       (seq (emit S2)
            (if S2 nothing (emit S1))))
     nothing))
```

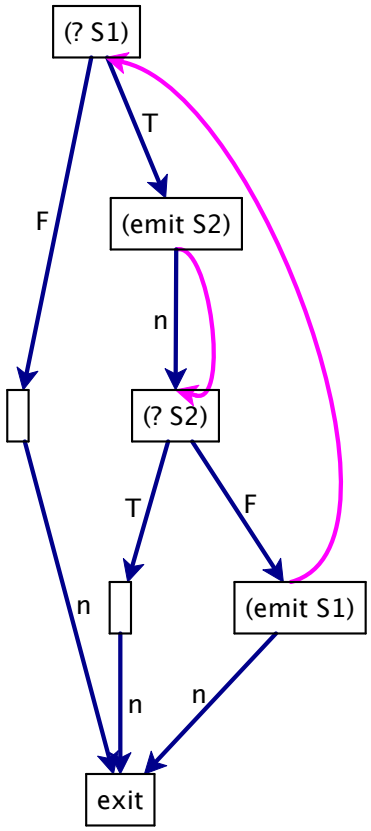


Figure 13: A non-constructive program

cannot happen. However we might try to reason about a fragment of this program locally, ignoring it's context. For example we might ignore the context:

$$(\text{signal S1 } (\text{if S1 } \bigcirc \text{ nothing}))$$

and focus on the fragment

```
(signal S2
  (seq (emit S2)
       (if S2 nothing (emit S1))))
```

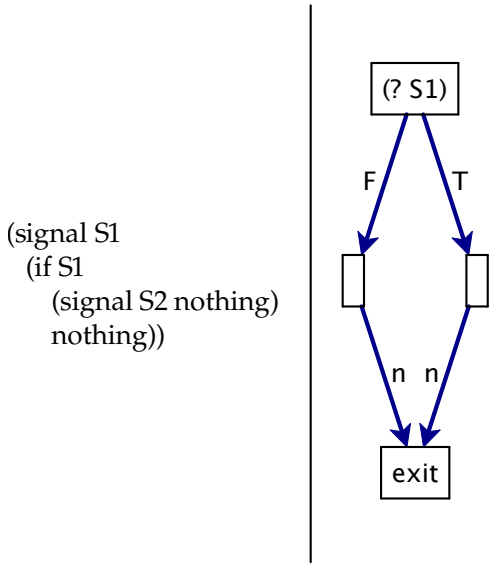


Figure 14: Breaking the cycle, illegally

If we live in a world without the control variable **A**, then we must assume that control reaches this part of the program. Therefore we can emit the S2, allowing the fragment to reduce to

(signal S2 nothing)

which, when plugged back into its context gives us the program in figure 14. But this program no longer has the non-constructive cycle! Therefore this local reasoning was not valid: we did not know that the (emit S2) must be reached, so it was not safe to emit it.

But when using a calculus we can never assume that we have full knowledge of the program: there may always been an outer context, meaning we can never know for sure if a term will be reached or not. To handle this the control variable **A** adds local information that tells us if the program term must be reached or not. When **A** is GO, this means that the term *must* be executed. If **A** is WAIT the term may or may not be executed.[2]

[2]These control variables are adapted from the microstep semantics of Berry and Rieg (2019). This semantics defines an evaluator for Esterel which tracks execution state via three colors: Red (0/Cannot), Green (1/Must), and Gray(⊥/unknown). My adaptation makes these colors local, which allows the Red color to be discarded. Green corresponds to GO, and Grey corresponds to WAIT.

The calculus itself will never introduce GO's, but rather only propagate them through the program. A GO can only safely be introduced by the evaluator—as it knows the whole program—and, theoretically, when the whole program is guaranteed to be acyclic. However, the calculus assumes that GO is only at the top of the program, and therefore while a programmer may choose to add GO to acyclic programs, doing so is not proven to be sound.

To understand why restricted maps θ^{r} are necessary, cast your eye back to figure 6 from section 2.1.3. That *must* and *cannot* corner of that diagram is an nonsensical state. If we used unrestricted maps for environments, however, the syntax of the language would allow for representing such program. Consider the program

$$(\varrho\ \langle\{\ \mathrm{S1} \mapsto 0\ \}, \mathrm{GO}\rangle.\ (\mathrm{emit\ S1}))$$

The GO and (emit S1) tells us that this program *must* emit S1. However the 0 in the environment tells us that S1 *cannot* be emitted. This is the exact contradiction we wish to avoid. Therefore the calculus forbids directly recording 0 in the environment. While such a program should never be reachable from a program without environments, it makes proofs about the calculus simpler to exclude such programs from the grammar altogether. Section 3.1.2.3 explains how 0 is recorded in the calculus.

Note that a term which swaps things around, recording that something *must* be emitted in a program that *cannot* emit it (e.g. $(\varrho\ \langle\{\ \mathrm{S1} \mapsto 1\ \}, \mathrm{GO}\rangle.\ \mathrm{nothing})$) does not contain a contraction. This is because the 1 in the environment records that at some point in the reduction sequence prior to the current state S1 must have been emitted. Therefore it is the case that this program actually states that S1 must be emitted (and resp. *can* be emitted). This is a manifestation of the asymmetry between *must* and *can*.

A small example of how environments work can be seen in the rule:

$$[\mathbf{signal}]\ (\mathrm{signal}\ \mathbf{S}\ \mathbf{p}) \rightharpoonup^{\mathrm{E}} (\varrho\ \langle\{\ \mathbf{S} \mapsto \bot\ \}, \mathrm{WAIT}\rangle.\ \mathbf{p})$$

which transforms a signal into a local environment. The map in this environment contains only the signal, mapped to $\bot$, representing that we do not yet know its value. The control variable is set to WAIT as we cannot know if this program fragment will be executed yet or not.

Evaluation Contexts. The next set of rules require evaluation contexts. Like the evaluation contexts we saw in section 2.2, these control where evaluation may take place (and therefore where state is valid), however, in this case the evaluation contexts can decompose non-deterministically because of par:

$$\begin{aligned}\mathbf{E} ::= {} & \bigcirc \\ & \mid (\text{seq } \mathbf{E}\ \mathbf{q}) \\ & \mid (\text{par } \mathbf{E}\ \mathbf{q}) \\ & \mid (\text{par } \mathbf{p}\ \mathbf{E}) \\ & \mid (\text{suspend } \mathbf{E}\ \mathbf{S}) \\ & \mid (\text{trap } \mathbf{E})\end{aligned}$$

With these in hand we can now look at the next three rules. Firstly, local environments may be merged together if they are within an evaluation context of each other:

$$\begin{aligned}&[\mathbf{merge}]\ (\varrho\ \langle \theta^{r}{}_{1}, \mathbf{A}_{1}\rangle.\ \mathbf{E}[(\varrho\ \langle \theta^{r}{}_{2}, \mathbf{A}_{2}\rangle.\ \mathbf{p})]) \rightharpoonup^{E} (\varrho\ \langle(\theta^{r}{}_{1} \leftarrow \theta^{r}{}_{2}), \mathbf{A}_{1}\rangle.\ \mathbf{E}[\mathbf{p}]) \\ &\qquad \text{if } \mathbf{A}_{1} \geq \mathbf{A}_{2}\end{aligned}$$

This merge overwrites bindings in the outer map with the inner one. In addition this merge may only happen if it would not expose the outer environment to more control information that it had before. That is, GO $\geq$ WAIT. So the merge will happen if the outer environment has a GO, or if both environments have a WAIT.

Next we may emit a signal when that signal is in an evaluation context relative to its binder, and when we know control will reach this point in the program:

$$\begin{aligned}&[\mathbf{emit}]\ (\varrho\ \langle \theta^{r}, \text{GO}\rangle.\ \mathbf{E}[(\text{emit } \mathbf{S})]) \rightharpoonup^{E} (\varrho\ \langle(\theta^{r} \leftarrow \{\ \mathbf{S} \mapsto 1\ \}), \text{GO}\rangle.\ \mathbf{E}[\text{nothing}]) \\ &\qquad \text{if } \mathbf{S} \in \mathit{dom}(\theta^{r})\end{aligned}$$

Emission is accomplished by updating the environment to map $\mathbf{S}$ to 1, and replacing the emit with nothing.

Once there is a 1 in the environment we may reduce to the then branch of a conditional:

$$[\mathbf{is\text{-}present}]\ (\varrho\ \langle \theta^{r}, \mathbf{A}\rangle.\ \mathbf{E}[(\text{if } \mathbf{S}\ \mathbf{p}\ \mathbf{q})]) \rightharpoonup^{E} (\varrho\ \langle \theta^{r}, \mathbf{A}\rangle.\ \mathbf{E}[\mathbf{p}])\ \text{if } \theta^{r}(\mathbf{S}) = 1$$

These [**emit**] and [**is-present**] rules together describe how the calculus handles signals that *must* be emitted. The handling of signals that *cannot* be emitted requires a different mechanism altogether.

$\boldsymbol{\kappa} \in$ Natural Numbers

$$
\begin{array}{|l|}
\hline
\downarrow^{\kappa} : \boldsymbol{\kappa} \to \boldsymbol{\kappa} \\
\downarrow^{\kappa} 0 = 0 \\
\downarrow^{\kappa} 1 = 1 \\
\downarrow^{\kappa} 2 = 0 \\
\downarrow^{\kappa} \mathbf{n} = \mathbf{n}\text{-}1 \\
\text{if } \mathbf{n} > 2 \\
\hline
\end{array}
$$

$$
\begin{array}{ll}
\mathit{Can} : \mathbf{p}\ \theta \to \{ \mathrm{S}: \mathbb{S}, \mathrm{K}: \mathbb{k}, \mathrm{sh}: \mathbb{s} \} & \\
\mathit{Can}(\text{nothing}, \theta) & = \{ \mathrm{S} = \varnothing, \mathrm{K} = \{ 0 \}, \mathrm{sh} = \varnothing \} \\
\mathit{Can}(\text{pause}, \theta) & = \{ \mathrm{S} = \varnothing, \mathrm{K} = \{ 1 \}, \mathrm{sh} = \varnothing \} \\
\mathit{Can}((\text{exit } \mathbf{n}), \theta) & = \{ \mathrm{S} = \varnothing, \mathrm{K} = \{ \mathbf{n} + 2 \}, \mathrm{sh} = \varnothing \} \\
\mathit{Can}((\text{emit } \mathbf{S}), \theta) & = \{ \mathrm{S} = \{ \mathbf{S} \}, \mathrm{K} = \{ 0 \}, \mathrm{sh} = \varnothing \} \\
\mathit{Can}((\text{if } \mathbf{S}\ \mathbf{p}\ \mathbf{q}), \theta) & = \mathit{Can}(\mathbf{p}, \theta) \\
\text{if } \theta(\mathbf{S}) = 1 & \\
\mathit{Can}((\text{if } \mathbf{S}\ \mathbf{p}\ \mathbf{q}), \theta) & = \mathit{Can}(\mathbf{q}, \theta) \\
\text{if } \theta(\mathbf{S}) = 0 & \\
\mathit{Can}((\text{if } \mathbf{S}\ \mathbf{p}\ \mathbf{q}), \theta) & = \{\ \mathrm{S} = \mathit{Can}^{\mathrm{S}}(\mathbf{p}, \theta) \cup \mathit{Can}^{\mathrm{S}}(\mathbf{q}, \theta), \\
 & \quad \mathrm{K} = \mathit{Can}^{\mathrm{K}}(\mathbf{p}, \theta) \cup \mathit{Can}^{\mathrm{K}}(\mathbf{q}, \theta), \\
 & \quad \mathrm{sh} = \mathit{Can}^{\mathrm{sh}}(\mathbf{p}, \theta) \cup \mathit{Can}^{\mathrm{sh}}(\mathbf{q}, \theta)\ \} \\
\mathit{Can}((\text{suspend } \mathbf{p}\ \mathbf{S}), \theta) & = \mathit{Can}(\mathbf{p}, \theta) \\
\mathit{Can}((\text{seq } \mathbf{p}\ \mathbf{q}), \theta) & = \mathit{Can}(\mathbf{p}, \theta) \\
\text{if } 0 \notin \mathit{Can}^{\mathrm{K}}(\mathbf{p}, \theta) & \\
\mathit{Can}((\text{seq } \mathbf{p}\ \mathbf{q}), \theta) & = \{\ \mathrm{S} = \mathit{Can}^{\mathrm{S}}(\mathbf{p}, \theta) \cup \mathit{Can}^{\mathrm{S}}(\mathbf{q}, \theta), \\
 & \quad \mathrm{K} = \mathit{Can}^{\mathrm{K}}(\mathbf{p}, \theta) \setminus \{ 0 \} \cup \mathit{Can}^{\mathrm{K}}(\mathbf{q}, \theta), \\
 & \quad \mathrm{sh} = \mathit{Can}^{\mathrm{sh}}(\mathbf{p}, \theta) \cup \mathit{Can}^{\mathrm{sh}}(\mathbf{q}, \theta)\ \} \\
\mathit{Can}((\text{par } \mathbf{p}\ \mathbf{q}), \theta) & = \{\ \mathrm{S} = \mathit{Can}^{\mathrm{S}}(\mathbf{p}, \theta) \cup \mathit{Can}^{\mathrm{S}}(\mathbf{q}, \theta), \\
 & \quad \mathrm{K} = \{\ \mathit{max}(\boldsymbol{\kappa}_1, \boldsymbol{\kappa}_2) \mid \kappa_1 \in \mathit{Can}^{\mathrm{K}}(\mathbf{p}, \theta)\ , \kappa_2 \in \mathit{Can}^{\mathrm{K}}(\mathbf{q}, \theta)\ \}, \\
 & \quad \mathrm{sh} = \mathit{Can}^{\mathrm{sh}}(\mathbf{p}, \theta) \cup \mathit{Can}^{\mathrm{sh}}(\mathbf{q}, \theta)\ \} \\
\mathit{Can}((\text{signal } \mathbf{S}\ \mathbf{p}), \theta) & = \{\ \mathrm{S} = \mathit{Can}^{\mathrm{S}}(\mathbf{p}, \theta \leftarrow \{ \mathbf{S} \mapsto 0 \}) \setminus \{ \mathbf{S} \}, \\
 & \quad \mathrm{K} = \mathit{Can}^{\mathrm{K}}(\mathbf{p}, \theta \leftarrow \{ \mathbf{S} \mapsto 0 \}), \\
 & \quad \mathrm{sh} = \mathit{Can}^{\mathrm{sh}}(\mathbf{p}, \theta \leftarrow \{ \mathbf{S} \mapsto 0 \})\ \} \\
\text{if } \mathbf{S} \notin \mathit{Can}^{\mathrm{S}}(\mathbf{p}, \theta \leftarrow \{ \mathbf{S} \mapsto \bot \}) & \\
\mathit{Can}((\text{signal } \mathbf{S}\ \mathbf{p}), \theta) & = \{ \mathrm{S} = \mathit{Can}^{\mathrm{S}}(\mathbf{p}, \theta_2) \setminus \{ \mathbf{S} \}, \mathrm{K} = \mathit{Can}^{\mathrm{K}}(\mathbf{p}, \theta_2), \mathrm{sh} = \mathit{Can}^{\mathrm{sh}}(\mathbf{p}, \theta_2)\ \} \\
\text{if } \theta_2 = \theta \leftarrow \{ \mathbf{S} \mapsto \bot \} & \\
\mathit{Can}((\varrho\ \langle \theta_1, \mathbf{A} \rangle .\ \mathbf{p}), \theta_2) & = \{\ \mathrm{S} = \mathit{Can}^{\mathrm{S}}_{\varrho}((\varrho\ \langle \theta_1, \mathbf{A} \rangle .\ \mathbf{p}), \theta_2) \setminus \mathit{dom}(\theta_1), \\
 & \quad \mathrm{K} = \mathit{Can}^{\mathrm{K}}_{\varrho}((\varrho\ \langle \theta_1, \mathbf{A} \rangle .\ \mathbf{p}), \theta_2), \\
 & \quad \mathrm{sh} = \mathit{Can}^{\mathrm{sh}}_{\varrho}((\varrho\ \langle \theta_1, \mathbf{A} \rangle .\ \mathbf{p}), \theta_2) \setminus \mathit{dom}(\theta_1)\ \}
\end{array}
$$

Figure 15: Can on pure, loop free terms

Can. If 0 cannot be put intro restricted environments, how are we to take the else branch? The answer lies the meaning of 0. A signal is 0 only when it has not been emitted (that is, is not 1) and *cannot* be emitted. Thus to take the else branch we analyze the program for what can be emitted. This is done by the metafunctions in figure 15 and figure 16.

The first metafunction, *Can*, computes what might happen during the execution of a program, given an environment of signals. The metafunction *Can* returns three sets. The first set S is a set of the signals that might be emitted during execution. The second set K is a set of return codes (κ), which describe in what ways the program might terminate. The code 0 means the program may reduce to nothing. The code 1 means the program might pause (reduce to $\hat{\mathbf{p}}$). A code of $\kappa = 2$ or greater means the program might reduce to (exit κ - 2). The final set sh is a set of shared variables which may be written to during execution of the program. This third set is discussed when the host language portion of the calculus is introduced in section 5.2.1.

I will denote accessing only one of these sets with a superscript: Can^{S} for the S set, Can^{K} for the K set, and Can^{sh} for the sh set.

Note that *Can* takes in a map θ not a restricted map θ^{r}. While *Can* will record 0s into this map, it cannot arrive at a contraction. This is because it only records a signal **S** as 0 in the map if **S** cannot be emitted, therefore it cannot enter the contradictory corner of figure 6.

While I explain this version of *Can* here, a much more detailed explanation can be found in chapters 4 and 5 of *Compiling Esterel* (Potop-Butucaru et al. 2007), from which this version of *Can* is adapted.

The first three clauses of *Can* handle the return codes for irreducible terms: nothing gets 0, etc. The S and sh sets are empty for these, as they can neither emit signals nor write to shared variables.

The next clause, for emit, puts **S** in the S set, and 0 in the K set, as (emit **S**) can reduce to only nothing, and can emit only **S**.

The next three clauses handle if. When θ knows that **S** is 1, then *Can* will only inspect the **p** branch, as the **q** branch cannot be reached. The reverse is true when θ maps **S** to 0. Otherwise, both branches are analyzed and, as both branches can happen, their result sets are unioned.

The next clause handles suspend, which just gives the result of analyzing the body of the suspend. This is because suspend does nothing on the first instant, and the inter-instant metafunction $\mathcal{E}$ will transform suspend into other forms, therefore no special reasoning is needed.

The next two clauses handle seq. If 0 is not in the possible return codes of the first part of the seq, we know that it cannot reduce to nothing, therefore the **q** can never be reached. Therefore in this case *Can* only analyzes **p**. If 0 is in the possible return codes of **p**, *Can* analyzes both parts, and unions the results. However 0 is removed from the return codes of **p**, as if **p** does in fact reduce to nothing then the return code will given by only **q**.

Next is par. The S and sh sets are just the union of the sets from the recursive calls. The return codes are give by the set of pairwise max of each possible return code of the subterm. This mimics exactly what the **[par-nothing]**, **[par-1exit]**, and **[par-2exit]** rules do.

The next clause handles trap. Again the S and sh sets are the same as the sets for the subterm. The return codes are given by the metafunction $\downarrow^{\kappa}$, which does for return codes what $\downarrow^{p}$ does for terms.

The next two clauses handle signal forms. If the signal **S** cannot be emitted by the body **p**, then the term is re-analyzed with **S** set to 0, as this refined information may give a more accurate result of what the term can do. Otherwise the recursive call is used as is. In both cases **S** is removed from the result set, as its name may not be unique and thus emissions from within this signal form need to be hidden to avoid conflicts with other signals of the same name.

The clause for ϱ delegates to the Can_{ϱ} metafunction. Like the signal case, it removes all of its bound variables from its result. The Can_{ϱ} function handles the analysis of ϱ forms. It essentially behaves as if the form was made of nested signal forms: for each signal, if the signal is $\bot$ and not in the S set of the recursive call then the form is re-analyzed with that signal set to 0. Otherwise the signal's value remains unchanged. The primary difference between this and the signal rule is that the bound variables are not removed from the resulting S set—this is handled by *Can*. We can understand why this is by looking at the rule which uses Can_{ϱ}:

$$\textbf{[is-absent]}\ (\varrho\ \langle \theta^{r}, \mathbf{A}\rangle.\ \mathbf{E}[(\text{if } \mathbf{S}\ \mathbf{p}\ \mathbf{q})]) \rightharpoonup^{E} (\varrho\ \langle \theta^{r}, \mathbf{A}\rangle.\ \mathbf{E}[\mathbf{q}]) \quad \text{if } \mathbf{S} \in \mathit{dom}(\theta^{r}),\ \theta^{r}(\mathbf{S}) = \bot,\ \mathbf{S} \notin \mathit{Can}^{S}_{\varrho}((\varrho\ \langle \theta^{r}, \mathbf{A}\rangle.\ \mathbf{E}[(\text{if } \mathbf{S}\ \mathbf{p}\ \mathbf{q})]), \{\})$$

This rule says that we may take the else branch of a conditional only when the signal is bound in an environment in a relative evaluation context to the conditional, an the signal cannot be emitted by the program. If the signals in θ^{r} were removed from the result of Can_{ϱ} this rule would always fire.

$$
\begin{array}{l}
\mathit{Can}_{\varrho} : (\varrho\ \langle \theta, \mathbf{A} \rangle.\ \mathbf{p})\ \theta \to \{\ \mathrm{S}: \mathbb{S}, \mathrm{K}: \mathbb{k}, \mathrm{sh}: \mathbb{s}\ \} \\
\mathit{Can}_{\varrho}(\!|(\varrho\ \langle \theta, \mathbf{A} \rangle.\ \mathbf{p}), \theta_2|\!) \quad = \quad \mathit{Can}_{\varrho}(\!|(\varrho\ \langle (\theta \setminus \{\mathbf{S}\}), \mathbf{A} \rangle.\ \mathbf{p}), \theta_2 \leftarrow \{\ \mathbf{S} \mapsto 0\ \}|\!) \\
\text{if } \mathbf{S} \in \mathit{dom}(\!|\theta|\!), \\
\quad \theta(\mathbf{S}) = \bot, \\
\quad \mathbf{S} \notin \overset{S}{\mathit{Can}}_{\varrho}(\!|(\varrho\ \langle (\theta \setminus \{\mathbf{S}\}), \mathbf{A} \rangle.\ \mathbf{p}), \theta_2 \leftarrow \{\ \mathbf{S} \mapsto \bot\ \}|\!) \\
\mathit{Can}_{\varrho}(\!|(\varrho\ \langle \theta, \mathbf{A} \rangle.\ \mathbf{p}), \theta_2|\!) \quad = \quad \mathit{Can}_{\varrho}(\!|(\varrho\ \langle (\theta \setminus \{\mathbf{S}\}), \mathbf{A} \rangle.\ \mathbf{p}), \theta_2 \leftarrow \{\ \mathbf{S} \mapsto \theta(\mathbf{S})\ \}|\!) \\
\text{if } \mathbf{S} \in \mathit{dom}(\!|\theta|\!) \\
\mathit{Can}_{\varrho}(\!|(\varrho\ \langle \theta_1, \mathbf{A} \rangle.\ \mathbf{p}), \theta_2|\!) \quad = \quad \mathit{Can}(\!|\mathbf{p}, \theta_2|\!)
\end{array}
$$

Figure 16: Can_{ϱ}

$$
\frac{\mathbf{p} \rightharpoonup^{\mathrm{E}} \mathbf{q}}{\mathbf{p} \equiv^{\mathrm{E}} \mathbf{q}}\,[\mathrm{step}] \quad \frac{}{\mathbf{p} \equiv^{\mathrm{E}} \mathbf{p}}\,[\mathrm{refl}]
$$

$$
\frac{\mathbf{q} \equiv^{\mathrm{E}} \mathbf{p}}{\mathbf{p} \equiv^{\mathrm{E}} \mathbf{q}}\,[\mathrm{sym}] \quad \frac{\mathbf{p} \equiv^{\mathrm{E}} \mathrm{r} \qquad \mathrm{r} \equiv^{\mathrm{E}} \mathbf{q}}{\mathbf{p} \equiv^{\mathrm{E}} \mathbf{q}}\,[\mathrm{trans}] \quad \frac{\mathbf{p} \equiv^{\mathrm{E}} \mathbf{q}}{\mathbf{C}[\mathbf{p}] \equiv^{\mathrm{E}} \mathbf{C}[\mathbf{q}]}\,[\mathrm{ctx}]
$$

Figure 17: The full equality relation

3.1.3. The equality relation

As a calculus should be congruent equality relation, the relation $\rightharpoonup^{\mathrm{E}}$ generates this relation via its symmetric, transitive, reflexive, compatible closure, seen in figure 17.

3.2. The evaluator

The evaluator defined by the calculus is a partial function which evaluates one instant of execution. Its signature is similar to that of the circuit evaluator $\mathit{eva}/^{\mathrm{C}}$:

$$\mathit{eva}/^{\mathrm{E}} : \mathbf{O}\ \mathbf{p} \to \langle \theta\ ,\ \mathbf{bool} \rangle$$

It takes a set of output signals and a program, and gives back a pair containing a map with the status of each of those signals and a Boolean which tells us if the program is constructive or not. The evaluator itself has two clauses, the first clause handling constructive programs, and the second clause handling non-constructive programs:

$$\begin{array}{l} \mathit{eval}/^{\mathrm{E}}(\mathbf{O}, (\varrho\ \langle \theta^{\mathrm{r}}{}_1, \mathrm{GO}\rangle.\ \mathbf{p}^{\mathrm{p}})) = \langle \mathit{restrict}(\theta^{\mathrm{r}}{}_2, \mathbf{O}, (\varrho\ \langle \theta^{\mathrm{r}}{}_2, \mathrm{GO}\rangle.\ \mathbf{p}^{\mathrm{D}})),\ \mathit{tt}\rangle \\ \text{if } (\varrho\ \langle \theta^{\mathrm{r}}{}_1, \mathrm{GO}\rangle.\ \mathbf{p}^{\mathrm{p}}) \equiv^{\mathrm{E}} (\varrho\ \langle \theta^{\mathrm{r}}{}_2, \mathrm{GO}\rangle.\ \mathbf{p}^{\mathrm{D}}),\ \mathit{complete\text{-}wrt}(\theta^{\mathrm{r}}{}_2, \mathbf{p}^{\mathrm{D}}) \\ \mathit{eval}/^{\mathrm{E}}(\mathbf{O}, (\varrho\ \langle \theta^{\mathrm{r}}{}_1, \mathrm{GO}\rangle.\ \mathbf{p}^{\mathrm{p}})) = \langle \mathit{restrict}(\theta^{\mathrm{r}}{}_2, \mathbf{O}, (\varrho\ \langle \theta^{\mathrm{r}}{}_2, \mathrm{GO}\rangle.\ \mathbf{q}^{\mathrm{p}})),\ \mathit{ff}\rangle \\ \text{if } (\varrho\ \langle \theta^{\mathrm{r}}{}_1, \mathrm{GO}\rangle.\ \mathbf{p}^{\mathrm{p}}) \equiv^{\mathrm{E}} (\varrho\ \langle \theta^{\mathrm{r}}{}_2, \mathrm{GO}\rangle.\ \mathbf{q}^{\mathrm{p}}),\ \theta^{\mathrm{r}}{}_2; \mathrm{GO}; \bigcirc \vdash_{\mathrm{B}} \mathbf{q}^{\mathrm{p}} \end{array}$$

If a program is $\equiv^{\mathrm{E}}$ to another program which is done ($\mathbf{p}^{\mathrm{D}}$), and that program has an environment which is *complete* with respect to that program, then, the program is constructive. The *complete-wrt* relation holds if every signal is either set to 1, or is set to $\perp$ and that signal is not in the result of $\mathit{Can}^{\mathrm{S}}_{\varrho}$:

Definition: $\mathit{complete\text{-}wrt}(\theta^{\mathrm{r}}, \mathbf{p}^{\mathrm{D}})$

$$\mathit{For\ all}\ \mathbf{S} \in \mathit{dom}(\theta^{\mathrm{r}}),\ \mathit{either}\ \theta^{\mathrm{r}}(\mathbf{S}) = 1\ \mathit{or}\ \theta^{\mathrm{r}}(\mathbf{S}) = \perp\ \mathit{and}\ \mathbf{S} \notin \mathit{Can}^{\mathrm{S}}_{\varrho}((\varrho\ \langle \theta^{\mathrm{r}}, \mathrm{GO}\rangle.\ \mathbf{p}^{\mathrm{D}}), \{\}).$$

This completeness means that every signal in the environment has a definite value. From there the value of the signals is extracted using the metafunction *restrict*, which gives back a map like $\theta^{\mathrm{r}}{}_2$, but with every signal that can be set to 0 set to 0, and with the domain restricted to $\mathbf{O}$:

Definition: $\mathit{restrict}(\theta, \mathbf{O}, \mathbf{p})$

read as: Restrict θ to signals in $\mathbf{O}$, given their values as determined by the program $\mathbf{p}$.

$$\mathit{restrict}(\theta, \mathbf{O}, \mathbf{p})(\mathbf{S}) = \begin{cases} 0 & \text{where } \mathbf{S} \in \mathbf{O},\ \theta(\mathbf{S}) = \perp, \text{ and } \mathbf{S} \notin \mathit{Can}^{\mathrm{S}}_{\varrho}(\mathbf{p}, \{\}) \\ \theta(\mathbf{S}) & \text{where } \mathbf{S} \in \mathbf{O} \end{cases}$$

The second clause of $\mathit{eval}/^{\mathrm{E}}$ recognizes programs which are non-constructive. This is accomplished with a special judgment, $\theta^{\mathrm{r}}; \mathbf{A}; \mathbf{E} \vdash_{\mathrm{B}} \mathbf{p}$, which can be read as "In the program context consisting of the state θ^{r}, the control variable $\mathbf{A}$ and the evaluation context $\mathbf{E}$ the program $\mathbf{p}$ is blocked on some signal or shared variable". In this case the program is non-constructive, and its signal statuses are given by the same *restrict* metafunction. The resulting signal statuses may, however, contain $\perp$ in this case.

$$\frac{\theta^{r}(\mathbf{S}) = \bot \qquad \mathbf{S} \in \mathit{Can}^{S}_{Q}(\,(Q\ \langle \theta^{r}, \mathbf{A}\rangle.\ \mathbf{E}[(\text{if } \mathbf{S}\ \mathbf{p}\ \mathbf{q})]),\ \{\}\,)}{\theta^{r};\ \mathbf{A};\ \mathbf{E} \vdash_{B} (\text{if } \mathbf{S}\ \mathbf{p}\ \mathbf{q})}\ [\text{if}] \qquad \frac{}{\theta^{r};\ \text{WAIT};\ \mathbf{E} \vdash_{B} (\text{emit } \mathbf{S})}\ [\text{emit-wait}]$$

$$\frac{\theta^{r};\ \mathbf{A};\ \mathbf{E}[(\text{suspend} \bigcirc \mathbf{S})] \vdash_{B} \mathbf{p}}{\theta^{r};\ \mathbf{A};\ \mathbf{E} \vdash_{B} (\text{suspend}\ \mathbf{p}\ \mathbf{S})}\ [\text{suspend}] \qquad \frac{\theta^{r};\ \mathbf{A};\ \mathbf{E}[(\text{trap} \bigcirc)] \vdash_{B} \mathbf{p}}{\theta^{r};\ \mathbf{A};\ \mathbf{E} \vdash_{B} (\text{trap}\ \mathbf{p})}\ [\text{trap}]$$

$$\frac{\theta^{r};\ \mathbf{A};\ \mathbf{E}[(\text{seq} \bigcirc \mathbf{q})] \vdash_{B} \mathbf{p}}{\theta^{r};\ \mathbf{A};\ \mathbf{E} \vdash_{B} (\text{seq}\ \mathbf{p}\ \mathbf{q})}\ [\text{seq}]$$

$$\frac{\theta^{r};\ \mathbf{A};\ \mathbf{E}[(\text{par} \bigcirc \mathbf{p}^{D})] \vdash_{B} \mathbf{p}}{\theta^{r};\ \mathbf{A};\ \mathbf{E} \vdash_{B} (\text{par}\ \mathbf{p}\ \mathbf{p}^{D})}\ [\text{parl}] \qquad \frac{\theta^{r};\ \mathbf{A};\ \mathbf{E}[(\text{par}\ \mathbf{p}^{D} \bigcirc)] \vdash_{B} \mathbf{q}}{\theta^{r};\ \mathbf{A};\ \mathbf{E} \vdash_{B} (\text{par}\ \mathbf{p}^{D}\ \mathbf{q})}\ [\text{parr}]$$

$$\frac{\theta^{r};\ \mathbf{A};\ \mathbf{E}[(\text{par} \bigcirc \mathbf{q})] \vdash_{B} \mathbf{p} \qquad \theta^{r};\ \mathbf{A};\ \mathbf{E}[(\text{par}\ \mathbf{p} \bigcirc)] \vdash_{B} \mathbf{q}}{\theta^{r};\ \mathbf{A};\ \mathbf{E} \vdash_{B} (\text{par}\ \mathbf{p}\ \mathbf{q})}\ [\text{par-both}]$$

Figure 18: The blocked judgment on pure terms

3.2.1. The blocked judgment

The $\theta^{r};\ \mathbf{A};\ \mathbf{E} \vdash_{B} \mathbf{p}$ judgment traverses the program and checks that at the leaf of each evaluation context there is either an if which is blocked on an signal or an emit which is awaiting a GO.

The relation is in figure 18. The first case, **[if]**, checks that, for a conditional, the status of its signal is $\bot$, and that the signal is not in the result of Can^{S}_{Q} for the whole program. These conditions mean that the **[is-absent]** and **[is-present]** rules cannot fire. The second rule **[emit-wait]** says that the program is blocked on an emit if the control variable is telling us to WAIT. Note that both of these clauses assume that $\mathbf{S}$ is in θ^{r}. We will return to this in section 3.2.2.

The remainder of the judgment recurs through the term following the grammar of evaluation contexts, copying each layer of the context into the evaluation context on the left of the judgment, so that the overall program can be reconstructed in the base cases.

The interesting part here is the handling of par which requires three clauses. Together these clauses ensure that at least one branch of the par is blocked, and that the other branch is either blocked, or is done evaluating.

$$\frac{}{\vdash_{\mathrm{CB}} \text{nothing}}\,[\text{nothing}] \quad \frac{}{\vdash_{\mathrm{CB}} \text{pause}}\,[\text{pause}] \quad \frac{}{\vdash_{\mathrm{CB}} (\text{emit } \mathbf{S})}\,[\text{emit}]$$

$$\frac{}{\vdash_{\mathrm{CB}} (\text{exit } \mathbf{n})}\,[\text{exit}] \quad \frac{\vdash_{\mathrm{CB}} \mathbf{p}}{\vdash_{\mathrm{CB}} (\text{trap } \mathbf{p})}\,[\text{trap}]$$

$$\frac{\vdash_{\mathrm{CB}} \mathbf{p}}{\vdash_{\mathrm{CB}} (\text{signal } \mathbf{S}\ \mathbf{p})}\,[\text{signal}] \quad \frac{\vdash_{\mathrm{CB}} \mathbf{p} \qquad \vdash_{\mathrm{CB}} \mathbf{q}}{\vdash_{\mathrm{CB}} (\text{if } \mathbf{S}\ \mathbf{p}\ \mathbf{q})}\,[\text{if}]$$

$$\frac{\mathcal{BV}(\mathbf{p}) \cap \mathcal{FV}(\mathbf{q}) = \varnothing \qquad \vdash_{\mathrm{CB}} \mathbf{p} \qquad \vdash_{\mathrm{CB}} \mathbf{q}}{\vdash_{\mathrm{CB}} (\text{seq } \mathbf{p}\ \mathbf{q})}\,[\text{seq}] \quad \frac{\{\mathbf{S}\} \cap \mathcal{BV}(\mathbf{p}) = \varnothing \qquad \vdash_{\mathrm{CB}} \mathbf{p}}{\vdash_{\mathrm{CB}} (\text{suspend } \mathbf{p}\ \mathbf{S})}\,[\text{suspend}] \quad \frac{\vdash_{\mathrm{CB}} \mathbf{p}}{\vdash_{\mathrm{CB}} (\varrho\,\langle\theta^{\mathrm{r}}, \mathbf{A}\rangle.\ \mathbf{p})}\,[\rho]$$

$$\frac{\mathcal{BV}(\mathbf{p}) \cap \mathcal{BV}(\mathbf{q}) = \varnothing \qquad \mathcal{FV}(\mathbf{p}) \cap \mathcal{BV}(\mathbf{q}) = \varnothing \qquad \mathcal{BV}(\mathbf{p}) \cap \mathcal{FV}(\mathbf{q}) = \varnothing}{\vdash_{\mathrm{CB}} (\text{par } \mathbf{p}\ \mathbf{q})}\,[\text{par}]$$

Figure 19: The correct binding judgment

3.2.2. Open programs

There are two major cases that make $\mathit{eval}^{\mathrm{E}}$ is a partial function. One of these involves loops, and we will return to this in section 5.1.4. The other is the case of open programs. If a program has a free variable it may reach a state where it is not $\vdash_{\mathrm{B}}$ or $\mathbf{p}^{\mathrm{D}}$, but it cannot progress. For example, the program $(\varrho\,\langle\{\}, \mathrm{GO}\rangle.\ (\text{emit } \mathbf{S}))$ can never be equal to a term which is $\vdash_{\mathrm{B}}$, because the $\vdash_{\mathrm{B}}$ judgment will see that the control variable is not WAIT, and will therefore determine that emits can be run. On the other hand $(\text{emit } \mathbf{S})$ is not in the grammar of $\mathbf{p}^{\mathrm{D}}$, because emits are terms which can execute. Therefore this particular program is stuck. Therefore $\mathit{eval}^{\mathrm{E}}$ is not defined on such terms.

3.3. Correct Binding & Schizophrenia

A natural question about the calculus for someone familiar with the lambda calculus might be "is there an $[\alpha]$ rule?". Instead of using the variable convention (Barendregt 1984) and working up to α-equivalence, as is common in the lambda calculus world, I take a different approach inspired by Esterel, circuits, and schizophrenia and work up to what I have called correct binding. The judgment for a program with correct binding is given in figure 19.

This judgment captures the class of programs that cannot accidentally capture a variable when it takes a reduction step. To understand this look at the rule for seq in $\vdash_{\text{CB}}$. It states that the bound variables of $\mathbf{p}$ must not overlap with the free variables of $\mathbf{q}$. This means that if an environment is lifted out of $\mathbf{p}$, it cannot capture any variables in $\mathbf{q}$.

This approach explains all the rules of $\vdash_{\text{CB}}$ in fact. For any term, if that term could be part of an evaluation context or could reduce to a term which could be part of an evaluation context, then the term that would be in the hole of that context must have distinct bound variables from any possible adjacent free variables.

LEMMA 1 (CORRECT BINDING IS PRESERVED)**.**

For all $\mathbf{p}$, $\mathbf{q}$, *if* $\mathbf{p} \rightharpoonup^{\text{E}} \mathbf{q}$ *and* $\vdash_{\text{CB}} \mathbf{p}$ *them* $\vdash_{\text{CB}} \mathbf{q}$

The full proof is given in lemma 53 (CORRECT BINDING IS PRESERVED). This follows by case analysis over the rules of $\rightharpoonup^{\text{E}}$. Note that any program can be renamed into a program with correct binding by making all variable names unique. Therefore, I assume that any program used in the calculus or in my proofs has correct binding.

Using correct binding instead of α-equivalence also explains the lack of a $[\mathbf{gc}]$ rule, as appears in the state calculus (Felleisen and Hieb 1992). As the calculus does not rename variables, but instead constantly replaces instances of variables in the environment with the new instances, there is a maximum size to every environment. This matches with actual Esterel implementations which, absent host language allocation, use a bounded amount of memory. In addition correct binding explains how the calculus avoids schizophrenic variables, which is discussed in section 5.1.3.

3.4. Using the calculus, by example

The calculus is designed to prove equivalences between program fragments because any two expressions that are $\equiv^{\text{E}}$ are contextually equivalent, which is proved in section 4.2. This section is designed to give some examples of what can and cannot be proved in the calculus, to give some sense of its limits. The proofs for the equalities in this section are given in appendix D. The first example is that adjacent signals can be swapped:

THEOREM 2 (CAN SWAP ADJACENT SIGNALS)**.**

For all $\mathbf{p}$, $\mathbf{S}_1$, $\mathbf{S}_2$, $(\text{signal } \mathbf{S}_1\ (\text{signal } \mathbf{S}_2\ \mathbf{p})) \simeq^{E} (\text{signal } \mathbf{S}_2\ (\text{signal } \mathbf{S}_1\ \mathbf{p}))$

The full proof is given in theorem 84 (CAN SWAP ADJACENT SIGNALS). This proof mainly relies on the $[\mathbf{merge}]$ and $[\mathbf{signal}]$ axioms of $\rightharpoonup^{\mathrm{E}}$, as well as the transitivity and symmetry of the equality relation.

The second proof shows that we can take the else branch of an if when the signal cannot be emitted:

THEOREM 3 (CAN TAKE THE ELSE BRANCH FOR ADJACENT SIGNALS)**.**

For all $\mathbf{S}$, $\mathbf{p}$, $\mathbf{q}$, *If* $\mathbf{S} \notin \mathit{Can}^{S}(\mathbf{p}, \{\mathbf{S} \mapsto \bot\})$ *and,* $\mathbf{S} \notin \mathit{Can}^{S}(\mathbf{q}, \{\mathbf{S} \mapsto \bot\})$, *then*
$(\text{signal } \mathbf{S}\ (\text{if } \mathbf{S}\ \mathbf{p}\ \mathbf{q})) \simeq^{E} (\text{signal } \mathbf{S}\ \mathbf{q})$

The full proof is given in theorem 85 (CAN TAKE THE ELSE BRANCH FOR ADJACENT SIGNALS).

The next proof demonstrates some of the weaknesses of the calculus. Specifically, in order to lift a signal out of an evaluation context an out environment is needed:

THEOREM 4 (LIFTING SIGNALS)**.**

For all $\mathbf{S}$, $\mathbf{p}$, $\mathbf{E}$, $\mathbf{A}$, $(\varrho\ \langle\{\}, \mathbf{A}\rangle.\ \mathbf{E}[(\text{signal } \mathbf{S}\ \mathbf{p})]) \simeq^{E} (\varrho\ \langle\{\}, \mathbf{A}\rangle.\ (\text{signal } \mathbf{S}\ \mathbf{E}[\mathbf{p}]))$

The full proof is given in theorem 86 (LIFTING SIGNALS). The environment is needed because the only rule that allows for moving environments around is the $[\mathbf{merge}]$ rule, which requires two environments. This could be fixed by adding the axiom $\mathbf{p} \equiv^{\mathrm{E}} (\varrho\ \langle\{\}, \mathrm{WAIT}\rangle.\ \mathbf{p})$. Such an axiom should be sound because, as is shown in section 4.1.1, the compilation of $\mathbf{p}$ and $(\varrho\ \langle\{\}, \mathrm{WAIT}\rangle.\ \mathbf{p})$ should be identical.

Next, we have another weakness in the calculus. The next theorem holds, but cannot be proven in the calculus:

CONJECTURE 5 (LIFT SIGNAL EMISSION (NOT PROVABLE))**.**

For all $\mathbf{E}$, $\mathbf{S}$, $\mathbf{E}[(\text{emit } \mathbf{S})] \simeq^{E} (\text{par } (\text{emit } \mathbf{S}) \mathbf{E}[\text{nothing}])$

In fact it cannot prove even this weaker statement:

CONJECTURE 6 (LIFT SIGNAL EMISSION, WITH BINDER (NOT PROVABLE))**.**

For all $\mathbf{E}$, $\mathbf{S}$, $(\text{signal } \mathbf{S} \mathbf{E}[(\text{emit } \mathbf{S})]) \simeq^{E} (\text{signal } \mathbf{S} (\text{par } (\text{emit } \mathbf{S}) \mathbf{E}[\text{nothing}]))$

This is because in order to lift up an emit we must run the emit, putting a 1 in an environment, then using $[\mathbf{sym}]$ run the emit backwards to place it elsewhere. However this can only be done if the environment has a GO, which the calculus cannot insert. Possible solutions to this, to strengthen the calculus, are discussed in section 7.2.

CHAPTER 4

Proving the calculus correct

The three properties for the calculus which require proof are consistency, soundness, and adequacy. This section provides an overview of those proofs. This section relies heavily on the background given in section 2.3, as well as the descriptions of the properties given in chapter 1.

4.1. Setup for the proofs

The purpose of this section is to give the setup needed to understand the statements of the theorems and their proofs. To start with, the proofs in this section are only for the pure loop free portion of Esterel. However some other proofs in this document are defined on the full kernel. To distinguish these I use the superscript p to denote pure, loop free terms (e.g $\mathbf{p}^{\mathsf{p}}$, $\mathbf{q}^{\mathsf{p}}$). Similarly contexts over pure, loop free programs are labeled with the same superscript (e.g. $\mathbf{E}^{\mathsf{p}}$). These pure terms also may only contain the control variable WAIT. In some cases I will need to discuss terms which may have the control variable set to GO. I will write these terms as $\mathbf{p}^{\mathsf{p}}_{\mathsf{GO}}$. Figure 20 gives the grammars for these terms.

4.1.1. The compiler

The proofs of soundness and adequacy are proved with respect the circuit semantics of Esterel. This semantics is, in general, the ground truth semantics and guides the actual implementation of an Esterel compiler. The core of this semantics is the compilation function $[\![\cdot]\!]$. This function translates Pure Esterel programs into circuits of the shape given in figure 21. The circuit compiler I describe here is the same as the one given in Berry (2002), except for two changes in the compilation of par. These changes are necessary for soundness, and were derived from the Esterel v7 compiler. I will describe them more later.

$$
\begin{array}{lll}
\mathbf{E}^{\mathrm{p}} ::= (\mathrm{seq}\ \mathbf{E}^{\mathrm{p}}\ \mathbf{q}^{\mathrm{p}}) & \mathbf{p}^{\mathrm{p}}, \mathbf{q}^{\mathrm{p}}, \mathbf{r}^{\mathrm{p}} ::= \mathrm{nothing} & \mathbf{p}^{\mathrm{p}}_{\mathrm{GO}} ::= \mathrm{nothing} \\
\quad | (\mathrm{par}\ \mathbf{E}^{\mathrm{p}}\ \mathbf{q}^{\mathrm{p}}) & \quad | \mathrm{pause} & \quad | \mathrm{pause} \\
\quad | (\mathrm{par}\ \mathbf{p}^{\mathrm{p}}\ \mathbf{E}^{\mathrm{p}}) & \quad | (\mathrm{seq}\ \mathbf{p}^{\mathrm{p}}\ \mathbf{p}^{\mathrm{p}}) & \quad | (\mathrm{seq}\ \mathbf{p}^{\mathrm{p}}_{\mathrm{GO}}\ \mathbf{p}^{\mathrm{p}}_{\mathrm{GO}}) \\
\quad | (\mathrm{suspend}\ \mathbf{E}^{\mathrm{p}}\ \mathbf{S}) & \quad | (\mathrm{par}\ \mathbf{p}^{\mathrm{p}}\ \mathbf{p}^{\mathrm{p}}) & \quad | (\mathrm{par}\ \mathbf{p}^{\mathrm{p}}_{\mathrm{GO}}\ \mathbf{p}^{\mathrm{p}}_{\mathrm{GO}}) \\
\quad | (\mathrm{trap}\ \mathbf{E}^{\mathrm{p}}) & \quad | (\mathrm{trap}\ \mathbf{p}^{\mathrm{p}}) & \quad | (\mathrm{trap}\ \mathbf{p}^{\mathrm{p}}_{\mathrm{GO}}) \\
\quad | \bigcirc & \quad | (\mathrm{exit}\ \mathbf{n}) & \quad | (\mathrm{exit}\ \mathbf{n}) \\
 & \quad | (\mathrm{signal}\ \mathbf{S}\ \mathbf{p}^{\mathrm{p}}) & \quad | (\mathrm{signal}\ \mathbf{S}\ \mathbf{p}^{\mathrm{p}}_{\mathrm{GO}}) \\
 & \quad | (\mathrm{suspend}\ \mathbf{p}^{\mathrm{p}}\ \mathbf{S}) & \quad | (\mathrm{suspend}\ \mathbf{p}^{\mathrm{p}}_{\mathrm{GO}}\ \mathbf{S}) \\
 & \quad | (\mathrm{if}\ \mathbf{S}\ \mathbf{p}^{\mathrm{p}}\ \mathbf{p}^{\mathrm{p}}) & \quad | (\mathrm{if}\ \mathbf{S}\ \mathbf{p}^{\mathrm{p}}_{\mathrm{GO}}\ \mathbf{p}^{\mathrm{p}}_{\mathrm{GO}}) \\
 & \quad | (\mathrm{emit}\ \mathbf{S}) & \quad | (\mathrm{emit}\ \mathbf{S}) \\
 & \quad | (\varrho\ \langle \theta^{\mathrm{r}}, \mathrm{WAIT} \rangle.\ \mathbf{p}^{\mathrm{p}}) & \quad | (\varrho\ \langle \theta^{\mathrm{r}}, \mathbf{A} \rangle.\ \mathbf{p}^{\mathrm{p}}_{\mathrm{GO}})
\end{array}
$$

Figure 20: Flavors of loop-free, pure terms

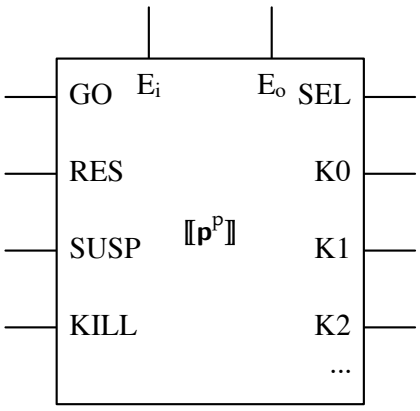


Figure 21: The shape of circuits returned by $[\![\mathbf{p}^{\mathrm{p}}]\!]$

The circuit compilation function, in essence, expresses the causality graphs described in section 2.1.3 as circuits. The circuits are more complex, as they handle more of Esterel than the causality diagrams do, but at their core they have the same execution model. The four input wires on the left of the diagram in figure 21 (GO, RES, SUSP, KILL) are control wires which guide the execution of the circuit. The GO wire is true when the circuit is supposed to start for the first time—it corresponds to an coming control edge that connects to the start of the program in the causality graph model. The RES wire is true when the circuit may resume execution in a previous instant (when, say, it has a register containing an 1)—it corresponds to an control edge that connects start node to a pause. The SUSP wire is used by the compilation of suspend to suspend a term. The KILL wire is used by trap and exit to abort the execution of a circuit. Neither SUSP nor KILL are expressed in the causality graph model.

The two wires on the top $\mathbf{E}_i$ and $\mathbf{E}_o$ represent bundles of wires that are input and output signals of the term. Any free signal which is tested by an if will have a corresponding wire in $\mathbf{E}_i$. Any free signal which occurs in an (emit **S**) will have a corresponding wire in $\mathbf{E}_o$.

The bottom output wires on the right (K0 et al.) encode the return codes. The wire K0 is 1 when the term completes, K1 is 1 when the term would pause, K2 is 1 when the term would exit to the first trap, etc. Only one of the Kn wires may be 1 at a given time. In circuit speak, the Kn wires are a one-hot encoding of the set of return codes from Can^{K}.

The final output wire SEL is 1 if there is any register in the circuit which holds a 1. Such a circuit is said to be selected. Registers are used to encode whether or not the program paused with the term. That is, each pause will generate a register, and that register will have an 1 when the term should resume from that pause.

A quick note about these circuits: their activation is completely controlled by GO, RES, and SEL: if GO and either RES or SEL are 0, then all of the output signals and return codes will be 0 and the circuit will be constructive. This is proven formally in lemma 76 (Activation Condition), and follows fairly easily by induction. In addition the compilation function assumes that GO and SEL are mutually exclusive: a selected term may not be started for the first time. This assumption, however, can be removed with a small change, which is discussed about in section 7.1.

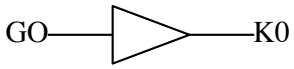


Figure 22: The compilation of ⟦nothing⟧

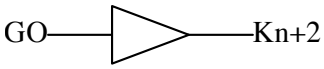


Figure 23: The compilation of ⟦(exit **n**)⟧

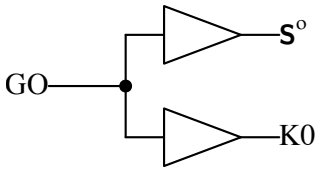


Figure 24: The compilation of ⟦(emit **S**)⟧

The simplest clause of the compiler is ⟦nothing⟧, shown in figure 22. Its compilation connects the GO wire directly to K0, as when nothing is reached it immediately terminates. Remember that any wire not draw in the diagram is taken to be 0, therefore this term can never be selected, and can never have a different exit code.

The next simplest compilation clause is exit, which just GO to corresponding return code wire for that exit code.

Next, we have the compilation of emit, found in figure 24. Like nothing, this connects GO to K0 as this term terminates immediately. It also adds the wire $\mathbf{S}^{o}$ to the output environment, as this signal will be emitted immediately when the term executes. Note that I will always name the output wires for a signal **S** as $\mathbf{S}^{o}$, and the input wires $\mathbf{S}^{i}$.

The last term without subterms, pause, is also more complex than the others. Its compilation is in figure 25. Firstly, the GO wire is connected to the K1 wire, as a pause will pause the first time is reached. The SEL wire is similarly straightforward: it is true when the register is true. The K0 wire just says that a pause finishes when it is selected and

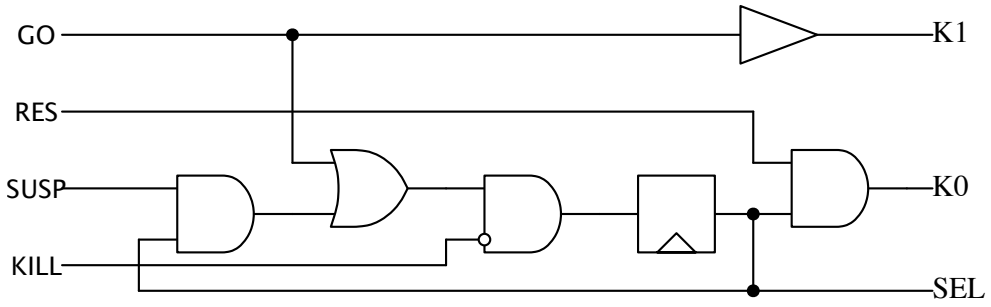


Figure 25: The compilation of $[\![\mathsf{pause}]\!]$

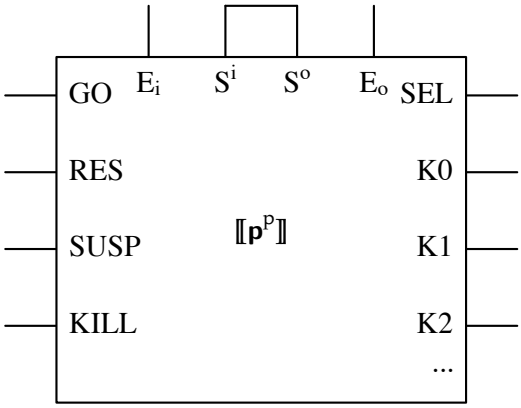


Figure 26: The compilation of $[\![(\mathsf{signal}\ \mathbf{S}\ \mathbf{p}^p)]\!]$

resumed. The complex part goes into determining if the term will be selected in the next instant. The register will get a 1 if it is not killed, and if either it is reached for the first time (GO) or it was already selected and it is being resumed, in which case it's selection status needs to be maintained.

The compilation of signal (figure 26) is fairly simple: the inner term is compiled, and the wires for the given signal are connected to each other, and removed from the input and output signal sets.

The compilation of if (figure 27) compiles both terms, and broadcasts all inputs except for GO to both subcircuits. All outputs are or'ed. The GO wire of both subcircuits is given by the overall GO and value of the conditioned signal.

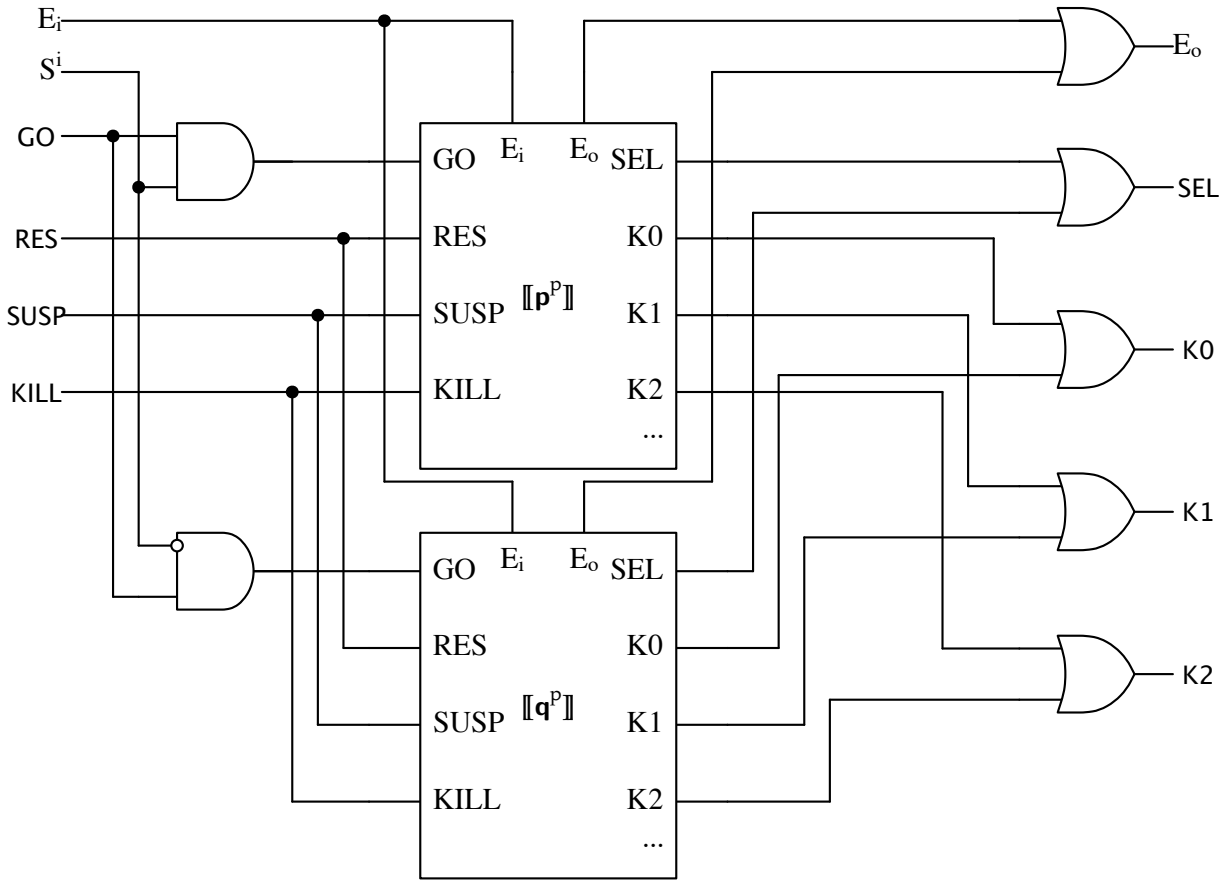


Figure 27: The compilation of ⟦(if **S** **p**^{p} **q**^{p})⟧

The ⟦**p**^{p}⟧ subcircuit activates if and only if both GO and **S**^{i} are 1. The ⟦**p**^{p}⟧ subcircuit activates if and only if GO is 1 and **S**^{i} is 0. That is a branch is activated if and only if the if is activated and the signal is in the corresponding state.

The compilation of suspend (figure 28) does nothing special to GO: remember suspended terms behave normally on the first instant they are reached. However the compilation intercepts the RES wire, and only resumes the subcircuit if the suspension signal **S** is 0. If the signal is 1 then the circuit is resumed instead, and this information is passed to the K1 wire. All of this only occurs, however, if the subcircuit is selected. If it is not, the RES and SUSP wires are suppressed.

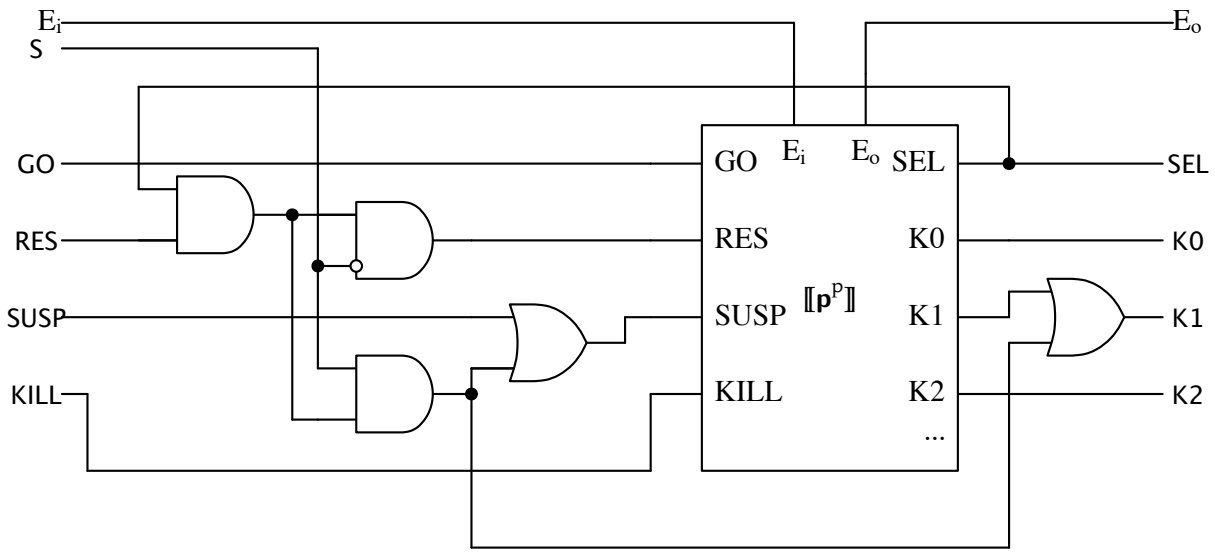


Figure 28: The compilation of $[\![(\text{suspend } \mathbf{p}^{\mathrm{p}}\ \mathbf{S})]\!]$

The compilation of seq (figure 29) wires the K0 wire of the first subcircuit to the GO wire of the second, causing the second subcircuit to start when the first finishes. The overall K0 wire is thus just the K0 wire of the second subcircuit, as the seq only completes when it does. The remainder of the outputs are or'ed, and the remainder of the inputs are broadcast to the subciruits.

The compilation of trap (figure 30) intercepts the K2 wire (which represents the abortion of this term) and passes it back to the KILL wire of the subcircuit, killing it when this trap catches its corresponding exit. It then shifts the return codes in the same way as $\downarrow^{\kappa}$.

The par circuit (figure 31) circuit is the most complex of the compilation clauses, and has two changes from the compiler given in Berry (2002). To start with, the inputs part: The GO, RES, SUSP and $\mathbf{E}_i$ wires are broadcast to the subcircuits. The SEL and $\mathbf{E}_o$ wires are the or'ed from the subcircuits. The complex part is in handling the return codes and the KILL wire.

To start with, the return codes are joined together by a synchronizer, which is given in figure 32. The synchronizer implements the max operation, as is used in *Can*. That is, the return code for the overall circuit is the max of the

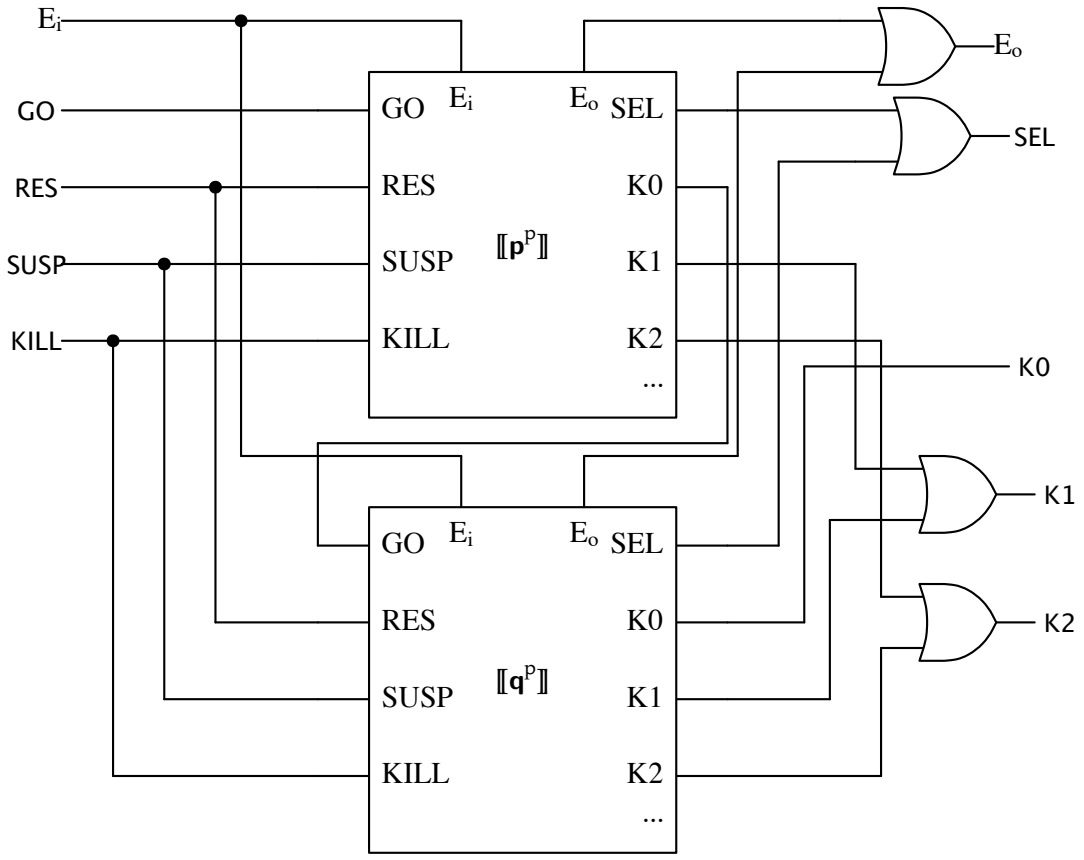


Figure 29: The compilation of $[\![(\mathsf{seq}\ \mathbf{p}^{\mathrm{p}}\ \mathbf{q}^{\mathrm{p}})]\!]$

return code of the subcircuits (the Ln and Rn wires). However this is complicated by multi-instant execution: special behavior is needed if one branch finished in a previous instant. In this case the return code of the live branch must be used. This is handled by the LEM and REM wires, which encode if the other branch is the only live branch. Which is to say, LEM is true if and only if the circuit is resuming, the $\mathbf{p}^{\mathrm{p}}$ branch is dead, and the $\mathbf{q}^{\mathrm{p}}$ branch is selected. The reverse holds for REM. There LEM and REM wires make it look like the dead branch has exit code 0, which, as the lowest return code, causes the synchronizer to output the other branches return code.

The last part of the synchronizer is the handling of KILL. Compilationing par must account for the scenario where one branch has an exit code greater than 1, which must abort the other branch. Normally this would be handled by the compilation of trap, but in this case we would loose local reasoning if we did that, as we do not know if the program

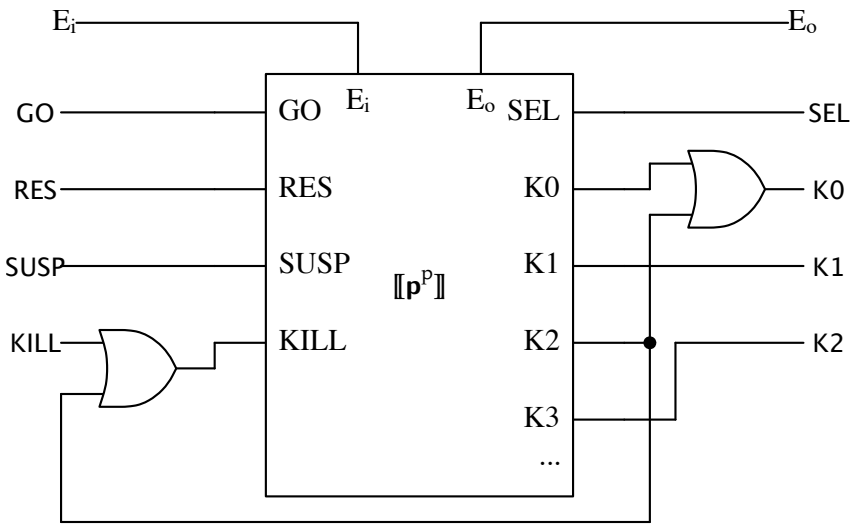


Figure 30: The compilation of $[\![(\text{trap } \mathbf{p}^{\text{p}})]\!]$

is in fact, closed, by a trap. Therefore both subcircuits are killed if the outer KILL wire is 1, or if the over all return code is 2 or greater.

The two changes to this from the compiler in Berry (2002) are the KILL wire including the return codes, and the definition of the LEM and REM wires. These changes are to handle a violation of soundness, and so are discussed in detail section 4.2.2.

The compilation of ϱ follows along similar lines to the compilation of signal: we take one signal at a time out of the environment and connect its input wire to it's output wire (figure 33), with one exception. The wire connection goes through $[\![\mathbf{status}^{\text{r}}]\!]$ which connects the two wires if $\mathbf{status}^{\text{r}} = \bot$. However if $\mathbf{status}^{\text{r}} = 1$ then the connection is cut, and the input wire is defined to be 1. This is shown in figure 34.

Once all signals have been compiled, the **A** part is compiled in a similar manner (figure 35). If the **A** is WAIT, the GO wire is taken from the environment. If **A** is GO, then the GO wire will be defined to be 1. This is shown in figure 36.

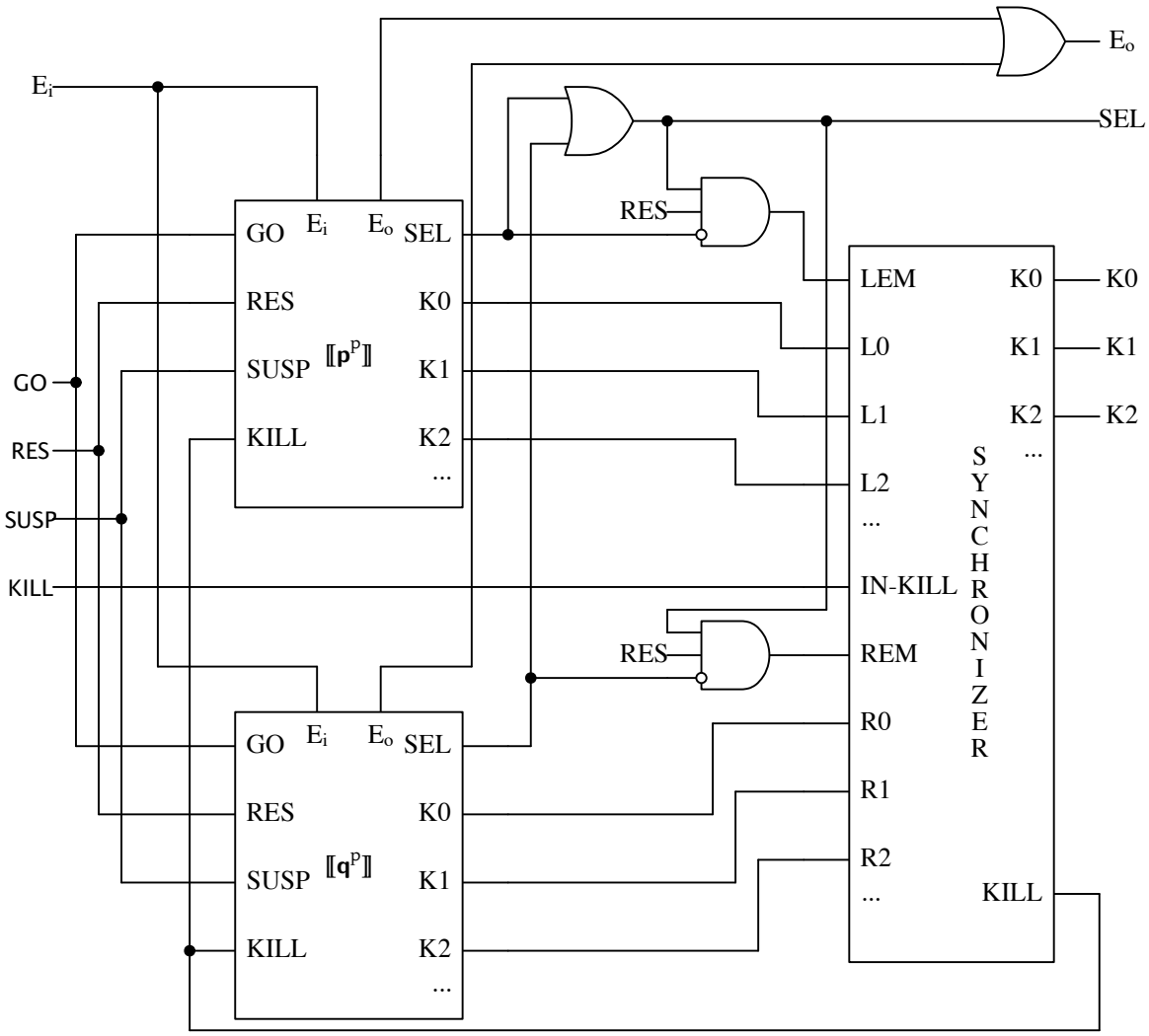


Figure 31: The compilation of $[\![(\mathsf{par}\ \mathbf{p}^{\mathsf{p}}\ \mathbf{q}^{\mathsf{p}})]\!]$

4.1.2. The Circuit Solver, Circuitous

To reason about circuits in a more automated fashion, I have implemented a symbolic reasoning engine—a solver—for concrete circuits. The solver I use is an implementation of the algorithm for executing constructive circuits given by Malik (1994) (and extended by Shiple et al. (1996) to handle registers) in the language Rosette (Torlak and Bodik 2013).

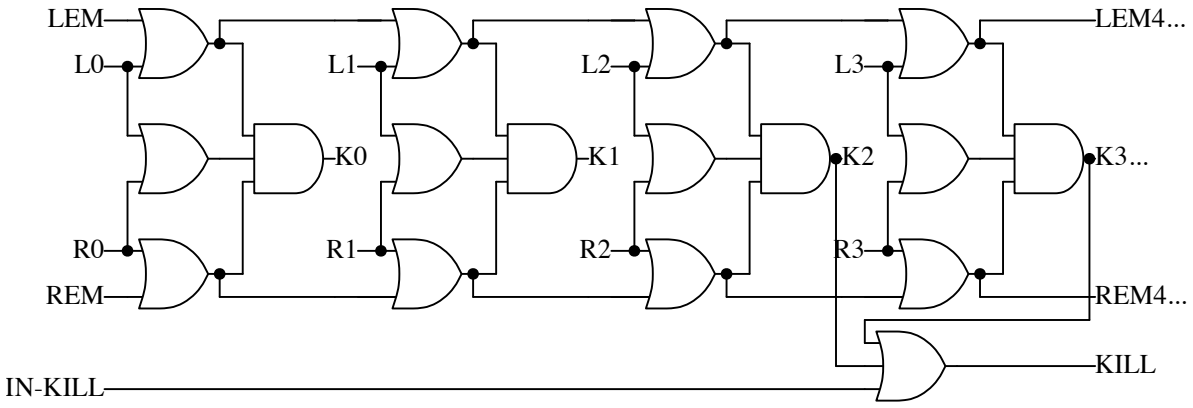


Figure 32: The parallel synchronizer

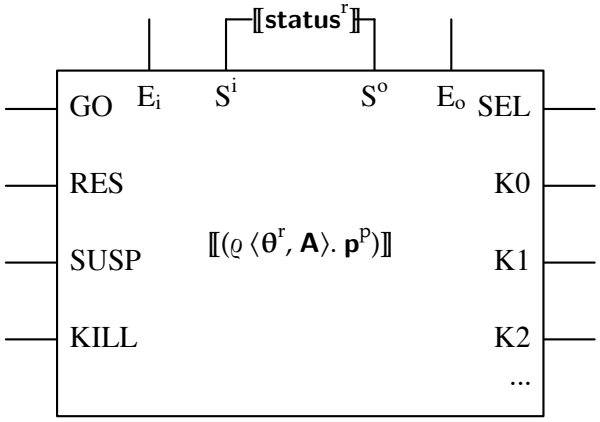


Figure 33: The compilation of $[\![(\varrho\,\langle\theta^{\mathsf{r}} \leftarrow \{\,\mathbf{S} \mapsto \mathbf{status}^{\mathsf{r}}\,\}, \mathbf{A}\rangle.\ \mathbf{p}^{\mathsf{p}})]\!]$

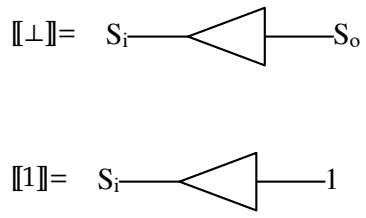


Figure 34: Compiling statuses

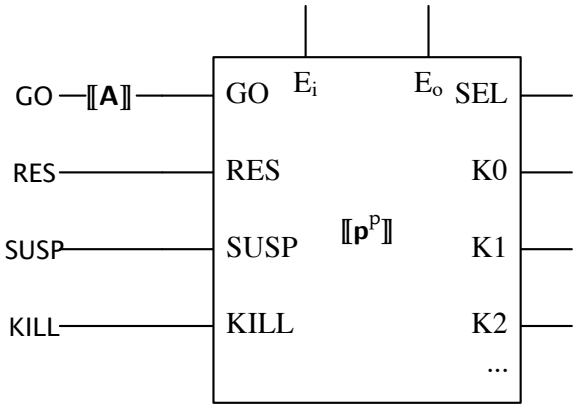


Figure 35: The compilation of $[\![(\varrho\ \langle\{\}, \mathbf{A}\rangle.\ \mathbf{p}^{\mathrm{P}})]\!]$

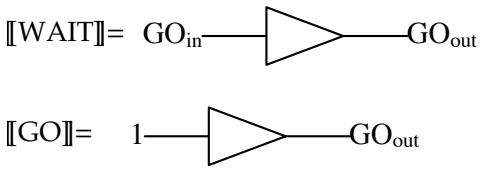


Figure 36: Compiling control variables

Rosette is an domain specific language embedded within Racket (Flatt and PLT 2010), which is designed for defining other domain specific languages so that the programs written in those language can be reasoned about using an SMT solver. Specifically Rosette allows for symbolic execution of programs such that the result of a program is not a value, but a symbolic expression which represents the value. This symbolic value may then be turned into a logic formal that can be given to an SMT solver.

Circuitous is capable of evaluating a given circuit on some inputs, verifying if two circuits are contextually equivalent, and verifying if a circuit is constructive for set of inputs which do not contain $\bot$. This solver is combined with a mechanized version of the compiler presented in section 4.1.1, which is in the codebase for this dissertation. Using these, the base cases of many of the proofs in this section simply invoke the circuit solver to complete the proof.

The source for this solver may be found at https://github.com/florence/circuitous/, and the core of the solver is listed in appendix C.

4.1.3. On Instants

The proofs in this section only look at a single instant of execution. This is accomplished by each proof having the assumption that the SEL wire is 0, thus forcing evaluation to occur in the first instant only. The calculus will be extended to multiple instants in section 5.3.

4.1.4. Agda Codebase

Some proofs I reference are not given in this document. Instead they are given in a separate Agda code base. which was attempt to prove the correctness of a previous version of the calculus (Florence et al. 2019). While the calculus has since changed, *Can* has not. Therefore I re-use some of the proofs from that work which relate to *Can*. This Agda codebase is located in the repository for this dissertation.

4.1.5. Notation

At times my theorem statements will say something to the effect of *For all* $\mathbf{p}^{\mathsf{p}}$. This is a pun which is meant to read as *For all terms* $\mathbf{p}^{\mathsf{p}}$ *drawn from the set of terms* $\mathbf{p}^{\mathsf{p}}$. Similarly I will sometimes say *For all* $\mathbf{r}^{\mathsf{p}} = \mathbf{E}^{\mathsf{p}}[\mathbf{p}^{\mathsf{p}}]$. This is shorthand for *For all* $\mathbf{r}^{\mathsf{p}}$, $\mathbf{E}^{\mathsf{p}}$, *and* $\mathbf{p}^{\mathsf{p}}$ *such that* $\mathbf{r}^{\mathsf{p}} = \mathbf{E}^{\mathsf{p}}[\mathbf{p}^{\mathsf{p}}]$.

4.2. Justifying Soundness

THEOREM 7 (SOUNDNESS)**.**

For all $\mathbf{p}^{\mathsf{p}}$ *and* $\mathbf{q}^{\mathsf{p}}$, *if* $\vdash_{\mathrm{CB}} \mathbf{p}^{\mathsf{p}}$, $\mathbf{p}^{\mathsf{p}} \equiv^{\mathrm{E}} \mathbf{q}^{\mathsf{p}}$, $[\![\mathbf{p}^{\mathsf{p}}]\!](\mathrm{SEL}) \simeq 0$, *and* $[\![\mathbf{q}^{\mathsf{p}}]\!](\mathrm{SEL}) \simeq 0$ *then* $[\![\mathbf{p}^{\mathsf{p}}]\!] \simeq^{C} [\![\mathbf{q}^{\mathsf{p}}]\!]$

The proof is given in the appendix as theorem 29 (Soundness). To pick the statement apart, it says that if two terms are $\equiv^{E}$, and those terms have correct binding, then, when we restrict ourselves to looking at a single instant, the compilation of those circuits is $\simeq^{C}$.

This proof proceeds by induction over the structure of the equality relation $\equiv^{E}$. Thus, the majority of the work in the proof goes into showing that it holds for each rule of $\rightharpoonup^{E}$. Each rule in $\rightharpoonup^{E}$ is proved sound, in general, by induction on $\mathbf{p}^{P}$. The base cases have concrete circuits, so in general the base cases are proven by the circuit solver.

4.2.1. Important lemmas

This section will discuss the proof sketches of the most interesting or informative lemmas needed to prove soundness of the various rules of $\rightharpoonup^{E}$. Many of the lemmas are trivial or uninformative, and so will not be discussed here. The interested reader can find them in appendices A.2 and A.3.

A first informative proof to look at is the proof that $[\mathsf{trap}]$ is sound:

Lemma 8 (trap is sound)**.**

For all $\mathbf{p}^{S}$, $[\![(\mathsf{trap}\ \mathbf{p}^{S})]\!] \simeq^{C} [\![\downarrow^{P} \mathbf{p}^{S}]\!]$

The full proof may be found at lemma 40 (trap is sound). The first thing to note is that this proof does not require the premise that we are in the first instant, or correct binding. Many of the equations do not touch pause or binding forms, and therefore are not sensitive to instants or binding. This proofs proceeds by cases on the structure of $\mathbf{p}^{S}$. The case where are $\mathbf{p}^{S} = \mathsf{nothing}$ invokes the solver.

We can also see that these two are the same if we draw out the circuits on paper: they give us the same graph! The last case is $\mathbf{p}^{S} = (\mathsf{exit}\ \mathbf{n})$. In this case we do cases on $\downarrow^{P}$. In the first of these cases we have $\mathbf{p}^{S} = (\mathsf{exit}\ 0)$, we have a concrete circuit, and so can use the solver again. In the last case we have $\mathbf{p}^{S} = (\mathsf{exit}\ \mathbf{n})$, where $\mathbf{n} > 0$. Again if we draw this out we get the exact same graph.

The next proof of interest is the proof that $[\textbf{emit}]$ is sound. This proof is more complex because it must deal with both evaluation contexts and environments.

LEMMA 9 (EMIT IS SOUND)**.**

For all $\mathbf{r}^{\mathrm{p}} = (\varrho\ \langle \theta^{\mathrm{r}}, \mathrm{GO}\rangle.\ \mathbf{E}^{\mathrm{p}}[(\mathrm{emit}\ \mathbf{S})])$,
$[\![(\varrho\ \langle \theta^{\mathrm{r}}, \mathrm{GO}\rangle.\ \mathbf{E}^{\mathrm{p}}[(\mathrm{emit}\ \mathbf{S})])]\!] \simeq^{C} [\![(\varrho\ \langle (\theta^{\mathrm{r}} \leftarrow \{\ \mathbf{S} \mapsto 1\ \}), \mathrm{GO}\rangle.\ \mathbf{E}^{\mathrm{p}}[\mathrm{nothing}])]\!]$

The full proof is given in lemma 47 (EMIT IS SOUND). This proof proceeds by induction over $\mathbf{E}^{\mathrm{p}}$. The base case is rather trivial: when $\mathbf{E}^{\mathrm{p}} = \bigcirc$ the two circuits look identical, as the 1 from the GO wire is directly connected to the $\mathbf{S}$ wire. The inductive case is more interesting: the proof uses the idea that evaluation contexts obey the property that in $[\![\mathbf{E}^{\mathrm{p}}[\mathbf{p}^{\mathrm{p}}]]\!]$, the GO and signal wires from the top of the term are passed, unchanged, to the subcircuit for $[\![\mathbf{p}^{\mathrm{p}}]\!]$. This is stated formally with these two lemmas:

LEMMA 10 (S IS MAINTAINED ACROSS E)**.**

For all $\mathbf{p}^{\mathrm{p}}{}_i = \mathbf{E}^{\mathrm{p}}[\mathbf{q}^{\mathrm{p}}{}_i]$, *and* $\mathbf{S}$, *if* $\mathbf{S}^{\mathrm{i}} \in \mathit{inputs}([\![\mathbf{p}^{\mathrm{p}}{}_i]\!])$ *then* $[\![\mathbf{q}^{\mathrm{p}}{}_i]\!](\mathbf{S}^{\mathrm{i}}) \simeq [\![\mathbf{p}^{\mathrm{p}}{}_i]\!](\mathbf{S}^{\mathrm{i}})$

LEMMA 11 (GO IS MAINTAINED ACROSS E)**.**

For all $\mathbf{p}^{\mathrm{p}} = \mathbf{E}^{\mathrm{p}}[\mathbf{q}^{\mathrm{p}}]$, $[\![\mathbf{q}^{\mathrm{p}}]\!](\mathrm{GO}) \simeq [\![\mathbf{p}^{\mathrm{p}}]\!](\mathrm{GO})$

The full proofs of which are given in lemma 79 (S IS MAINTAINED ACROSS E) and lemma 80 (GO IS MAINTAINED ACROSS E). Both lemmas follow directly by induction on $\mathbf{E}^{\mathrm{p}}$ and the definition of $[\![\cdot]\!]$. These two lemmas together give that the inputs of the subcircuit are unchanged by the context. The remainder of the inductive case for lemma 47 (EMIT IS SOUND) follows from the notion that the $\mathbf{S}^{\mathrm{o}}$ wires are always or'ed with each other, therefore a 1 in any subterm leads to the overall signal wire being 1.

The last proof described here is the proof for $[\textbf{is-absent}]$:

LEMMA 12 (IS-ABSENT IS SOUND)**.**

For all $\mathbf{r}^{\mathrm{p}} = (\varrho\ \langle \theta, \mathbf{A}\rangle.\ \mathbf{E}^{\mathrm{p}}[(\text{if } \mathbf{S}\ \mathbf{p}^{\mathrm{p}}\ \mathbf{q}^{\mathrm{p}})])$,

if $\theta(\mathbf{S}) = \perp$,

$\mathbf{S} \notin \mathit{Can}^{S}_{\varrho}(\ (\varrho\ \langle \theta, \mathbf{A}\rangle.\ \mathbf{E}^{\mathrm{p}}[(\text{if } \mathbf{S}\ \mathbf{p}^{\mathrm{p}}\ \mathbf{q}^{\mathrm{p}})]),\ \{\}\ )$ *and,*

$[\![(\varrho\ \langle \theta, \mathbf{A}\rangle.\ \mathbf{E}^{\mathrm{p}}[(\text{if } \mathbf{S}\ \mathbf{p}^{\mathrm{p}}\ \mathbf{q}^{\mathrm{p}})])]\!](\mathrm{SEL}) \simeq 0$,

then

$[\![(\varrho\ \langle \theta, \mathbf{A}\rangle.\ \mathbf{E}^{\mathrm{p}}[(\text{if } \mathbf{S}\ \mathbf{p}^{\mathrm{p}}\ \mathbf{q}^{\mathrm{p}})])]\!] \simeq^{C} [\![(\varrho\ \langle \theta, \mathbf{A}\rangle.\ \mathbf{E}^{\mathrm{p}}[\mathbf{q}^{\mathrm{p}}])]\!]$

The full proof is given in lemma 49 (IS-ABSENT IS SOUND). This proof is one of key proofs which requires the premise that we are in the first instant. This is because this proof relies on *Can*, which assumes that control will not resume from within a term—that is, it assumes it is in the first instant. Other variations of *Can*, such as those from the State Behavioral Semantics (Berry 2002) or the Constructive Operation Semantics (Potop-Butucaru 2002) drop this assumption by reflecting register state back in the syntax of the program.

This proof is essentially a chaining of several other lemmas. As with lemma 47 (EMIT IS SOUND), lemma 79 (S IS MAINTAINED ACROSS E) and lemma 80 (GO IS MAINTAINED ACROSS E) are used to shed the evaluation contexts in the rule. From there the proof follows from the following lemma:

LEMMA 13 (CAN S IS SOUND)**.**

For any term and environment $\mathbf{p}^{\mathrm{p}}$ *and* θ *and any signal* $\mathbf{S}$, *if* $[\![\mathbf{p}^{\mathrm{p}}]\!] \setminus \theta$, $\mathbf{S} \notin \mathit{Can}^{S}(\ \mathbf{p}^{\mathrm{p}}, \theta\ )$, *and* $[\![\mathbf{p}^{\mathrm{p}}]\!](\mathrm{SEL}) \simeq 0$, *then* $[\![\mathbf{p}^{\mathrm{p}}]\!](\mathbf{S}^{\mathrm{o}}) \simeq 0$

To understand this proof statement, I must explain a little bit of notation. The phrase $[\![\mathbf{p}^{\mathrm{p}}]\!] \setminus \theta$ exists to tie the syntactic world of Esterel to the circuit world. It, in essence, states that the knowledge contained in the map θ also holds when reasoning about the circuit. Formally, it is defined as:

Definition: $[\![\mathbf{p}^{\mathrm{p}}]\!] \setminus \theta$

read as: θ binds $[\![\mathbf{p}^{\mathrm{p}}]\!]$

$[\![\mathbf{p}^{\mathrm{p}}]\!] \setminus \theta$ *if and only if* $\forall\ \mathbf{S} \in \mathit{dom}(\theta),\ \theta(\mathbf{S}) = 1 \Leftrightarrow [\![\mathbf{p}^{\mathrm{p}}]\!](\mathbf{S}^{\mathrm{i}}) \simeq 1$, *and* $\theta(\mathbf{S}) = 0 \Leftrightarrow [\![\mathbf{p}^{\mathrm{p}}]\!](\mathbf{S}^{\mathrm{i}}) \simeq 0$.

With this in hand we can interpret lemma 71 (CAN S IS SOUND): If we restrict our view to the first instant, and the environment given to *Can* is valid with respect to the circuit, then *Can* accurately predicts when signal wires will be set to 0 (or rather, the complement of *Can* accurately predicts this).

The proof of lemma 71 (CAN S IS SOUND) proceeds by induction over the structure of $\mathbf{p}^{\mathrm{p}}$, following the cases laid out by *Can*. The majority of this lemma consists of tracing how the definition of *Can* walks the program, and compares that to the structure of the generate circuit. In most cases the result follows directly. In the end there are two interesting cases: signal and seq. The signal case in interesting only when the bound signal is not in the result of *Can*. In this case we must use our inductive hypothesis to show that the output of the bound signal is 0, and use that to invoke our inductive hypothesis to show that the goal signal is also 0. The seq case of *Can* relies on the return codes. Thus we use an auxiliary lemma to reason about those codes:

LEMMA 14 (CAN K IS SOUND)**.**

For any term and environment $\mathbf{p}^{\mathrm{p}}$ *and* θ *and any return code* κ, *if* $[\![\mathbf{p}^{\mathrm{p}}]\!] \setminus \theta$, $\kappa \notin \mathit{Can}^{\mathrm{K}}(\mathbf{p}^{\mathrm{p}}, \theta)$, *and*
$[\![\mathbf{p}^{\mathrm{p}}]\!](\mathrm{SEL}) \simeq 0$, *then* $[\![\mathbf{p}^{\mathrm{p}}]\!](\mathrm{K}\kappa) \simeq 0$

This lemma is similar to lemma 71 (CAN S IS SOUND), except that it tells us which return code wires must be 0. It is proved in essentially the same way as lemma 71 (CAN S IS SOUND).

These two lemmas also have counterparts for Can_0:

LEMMA 15 (CAN RHO S IS SOUND)**.**

For all $\mathbf{p}^{\mathrm{p}}$, θ, $\mathbf{A}$, $\mathbf{S}$, *if* $\mathbf{S} \notin \mathit{Can}_{\varrho}^{\mathrm{S}}(\!|(\varrho\ \langle \theta^{\mathrm{r}}, \mathbf{A}\rangle.\ \mathbf{p}^{\mathrm{p}}), \{\}|\!)$ *and* $[\![(\varrho\ \langle \theta^{\mathrm{r}}, \mathbf{A}\rangle.\ \mathbf{p}^{\mathrm{p}})]\!](\mathrm{SEL}) \simeq 0$ *then* $[\![(\varrho\ \langle \theta^{\mathrm{r}}, \mathbf{A}\rangle.\ \mathbf{p}^{\mathrm{p}})]\!](\mathbf{S}^{\mathrm{o}}) \simeq 0$

LEMMA 16 (CAN RHO K IS SOUND)**.**

For any term and environment $\mathbf{p}^{\mathrm{p}}$ *and* θ *and* $\mathbf{A}$, *and return code* $\boldsymbol{\kappa}$ *if* $\boldsymbol{\kappa} \notin \mathit{Can}_{\varrho}^{\mathrm{K}}(\!|(\varrho\ \langle \theta^{\mathrm{r}}, \mathbf{A}\rangle.\ \mathbf{p}^{\mathrm{p}}), \{\}|\!)$,
and $[\![(\varrho\ \langle \theta^{\mathrm{r}}, \mathbf{A}\rangle.\ \mathbf{p}^{\mathrm{p}})]\!](\mathrm{SEL}) \simeq 0$, *then* $[\![(\varrho\ \langle \theta^{\mathrm{r}}, \mathbf{A}\rangle.\ \mathbf{p}^{\mathrm{p}})]\!](\mathrm{K}\boldsymbol{\kappa}) \simeq 0$

However as Can_{ϱ} is essentially just repeated applications of the signal case of Can, these proofs are relatively uninteresting.

4.2.2. Changing the compiler for Soundness

The compiler used here differs from the compiler in Berry (2002) in how it handles the compilation of par. Specifically the changes are the KILL wire including the return codes, and the definition of the LEM and REM wires. The old compiler broadcasts the KILL wires directly to the subcircuit. In addition it defines REM = ¬GO ∨ p-SEL (and LEM similarly q-SEL). Both of these definitions result in unsoundness under local rewrites to the syntax of the program that should hold. Both of these changes were used in the Esterel v7 compiler, however they have not yet been published anywhere.[1]

The change to the REM and LEM wires can be explained by noticing that we expect

$$(\text{trap (par (exit 0) pause)}) \simeq^{E} \text{nothing}$$

as this circuit will always abort, and immediately catch the exit. Therefore we want

$$[\![(\text{trap (par (exit 0) pause)})]\!] \simeq^{C} [\![\text{nothing}]\!]$$

[1] The information about these changes comes from personal communication with Gérard Berry.

to hold. However in this equivalence did not hold in the Berry (2002) compiler, specifically in the case where GO = $\perp$. In that case both LEM and REM are $\perp$, causing K2 to be $\perp$. However in the circuit for nothing, K2 is defined to be 0. The Esterel v7 compiler handles this case correctly.

The change to the KILL wire can be seen by a similar example. We expect

$$(\text{par } (\text{exit } 2) \text{ pause}) \simeq^{E} (\text{exit } 2)$$

for a similar reason as before. As before, this means we want

$$[\![(\text{par } (\text{exit } 2) \text{ pause})]\!] \simeq^{C} [\![(\text{exit } 2)]\!]$$

to hold. With the Berry (2002) compiler, however, we are free to let KILL be 0, even when GO is 1. This means that in the second instant the pause is selected. However there is no pause in (exit 2), therefore its SEL wire must be 0. The new compiler handles this correctly by killing the other branch of the par even if the outer KILL wire is 0.

Note that these two issues both rely on both subcircuits being simultaneously active at some point to trigger the change in behavior. As par is the only case where two subcircuits can be active simultaneously, this is the only case that requires this special care.

It should be noted that equality violations above do not constitute a bug in actual Esterel compiler implementations. In actual compilers, if GO is $\perp$, then the program raise an error and the different in wire states cannot be observed. In addition if an exit code is raised, then the KILL wire will be set, as full programs must be closed and therefore an external trap will catch the code and set the KILL wire. Therefore, behaviorally at least, the changes used here should not effect an actual Esterel implementation.

4.3. Justifying Adequacy

Adequacy is the statement that a calculus can define an evaluator for it's language. In this case, we want Computational Adequacy, which is the statement that the calculus's evaluator is equivalent to the ground truth evaluator:

THEOREM 17 (COMPUTATIONAL ADEQUACY)**.**

For all $\mathbf{p}^{\mathrm{p}}$, $\mathbf{O}$, *if* $\mathit{closed}(\mathbf{p}^{\mathrm{p}}_{\mathrm{GO}})$ *and* $[\![\mathbf{p}^{\mathrm{p}}_{\mathrm{GO}}]\!](\mathrm{SEL}) \simeq 0$ *then*
$\mathit{eval}^{\mathrm{E}}(\mathbf{O}, \mathbf{p}^{\mathrm{p}}_{\mathrm{GO}}) = \langle \theta\ , \mathbf{bool} \rangle$ *if and only if* $\mathit{eval}^{\mathrm{C}}(\mathbf{O}, [\![\mathbf{p}^{\mathrm{p}}_{\mathrm{GO}}]\!]) = \langle \theta\ , \mathbf{bool} \rangle$

The full proof can be found at theorem 30 (COMPUTATIONAL ADEQUACY). The first premise of this theorem requires that the program be *closed*, as the evaluator is only really meant to work on full programs. However *closed* here is slightly different from the usual definition, because it restricts programs to those which will also generate closed circuits which will execute their first instant. Formally, a program is *closed* if it is a ϱ term with the control variable GO, and it has no free variables:

Definition: $\mathit{closed}(\mathbf{p}^{\mathrm{p}}_{\mathrm{GO}})$

$$\frac{\mathit{FV}(\varrho\ \langle \theta^{\mathrm{r}}, \mathrm{GO} \rangle.\ \mathbf{q}^{\mathrm{p}}) = \varnothing}{\mathit{closed}(\varrho\ \langle \theta^{\mathrm{r}}, \mathrm{GO} \rangle.\ \mathbf{q}^{\mathrm{p}})}$$

By setting $\mathbf{A}$ to GO we force the GO wire in the compilation to be 1, which causes the circuit to execute its first instant. The next premise is the usual statement that we are only observing the first instant of execution. The conclusion of the proof states that the output signals and constructivity from the two evaluators are the same.

To complete the proof we use a set of canonical forms for terms in the calculus. Any closed term is equal to such a canonical term, and that these canonical forms are the exact cases that $\mathit{eval}^{\mathrm{E}}$ looks at. These canonical forms are equivalent modulo **[par-swap]**, meaning that, while a canonical form, they may still step via **[par-swap]**, but may not take any other steps. To prove this $\longrightarrow^{\mathrm{E}}$ is broken up into two parts: $\longrightarrow^{\mathrm{S}}$, which contains only the compatible closure of **[par-swap]**, and $\longrightarrow^{\mathrm{R}}$, which is the compatible closure of every other rule.[2] With that we can state theorem about these canonical forms like so:

[2]The S stands for "swap", and the R stands for "remainder".

Lemma 18 (Non-stepping terms are values)**.**

For all $\mathbf{q}^{\mathrm{p}} = (\varrho\ \langle \theta^{\mathrm{r}}, \mathbf{A}\rangle.\ \mathbf{p}^{\mathrm{p}})$,
If $\mathit{closed}(\mathbf{q}^{\mathrm{p}})$, *and there does not exists any* ${\theta^{\mathrm{r}}}_o$ *and* ${\mathbf{p}^{\mathrm{p}}}_o$ *such that (either* $\mathbf{q}^{\mathrm{p}} \longrightarrow^{\mathrm{R}} (\varrho\ \langle {\theta^{\mathrm{r}}}_o, \mathbf{A}\rangle.\ \mathbf{E}^{\mathrm{p}}[{\mathbf{p}^{\mathrm{p}}}_o])$ *or there exists some* r *such that* $\mathbf{q}^{\mathrm{p}} \longrightarrow^{\mathrm{S}} (\varrho\ \langle \theta^{\mathrm{r}}, \mathbf{A}\rangle.\ \mathbf{E}^{\mathrm{p}}[\mathbf{r}^{\mathrm{p}}]) \longrightarrow^{\mathrm{R}} (\varrho\ \langle \theta_o, \mathbf{A}\rangle.\ \mathbf{E}^{\mathrm{p}}[{\mathbf{p}^{\mathrm{p}}}_o])$*) then either* $\mathbf{p}^{\mathrm{p}} \in \mathbf{p}^{\mathrm{D}}$ *or* $\theta^{\mathrm{r}}; \mathbf{A}; \bigcirc \vdash_{\mathrm{B}} \mathbf{p}^{\mathrm{p}}$

The full proof is in lemma 69 (Non-stepping terms are values). To unpack this: The proof states that canonical forms are forms which both cannot step by $\longrightarrow^{\mathrm{R}}$ and if they step by $\longrightarrow^{\mathrm{S}}$, then the resulting form also cannot step by $\longrightarrow^{\mathrm{R}}$. We only need to check for one step of $\longrightarrow^{\mathrm{S}}$, because if multiple $\longrightarrow^{\mathrm{S}}$ could uncover a reduction in $\longrightarrow^{\mathrm{R}}$, then there would exist some term which would be one step $\longrightarrow^{\mathrm{S}}$ away from a $\longrightarrow^{\mathrm{R}}$ reduction which would violate the lemma. The negative existential in this would make it tricky to prove. However, we are in luck: everything used in this statement is decidable. Therefore this is proved by proving it's contrapositive:

Lemma 19 (Not values must step)**.**

For all $\mathbf{q}^{\mathrm{p}} = (\varrho\ \langle \theta^{\mathrm{r}}, \mathbf{A}\rangle.\ \mathbf{E}^{\mathrm{p}}[\mathbf{p}^{\mathrm{p}}])$, *If* $\mathit{closed}(\mathbf{q}^{\mathrm{p}})$, $\mathbf{p}^{\mathrm{p}} \notin \mathbf{p}^{\mathrm{D}}$, *and* $\theta^{\mathrm{r}}; \mathbf{A}; \mathbf{E}^{\mathrm{p}} \nvdash_{\mathrm{B}} \mathbf{p}^{\mathrm{p}}$ *then there exists some* ${\theta^{\mathrm{r}}}_o$ *and* ${\mathbf{p}^{\mathrm{p}}}_o$ *such that either* $\mathbf{q}^{\mathrm{p}} \longrightarrow^{\mathrm{R}} (\varrho\ \langle \theta_o, \mathbf{A}\rangle.\ \mathbf{E}^{\mathrm{p}}[{\mathbf{p}^{\mathrm{p}}}_o])$ *or there exists some* $\mathbf{r}^{\mathrm{p}}$ *such that* $\mathbf{q}^{\mathrm{p}} \longrightarrow^{\mathrm{S}} (\varrho\ \langle \theta^{\mathrm{r}}, \mathbf{A}\rangle.\ \mathbf{E}^{\mathrm{p}}[\mathbf{r}^{\mathrm{p}}]) \longrightarrow^{\mathrm{R}} (\varrho\ \langle {\theta^{\mathrm{r}}}_o, \mathbf{A}\rangle.\ \mathbf{E}^{\mathrm{p}}[{\mathbf{p}^{\mathrm{p}}}_o])$

The full proof can be found at lemma 70 (Not values must step). This proof states that if a term is not one of our canonical forms, then it must be able to either take a step in $\longrightarrow^{\mathrm{R}}$, or step by $\longrightarrow^{\mathrm{S}}$ then by $\longrightarrow^{\mathrm{R}}$. The proof of this follows by induction of $\mathbf{p}^{\mathrm{p}}$, with some case analysis on $\vdash_{\mathrm{B}}$ and $\mathbf{p}^{\mathrm{D}}$ along the way.

Beyond this, it is the case that $\longrightarrow^{\mathrm{R}}$ is a strongly canonicalizing relation. Therefore it must be the case that we can reach a canonical form using a finite number of $\longrightarrow^{\mathrm{R}}$ and $\longrightarrow^{\mathrm{S}}$ steps:

Lemma 20 (Strongly Canonicalizing)**.**

For all $\mathbf{p}^{\mathrm{p}}_{\mathrm{GO}}$, $\mathbf{q}^{\mathrm{p}}_{\mathrm{GO}}$, *if* $\mathbf{p}^{\mathrm{p}}_{\mathrm{GO}} \longrightarrow^{\mathrm{R}} \mathbf{q}^{\mathrm{p}}_{\mathrm{GO}}$, *then* $\mathscr{S}(\mathbf{p}^{\mathrm{p}}_{\mathrm{GO}}) > \mathscr{S}(\mathbf{q}^{\mathrm{p}}_{\mathrm{GO}})$.

The full proof is given in lemma 57 (Strongly Canonicalizing). The function $\mathscr{S}$ acts as an estimate on the number of steps that a term can take. Because it is strictly decreasing and gives back a non-negative number, we must eventually reach a case where no more $\longrightarrow^{R}$ steps can be taken. Whats more its easy to show that $\longrightarrow^{S}$ does not change the count, therefore there always exists a finite reduction path to one of these canonical forms. Therefore all closed terms are ≡^esterel to some canonical term.

Now that we have show that there exist canonical forms, and that every closed pure Esterel term is ≡^esterel to one of these forms, we know that eval^{E} is defined on all closed pure terms. The next step in proving adequacy is to show that these two canonical forms give back the same signal set as their circuit compilation. Fortunately this follows fairly directly from soundness, as we know that our canonical forms are $\equiv^{E}$ to the original term, and that $\equiv^{E}$ is sound with respect to the circuit compilation.

The final step is to show that the two types of canonical forms map exactly to constructive and non-constructive circuits respectively. The simpler of these is:

Lemma 21 (Esterel Value is Circuit Value)**.**

Forall $(\varrho\ \langle \theta^{r}, \mathrm{GO}\rangle.\ \mathbf{p}^{D})$, *if* $\mathit{complete\text{-}wrt}(\theta^{r}, \mathbf{p}^{D})$, $(\varrho\ \langle \theta^{r}, \mathrm{GO}\rangle.\ \mathbf{p}^{D})$ *is closed, and*

$[\![(\varrho\ \langle \theta^{r}, \mathrm{GO}\rangle.\ \mathbf{p}^{D})]\!](\mathrm{RES}) = [\![(\varrho\ \langle \theta^{r}, \mathrm{GO}\rangle.\ \mathbf{p}^{D})]\!](\mathrm{SUSP}) = [\![(\varrho\ \langle \theta^{r}, \mathrm{GO}\rangle.\ \mathbf{p}^{D})]\!](\mathrm{KILL}) = 0$, *and* $[\![(\varrho\ \langle \theta^{r}, \mathrm{GO}\rangle.\ \mathbf{p}^{D})]\!](\mathrm{GO}) = 1$.

then $[\![(\varrho\ \langle \theta^{r}, \mathrm{GO}\rangle.\ \mathbf{p}^{D})]\!]$ *is constructive.*

Which is proved fully in lemma 61 (Done is Constructive). Note that the premise $\mathit{complete\text{-}wrt}(\theta^{r}, \mathbf{p}^{D})$ always holds, by the proof can$_S$-done from the Agda code base which states that the result of Can^{S} on any $\mathbf{p}^{D}$ is empty. The premises about the control wires are given by the definition of $[\![\cdot]\!]$, and by the fact that unassigned input wires are set to 0 by eval^{C}. This proof follows by induction on the structure of $\mathbf{p}^{D}$.

The other side, the statement that $\vdash_{B}$ corresponds to non-constructive circuits is given by:

LEMMA 22 (BLOCKED TERMS ARE NON-CONSTRUCTIVE)**.**

For all $\mathbf{r}^{\mathrm{p}}{}_{outer} = (\varrho\ \langle \theta^{\mathrm{r}}, \mathrm{GO}\rangle.\ \mathbf{r}^{\mathrm{p}})$, *if* $\theta^{\mathrm{r}}; \mathrm{GO}; \bigcirc \vdash_{\mathrm{B}} \mathbf{r}^{\mathrm{p}}$, *and* $[\![(\varrho\ \langle \theta^{\mathrm{r}}, \mathrm{GO}\rangle.\ \mathbf{r}^{\mathrm{p}})]\!](\mathrm{SEL}) \simeq 0$ *then* $[\![\mathbf{r}^{\mathrm{p}}{}_{outer}]\!]$ *is non-constructive.*

The proof of which can be found at lemma 62 (BLOCKED TERMS ARE NON-CONSTRUCTIVE). The proof of this lemma is complex. It relies on a subject-reduction lemma which shows that, as the circuit reduction relation $\longrightarrow^{\mathrm{C}}$ steps through the term, the wires which are in Can^{S} cannot change from $\bot$. The core of this lemma is another subject-reduction lemma which shows that, assuming GO is $\bot$, *Can* is perfectly adequate to describe evaluation:

LEMMA 23 (ADEQUACY OF CAN)**.**

For all $\mathbf{r}^{\mathrm{p}}$, θ, $\theta^{\mathfrak{c}}{}_1$, $\theta^{\mathfrak{c}}{}_2$ *let* $\mathfrak{c} = [\![\mathbf{r}^{\mathrm{p}}]\!]$, *if* $\theta^{\mathfrak{c}}{}_1 \longrightarrow^{\mathrm{C}} \theta^{\mathfrak{c}}{}_2$, $[\![\mathbf{r}^{\mathrm{p}}]\!](\mathrm{SEL}) \simeq 0$, $[\![\mathbf{r}^{\mathrm{p}}]\!] \setminus \theta,\ \theta^{\mathfrak{c}}{}_1(\mathrm{GO}) = \bot$, *and* $nc(\mathbf{r}^{\mathrm{p}}, \theta, \theta^{\mathfrak{c}}{}_1)$ *then* $nc(\mathbf{r}^{\mathrm{p}}, \theta, \theta^{\mathfrak{c}}{}_2)$

The full proof is in lemma 67 (ADEQUACY OF CAN). To unpack: If we are given some term $\mathbf{r}^{\mathrm{p}}$, and two circuit states $\theta^{\mathfrak{c}}{}_1$ and $\theta^{\mathfrak{c}}{}_2$ such that both circuit states are states of the compilation of $\mathbf{r}^{\mathrm{p}}$, and $\theta^{\mathfrak{c}}{}_1$ steps to $\theta^{\mathfrak{c}}{}_2$, and we know about the signals of $[\![\mathbf{r}^{\mathrm{p}}]\!]$ via θ, and we know that the GO wire of $\theta^{\mathfrak{c}}{}_1$ is currently bottom, then the invariant *nc* is preserved. The invariant *nc* (figure 37) is formed of three judgments. The first of these, *nc-S*, says that any signal wire is currently $\bot$ if it is both Can^{S} and is $\bot$ in θ. The second, all-bot-κ says the same, but for Can^{K}, return codes, and their wires.

The last of the judgments, *nc-r* (figure 38) looks complex, but all it says is that the *nc* judgment holds for subterms, subcircuits, and environments that match how *Can* recurs over the term. Together all of these properties mean that "*Can* accurately predicts when wires are $\bot$". Therefore the overall proof states that "*Can* accurately predicts when wires are $\bot$ when GO is $\bot$"[3] The last step in completing this proof is to argue that initial states are always *nc*, but this follows fairly directly since wires are internal and output wires are initialized to $\bot$ in the initial state. Note that in this judgment the metafunction *sub* extracts the substate of the circuit corresponding to the given subterm.

[3] This is why I call this proof "adequacy". When combined with the soundness of *Can*, it tells us when *Can* gives a complete evaluator.

Definition: $nc(\mathbf{p}^{\mathrm{p}}, \theta, \theta^{\mathfrak{C}})$

$$\frac{nc\text{-}\kappa(\mathbf{p}^{\mathrm{p}}, \theta, \theta^{\mathfrak{C}}) \qquad nc\text{-}\mathcal{S}(\mathbf{p}^{\mathrm{p}}, \theta, \theta^{\mathfrak{C}}) \qquad nc\text{-}\mathcal{N}(\mathbf{p}^{\mathrm{p}}, \theta, \theta^{\mathfrak{C}})}{nc(\mathbf{p}^{\mathrm{p}}, \theta, \theta^{\mathfrak{C}})}$$

Definition: $nc\text{-}\kappa(\mathbf{p}^{\mathrm{p}}, \theta, \theta^{\mathfrak{C}})$

$$\frac{\forall\, \mathbf{n} \in \mathit{Can}^{\mathrm{K}}(\mathbf{p}^{\mathrm{p}}, \theta),\ \theta^{\mathfrak{C}}(\mathrm{K}\mathbf{n}) = \bot}{nc\text{-}\kappa(\mathbf{p}^{\mathrm{p}}, \theta, \theta^{\mathfrak{C}})}$$

Definition: $nc\text{-}\mathcal{S}(\mathbf{p}^{\mathrm{p}}, \theta, \theta^{\mathfrak{C}})$

$$\frac{\forall\, \mathbf{S} \in \mathit{Can}^{\mathrm{S}}(\mathbf{p}^{\mathrm{p}}, \theta),\ \theta(\mathbf{S}) = \bot \Rightarrow \theta^{\mathfrak{C}}(\mathbf{S}^{\mathrm{i}}) = \theta^{\mathfrak{C}}(\mathbf{S}^{\mathrm{o}}) = \bot}{nc\text{-}\mathcal{S}(\mathbf{p}^{\mathrm{p}}, \theta, \theta^{\mathfrak{C}})}$$

Figure 37: The smaller judgments for *nc*

At it's core the proof of lemma 67 (Adequacy of Can) holds because all return code and signal wires are and'ed with the GO wire, therefore they can never be set to 1 unless the GO wire is 1, and they can only be set to 0 when they are not in $\mathit{Can}^{\mathrm{S}}$. From this we can argue that lemma 62 (Blocked terms are non-constructive) holds, essentially, because the GO wires the leaves of the $\vdash_{\mathrm{B}}$ must be blocked on a signal wire, and therefore they depend on a GO which itself depends on one of these signal, and therefore that GO wire must always remain bottom.

4.4. Justifying Consistency

Consistency, at it's core, means that a theory cannot disagree with itself. In the case of Esterel this can be boiled down to a single property: That $\mathit{eval}^{\mathrm{E}}$ is a function. Or, more formally:

$$\frac{}{\mathit{nc\text{-}r}(\text{nothing}, \theta, \theta^{\mathsf{c}})}\,[\text{nothing}] \qquad \frac{}{\mathit{nc\text{-}r}((\text{exit } \mathbf{n}), \theta, \theta^{\mathsf{c}})}\,[\text{exit}]$$

$$\frac{}{\mathit{nc\text{-}r}((\text{emit } \mathbf{S}), \theta, \theta^{\mathsf{c}})}\,[\text{emit}] \qquad \frac{}{\mathit{nc\text{-}r}(\text{pause}, \theta, \theta^{\mathsf{c}})}\,[\text{pause}]$$

$$\frac{\mathit{nc}(\mathbf{p}^{\mathsf{p}}, \theta, \mathit{sub}((\text{trap } \mathbf{p}^{\mathsf{p}}), \mathbf{p}^{\mathsf{p}}, \theta^{\mathsf{c}}))}{\mathit{nc\text{-}r}((\text{trap } \mathbf{p}^{\mathsf{p}}), \theta, \theta^{\mathsf{c}})}\,[\text{trap}] \qquad \frac{\mathit{nc}(\mathbf{p}^{\mathsf{p}}, \theta, \mathit{sub}((\text{suspend } \mathbf{p}^{\mathsf{p}}\ \mathbf{S}), \mathbf{p}^{\mathsf{p}}, \theta^{\mathsf{c}}))}{\mathit{nc\text{-}r}((\text{suspend } \mathbf{p}^{\mathsf{p}}\ \mathbf{S}), \theta, \theta^{\mathsf{c}})}\,[\text{suspend}]$$

$$\frac{\theta(\mathbf{S}) = 0 \qquad \mathit{nc}(\mathbf{q}^{\mathsf{p}}, \theta, \mathit{sub}((\text{if } \mathbf{S}\ \mathbf{p}^{\mathsf{p}}\ \mathbf{q}^{\mathsf{p}}), \mathbf{q}^{\mathsf{p}}, \theta^{\mathsf{c}}))}{\mathit{nc\text{-}r}((\text{if } \mathbf{S}\ \mathbf{p}^{\mathsf{p}}\ \mathbf{q}^{\mathsf{p}}), \theta, \theta^{\mathsf{c}})}\,[\text{if-0}] \qquad \frac{\theta(\mathbf{S}) = 1 \qquad \mathit{nc}(\mathbf{p}^{\mathsf{p}}, \theta, \mathit{sub}((\text{if } \mathbf{S}\ \mathbf{p}^{\mathsf{p}}\ \mathbf{q}^{\mathsf{p}}), \mathbf{p}^{\mathsf{p}}, \theta^{\mathsf{c}}))}{\mathit{nc\text{-}r}((\text{if } \mathbf{S}\ \mathbf{p}^{\mathsf{p}}\ \mathbf{q}^{\mathsf{p}}), \theta, \theta^{\mathsf{c}})}\,[\text{if-1}]$$

$$\frac{\theta(\mathbf{S}) = \bot \qquad \mathit{nc}(\mathbf{p}^{\mathsf{p}}, \theta, \mathit{sub}((\text{if } \mathbf{S}\ \mathbf{p}^{\mathsf{p}}\ \mathbf{q}^{\mathsf{p}}), \mathbf{p}^{\mathsf{p}}, \theta^{\mathsf{c}})) \qquad \mathit{nc}(\mathbf{q}^{\mathsf{p}}, \theta, \mathit{sub}((\text{if } \mathbf{S}\ \mathbf{p}^{\mathsf{p}}\ \mathbf{q}^{\mathsf{p}}), \mathbf{q}^{\mathsf{p}}, \theta^{\mathsf{c}}))}{\mathit{nc\text{-}r}((\text{if } \mathbf{S}\ \mathbf{p}^{\mathsf{p}}\ \mathbf{q}^{\mathsf{p}}), \theta, \theta^{\mathsf{c}})}\,[\text{if-}\bot]$$

$$\frac{0 \notin \mathit{Can}^{\mathrm{K}}(\mathbf{p}^{\mathsf{p}}, \theta) \qquad \mathit{nc}(\mathbf{p}^{\mathsf{p}}, \theta, \mathit{sub}((\text{seq } \mathbf{p}^{\mathsf{p}}\ \mathbf{q}^{\mathsf{p}}), \mathbf{p}^{\mathsf{p}}, \theta^{\mathsf{c}}))}{\mathit{nc\text{-}r}((\text{seq } \mathbf{p}^{\mathsf{p}}\ \mathbf{q}^{\mathsf{p}}), \theta, \theta^{\mathsf{c}})}\,[\text{seq-}\neg 0]$$

$$\frac{0 \in \mathit{Can}^{\mathrm{K}}(\mathbf{p}^{\mathsf{p}}, \theta) \qquad \mathit{nc}(\mathbf{p}^{\mathsf{p}}, \theta, \mathit{sub}((\text{seq } \mathbf{p}^{\mathsf{p}}\ \mathbf{q}^{\mathsf{p}}), \mathbf{p}^{\mathsf{p}}, \theta^{\mathsf{c}})) \qquad \mathit{nc}(\mathbf{q}^{\mathsf{p}}, \theta, \mathit{sub}((\text{seq } \mathbf{p}^{\mathsf{p}}\ \mathbf{q}^{\mathsf{p}}), \mathbf{q}^{\mathsf{p}}, \theta^{\mathsf{c}}))}{\mathit{nc\text{-}r}((\text{seq } \mathbf{p}^{\mathsf{p}}\ \mathbf{q}^{\mathsf{p}}), \theta, \theta^{\mathsf{c}})}\,[\text{seq-0}]$$

$$\frac{\mathit{nc}(\mathbf{p}^{\mathsf{p}}, \theta, \mathit{sub}((\text{par } \mathbf{p}^{\mathsf{p}}\ \mathbf{q}^{\mathsf{p}}), \mathbf{p}^{\mathsf{p}}, \theta^{\mathsf{c}})) \qquad \mathit{nc}(\mathbf{q}^{\mathsf{p}}, \theta, \mathit{sub}((\text{par } \mathbf{p}^{\mathsf{p}}\ \mathbf{q}^{\mathsf{p}}), \mathbf{q}^{\mathsf{p}}, \theta^{\mathsf{c}}))}{\mathit{nc\text{-}r}((\text{par } \mathbf{p}^{\mathsf{p}}\ \mathbf{q}^{\mathsf{p}}), \theta, \theta^{\mathsf{c}})}\,[\text{par}]$$

$$\frac{\mathit{nc\text{-}r}(\mathbf{p}^{\mathsf{p}}, \theta, \mathit{sub}((\varrho\ \langle\{\}, \text{WAIT}\rangle.\ \mathbf{p}^{\mathsf{p}}), \mathbf{p}^{\mathsf{p}}, \theta^{\mathsf{c}}))}{\mathit{nc\text{-}r}((\varrho\ \langle\{\}, \text{WAIT}\rangle.\ \mathbf{p}^{\mathsf{p}}), \theta, \theta^{\mathsf{c}})}\,[\rho\text{-}\{\}]$$

$$\frac{\mathbf{S} \in \mathit{dom}(\theta) \qquad \theta^{\mathrm{r}}(\mathbf{S}) = \bot \qquad \mathbf{S} \notin \mathit{Can}^{\mathrm{S}}_{\varrho}((\varrho\ \langle(\theta^{\mathrm{r}} \setminus \{\mathbf{S}\}), \text{WAIT}\rangle.\ \mathbf{p}^{\mathsf{p}}), \theta \leftarrow \{\ \mathbf{S} \mapsto \bot\ \}) \qquad \mathit{nc\text{-}r}((\varrho\ \langle(\theta^{\mathrm{r}} \setminus \{\mathbf{S}\}), \text{WAIT}\rangle.\ \mathbf{p}^{\mathsf{p}}), \theta \leftarrow \{\ \mathbf{S} \mapsto 0\ \}, \mathit{sub}((\varrho\ \langle\theta^{\mathrm{r}}, \text{WAIT}\rangle.\ \mathbf{p}^{\mathsf{p}}), (\varrho\ \langle(\theta^{\mathrm{r}} \setminus \{\mathbf{S}\}), \text{WAIT}\rangle.\ \mathbf{p}^{\mathsf{p}}), \theta^{\mathsf{c}}))}{\mathit{nc\text{-}r}((\varrho\ \langle\theta^{\mathrm{r}}, \text{WAIT}\rangle.\ \mathbf{p}^{\mathsf{p}}), \theta, \theta^{\mathsf{c}})}\,[\rho\text{-}0]$$

$$\frac{\mathbf{S} \in \mathit{dom}(\theta) \qquad \theta(\mathbf{S}) \neq \bot \qquad \mathit{nc\text{-}r}((\varrho\ \langle(\theta^{\mathrm{r}} \setminus \{\mathbf{S}\}), \text{WAIT}\rangle.\ \mathbf{p}^{\mathsf{p}}), \theta \leftarrow \{\ \mathbf{S} \mapsto 1\ \}, \mathit{sub}((\varrho\ \langle\theta^{\mathrm{r}}, \text{WAIT}\rangle.\ \mathbf{p}^{\mathsf{p}}), (\varrho\ \langle(\theta^{\mathrm{r}} \setminus \{\mathbf{S}\}), \text{WAIT}\rangle.\ \mathbf{p}^{\mathsf{p}}), \theta^{\mathsf{c}}))}{\mathit{nc\text{-}r}((\varrho\ \langle\theta^{\mathrm{r}}, \text{WAIT}\rangle.\ \mathbf{p}^{\mathsf{p}}), \theta, \theta^{\mathsf{c}})}\,[\rho\text{-}\neg\bot]$$

Figure 38: The recursive judgment part of *nc*

THEOREM 24 (CONSISTENCY OF EVAL)**.**

For all $\mathbf{p}^{\mathrm{p}}_{\mathrm{GO}}$ *and* $\mathbf{O}$, *if* $\mathit{closed}(\mathbf{p}^{\mathrm{p}}_{\mathrm{GO}})$, $\mathit{eval}^{\mathrm{E}}(\mathbf{O}, \mathbf{p}^{\mathrm{p}}_{\mathrm{GO}}) = \langle \theta_1 , \mathbf{bool}_1 \rangle$,
and $\mathit{eval}^{\mathrm{E}}(\mathbf{O}, \mathbf{p}^{\mathrm{p}}_{\mathrm{GO}}) = \langle \theta_2 , \mathbf{bool}_2 \rangle$,
then $\langle \theta_1 , \mathbf{bool}_1 \rangle = \langle \theta_2 , \mathbf{bool}_2 \rangle$.

The full proof is given in the appendices as theorem 31 (CONSISTENCY OF EVAL). Usually, consistency is proven using the confluence of the underlying reduction semantics. However, in this case proving confluence is not necessary: consistency here follows as a corollary of the adequacy of the calculus. This is because we know from works such as Berry (2002) and Mendler et al. (2012) that $\mathit{eval}^{\mathrm{C}}$ is a consistent model[4] of circuits, therefore by theorem 30 (COMPUTATIONAL ADEQUACY), $\mathit{eval}^{\mathrm{E}}$ as the same function as $\mathit{eval}^{\mathrm{C}}$ composed with the compiler, then it too must be consistent.

[4]Specifically, Lemma 7 from Berry (2002), and Theorem 1 of Mendler et al. (2012), assuming that the Algabraic semantics they give is equivalent to the reduction semantics. This equivalence is given as Theorem 4 from Berry (2002).

CHAPTER 5

Adding in the rest of Esterel

The evidence that the calculus stands on includes prior work and testing, alongside the proofs. So far I have only given one of those legs, the proofs, and only for the pure, loop free, single instant, fragment of Esterel. This chapter fills in the parts of the calculus which handle loops, host language expressions, and multi-instant execution. It will also give the other kinds of evidence I have for the correctness of the calculus: testing and prior work. While the parts of the calculus given so far stand on all three legs, these new parts are supported only by testing and prior work.

5.1. Loops

The calculus handles loops with two new administrative rules, which rely on the new form:

$$\begin{array}{l}\mathbf{p} ::= .... \mid (\overline{\mathsf{loop}}\ \mathbf{p}\ \mathbf{q}) \\ \mathbf{E} ::= .... \mid (\overline{\mathsf{loop}}\ \mathbf{E}\ \mathbf{q}) \\ \hat{\mathbf{p}} ::= .... \mid (\overline{\mathsf{loop}}\ \hat{\mathbf{p}}\ \mathbf{q})\end{array}$$

This new form, $\overline{\mathsf{loop}}$ represents a loop which has been unrolled once. To understand its usage, the loop rules are:

$$\begin{array}{ll}[\mathbf{loop}] & (\mathsf{loop}\ \mathbf{p}) \rightharpoonup^{E} (\overline{\mathsf{loop}}\ \mathbf{p}\ \mathbf{p}) \\ [\mathbf{loop\hat{}stop\text{-}exit}] & (\overline{\mathsf{loop}}\ (\mathsf{exit}\ \mathbf{n})\ \mathbf{q}) \rightharpoonup^{E} (\mathsf{exit}\ \mathbf{n})\end{array}$$

The first rule, [**loop**] expands a loop into a $\overline{\mathsf{loop}}$. This $(\overline{\mathsf{loop}}\ \mathbf{p}\ \mathbf{q})$ is essentially equivalent to (seq **p** (loop **p**))—that is, it represent one unrolling of a loop—however, unlike seq, the second part is inaccessible: if **p** reduces to nothing, the loop cannot restart, instead the program gets stuck. This handles instantaneous loops, which are an error in Esterel. The second loop rule, [**loop^stop-exit**], is analogous to [**seq-exit**], aborting the loop if its body exits.

$$\frac{\mathcal{BV}(\mathbf{p}) \cap \mathcal{FV}(\mathbf{p}) = \emptyset \qquad \vdash_{\mathrm{CB}} \mathbf{p}}{\vdash_{\mathrm{CB}} (\mathrm{loop}\ \mathbf{p})}\,[\mathrm{loop}]$$

$$\frac{\mathcal{BV}(\mathbf{p}) \cap \mathcal{FV}(\mathbf{q}) = \emptyset \qquad \mathcal{BV}(\mathbf{q}) \cap \mathcal{FV}(\mathbf{q}) = \emptyset \qquad \vdash_{\mathrm{CB}} \mathbf{p} \qquad \vdash_{\mathrm{CB}} \mathbf{q}}{\vdash_{\mathrm{CB}} (\overline{\mathrm{loop}}\ \mathbf{p}\ \mathbf{q})}\,[\mathrm{loop\hat{}stop}]$$

Figure 39: Correct Binding and Loops

5.1.1. Loops and Can

To handle loops we must add two clauses below, which both behave as their bodies do. In the case of $\overline{\mathrm{loop}}$ the right hand side, which is the loops original body, is not analyzed as it *cannot* be reached in this instant.

$$\begin{aligned} \mathit{Can}(\mathrm{loop}\ \mathbf{p}, \theta) &= \mathit{Can}(\mathbf{p}, \theta) \\ \mathit{Can}(\overline{\mathrm{loop}}\ \mathbf{p}\ \mathbf{q}, \theta) &= \mathit{Can}(\mathbf{p}, \theta) \end{aligned}$$

5.1.2. Loops and Blocked

As adding loops adds a new evaluation context, the blocked relation must be extended to handle it. This extension, listed below, treats $\overline{\mathrm{loop}}$ the same as seq.

$$\frac{\theta^{\mathrm{r}}; \mathbf{A}; \mathbf{E}[(\overline{\mathrm{loop}} \bigcirc \mathbf{q})] \vdash_{\mathrm{B}} \mathbf{p}}{\theta^{\mathrm{r}}; \mathbf{A}; \mathbf{E} \vdash_{\mathrm{B}} (\overline{\mathrm{loop}}\ \mathbf{p}\ \mathbf{q})}\,[\mathrm{loop\hat{}stop}]$$

5.1.3. Loops and Correct Binding

We can now turn to the issue of Schizophrenia in the calculus. Correct binding prevents schizophrenia, because at their heart schizophrenic variables are variables which have two instances, and one captures the other. Thus, the loop clause in $\vdash_{\mathrm{CB}}$ (figure 39) forbids the bound and free variables of the loop body from overlapping at all. This way then the loop expands into $\overline{\mathrm{loop}}$, the constraint given the corresponding clause (which is the same as the constraint given for seq) is preserved, preventing any variable capture.

5.1.4. Loops and the evaluator

Loops give us another scenario where the evaluator is undefined: instantaneous loops. Instantaneous loops will always reach a state where they contain a program fragment which matches ($\overline{\text{loop}}$ nothing **q**). Such a program has had the loop body terminate in the current instant, and there is no rule which can reduce this term. This term is not $\mathbf{p}^{\mathrm{D}}$, however, because it is not a complete program state. In addition, this program is not counted as $\vdash_{\mathrm{B}}$. This is because if such a program were to be counted as non-constructive then the definitions of non-constructive in Esterel would not cleanly match the definition of non-constructive in circuits. The Esterel compiler from Berry (2002) requires that instantaneous loops be eliminated statically before compilation, and therefore is not defined on such programs. Therefore I have chosen to make $\mathit{eval}^{\mathrm{E}}$ also not defined on such programs.

5.1.5. Leaving loops out of the proofs

There are two reasons I have left loops out of my proofs, both of which relate to difficulties in stating the theorems correctly. Firstly there is a subtle difference in how the circuit semantics and the evaluator handle instantaneous loops. The circuit semantics requires that all loops be checked statically to ensure that they can never be instantaneous. The evaluator, however, is undefined only on programs that trigger instantaneous loops dynamically. This means that the evaluator is defined on more programs than the circuit semantics.

The next issue is the issue of schizophrenia. The correct binding judgment ensures that the calculus never suffers from schizophrenia. However the circuit semantics requires that programs be transformed to eliminate possibly schizophrenic variables. The simplest of these is to fully duplicate every loop body, by transforming every (loop **p**) into (loop (seq **p** **p**)). This is not what is done in practice: in general parts of the program are selectively duplicated, using methods such as those given by Schneider and Wenz (2001) or chapter 12 of Berry (2002). This however means in practice that the loops that the circuit semantics operates on look different to those that the evaluator operates on, in a way that is difficult to formalize.

The handling of in the calculus loops is adapted from the COS (Potop-Butucaru 2002), which annotates loops with STOP and GO to determine if the loop can restart.

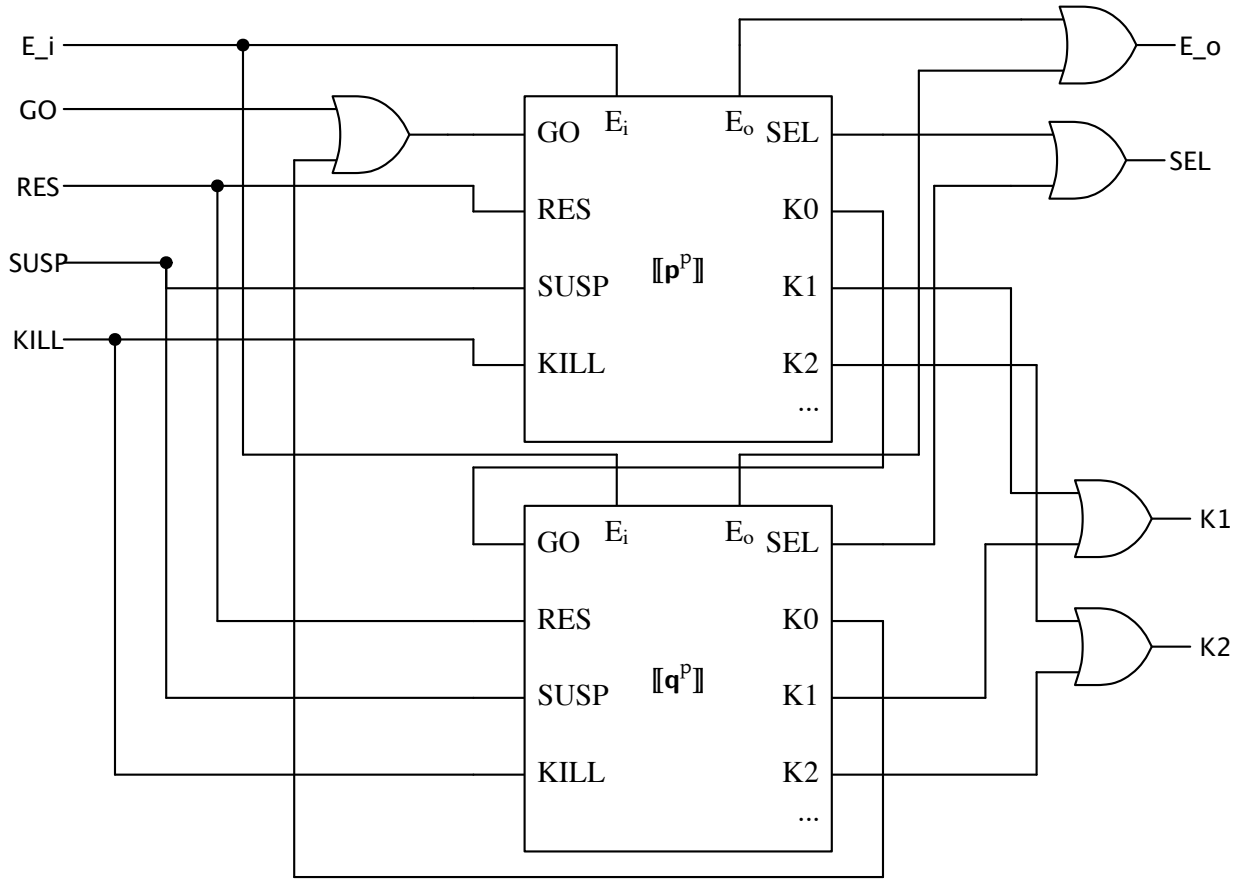


Figure 40: The compilation of $[\![(\overline{\text{loop}}\ \mathbf{p}^{\mathrm{p}}\ \mathbf{q}^{\mathrm{p}})]\!]$

Loops are handled by the circuit semantics by duplicating the loop body, in a manner described by Berry (2002). To see how this works, it is easiest to look at the compilation of $\overline{\text{loop}}$, seen in figure 40. This is essentially the compilation of seq, except that the output K0 wire is removed (and is therefore 0), and the K0 wire of $\mathbf{q}^{\mathrm{p}}$ is or'ed with the GO wire, restarting the whole loop when it terminates. From here we can define $[\![(\text{loop}\ \mathbf{p}^{\mathrm{p}})]\!] = [\![(\overline{\text{loop}}\ \mathbf{p}^{\mathrm{p}}\ \mathbf{p}^{\mathrm{p}})]\!]$. Note that this loop compilation assumes that the loop is never instantaneous.

5.2. Host language rules

To handle host language forms, θ and θ^{r} must be extended to accept shared and host language variables. Thus θ will also map $\mathbf{x}$s to their values, and $\mathbf{s}$s to a pair of a value and their current status:

$$\text{shared-status} ::= \text{old} \mid \text{new}$$

To understand these shared-statuses, observe the rules for shared variables:

[**shared**] $(\varrho\ \langle\theta^{\mathrm{r}}, \mathbf{A}\rangle.\ \mathbf{E}[(\text{shared } \mathbf{s} := \mathbf{e}\ \mathbf{p})]) \rightharpoonup^{\mathrm{E}} (\varrho\ \langle\theta^{\mathrm{r}}, \mathbf{A}\rangle.\ \mathbf{E}[(\varrho\ \langle\{\ \mathbf{s} \mapsto \langle \mathbf{n}\ , \text{old}\rangle\ \}, \text{WAIT}\rangle.\ \mathbf{p})])$
if $\mathit{FV}(\mathbf{e}) \subseteq \mathit{dom}(\theta^{\mathrm{r}})$, $\forall\ \mathbf{s} \in \mathit{FV}(\mathbf{e}).\ \mathbf{s} \notin \mathit{Can}^{\mathrm{sh}}_{\varrho}((\varrho\ \langle\theta^{\mathrm{r}}, \mathbf{A}\rangle\ \mathbf{E}[(\text{shared } \mathbf{s} := \mathbf{e}\ \mathbf{p})]), \cdot)$,
$\mathbf{n} = \mathit{eval}^{\mathrm{H}}(\mathbf{e}\ , \theta^{\mathrm{r}})$

[**set-old**] $(\varrho\ \langle\theta^{\mathrm{r}}, \text{GO}\rangle.\ \mathbf{E}[(\mathrel{+}= \mathbf{s}\ \mathbf{e})]) \rightharpoonup^{\mathrm{E}} (\varrho\ \langle(\theta^{\mathrm{r}} \leftarrow \{\ \mathbf{s} \mapsto \langle \mathit{eval}^{\mathrm{H}}(\mathbf{e}\ , \theta^{\mathrm{r}})\ , \text{new}\rangle\ \}), \text{GO}\rangle.\ \mathbf{E}[\text{nothing}])$
if $\theta^{\mathrm{r}}(\mathbf{s}) = \langle \_\ , \text{old}\rangle$, $\mathit{FV}(\mathbf{e}) \subseteq \mathit{dom}(\theta^{\mathrm{r}})$, $\forall\ \mathbf{s} \in \mathit{FV}(\mathbf{e}).\ \mathbf{s} \notin \mathit{Can}^{\mathrm{sh}}_{\varrho}((\varrho\ \langle\theta^{\mathrm{r}}, \text{GO}\rangle\ \mathbf{E}[(\mathrel{+}= \mathbf{s}\ \mathbf{e})]), \cdot)$

[**set-new**] $(\varrho\ \langle\theta^{\mathrm{r}}, \text{GO}\rangle.\ \mathbf{E}[(\mathrel{+}= \mathbf{s}\ \mathbf{e})]) \rightharpoonup^{\mathrm{E}} (\varrho\ \langle(\theta^{\mathrm{r}} \leftarrow \{\ \mathbf{s} \mapsto \langle \mathbf{n} + \mathit{eval}^{\mathrm{H}}(\mathbf{e}\ , \theta^{\mathrm{r}})\ , \text{new}\rangle\ \}), \text{GO}\rangle.\ \mathbf{E}[\text{nothing}])$
if $\mathit{FV}(\mathbf{e}) \subseteq \mathit{dom}(\theta^{\mathrm{r}})$, $\theta^{\mathrm{r}}(\mathbf{s}) = \langle \mathbf{n}\ , \text{new}\rangle$, $\forall\ \mathbf{s} \in \mathit{FV}(\mathbf{e}).\ \mathbf{s} \notin \mathit{Can}^{\mathrm{sh}}_{\varrho}((\varrho\ \langle\theta^{\mathrm{r}}, \text{GO}\rangle\ \mathbf{E}[(\mathrel{+}= \mathbf{s}\ \mathbf{e})]), \cdot)$

The first rule is analogous to the [**signal**] rule, converting a shared variable binder into a local environment with the control variable WAIT. However the rule [**shared**] must evaluate a host language expression to determine the default value for the shared variable. To do this it checks if every single shared and host language variable in the host language expression is bound in the local environment, and that all of the referenced shared variables cannot be written to, according to $\mathit{Can}^{\mathrm{sh}}_{\varrho}$. Only if this is true can the expression $\mathbf{e}$ be given to the host language evaluator $\mathit{eval}^{\mathrm{H}}$, which accepts the host language expression and the binding environment. The new environment initializes the shared variable's status to old, because the default value of the shared value acts as if it came from the previous instant and should only be used if the shared variable is not written to.

The [**set-old**] and [**set-new**] rules both update the value of a shared variable, using the same rules as [**shared**] to decide if the host language expression can be evaluated. The difference is in the actual updating. If the status is old, then the new value replaces the old value, and the status is set to new. If the status is already new, then the value gotten from evaluating $\mathbf{e}$ is combined by the associative and commutative operator for that variable. In this model the only operator is +, however in a real program the operator is given by the program itself. Like [**emit**], these rules may only fire when the control variable is GO, for a similar reason as to [**emit**].

Initializing and updating host language variables is simpler:

$$\begin{array}{ll}
[\textbf{var}] & (\varrho\,\langle\theta^{\mathrm{r}}, \mathbf{A}\rangle.\ \mathbf{E}[(\mathrm{var}\ \mathbf{x} := \mathbf{e}\ \mathbf{p})]) \rightharpoonup^{\mathrm{E}} (\varrho\,\langle\theta^{\mathrm{r}}, \mathbf{A}\rangle.\ \mathbf{E}[(\varrho\,\langle\{\ \mathbf{x} \mapsto \mathit{eval}^{\mathrm{H}}(\mathbf{e}, \theta^{\mathrm{r}})\ \}, \mathrm{WAIT}\rangle.\ \mathbf{p})]) \\
 & \quad \text{if } \mathit{FV}(\mathbf{e}) \subseteq \mathit{dom}(\theta^{\mathrm{r}}),\ \forall\, \mathbf{s} \in \mathit{FV}(\mathbf{e}).\ \mathbf{s} \notin \mathit{Can}^{\mathrm{sh}}_{\varrho}((\varrho\,\langle\theta^{\mathrm{r}}, \mathbf{A}\rangle\ \mathbf{E}[(\mathrm{var}\ \mathbf{x} := \mathbf{e}\ \mathbf{p})]), \cdot) \\
[\textbf{set-var}] & (\varrho\,\langle\theta^{\mathrm{r}}, \mathbf{A}\rangle.\ \mathbf{E}[(:= \mathbf{x}\ \mathbf{e})]) \rightharpoonup^{\mathrm{E}} (\varrho\,\langle(\theta^{\mathrm{r}} \leftarrow \{\ \mathbf{x} \mapsto \mathit{eval}^{\mathrm{H}}(\mathbf{e}, \theta^{\mathrm{r}})\ \}), \mathbf{A}\rangle.\ \mathbf{E}[\mathrm{nothing}]) \\
 & \quad \text{if } \mathbf{x} \in \mathit{dom}(\theta^{\mathrm{r}}),\ \mathit{FV}(\mathbf{e}) \subseteq \mathit{dom}(\theta^{\mathrm{r}}),\ \forall\, \mathbf{s} \in \mathit{FV}(\mathbf{e}).\ \mathbf{s} \notin \mathit{Can}^{\mathrm{sh}}_{\varrho}((\varrho\,\langle\theta^{\mathrm{r}}, \mathbf{A}\rangle\ \mathbf{E}[(:= \mathbf{x}\ \mathbf{e})]), \cdot)
\end{array}$$

The initialization of the host language variable via [**var**] is nearly identical to the initialization of shared variables, the only difference being the type of variable, and the lack of a status. Updating a host language variable is also akin to [**set-old**], except that because Esterel doesn't directly provide guarantees about concurrency with host language variables, the new value just replaces the old one in the environment.

We can condition on host language variables using the if!0 form:

$$\begin{array}{ll}
[\textbf{if-true}] & (\varrho\,\langle\theta^{\mathrm{r}}, \mathrm{GO}\rangle.\ \mathbf{E}[(\mathrm{if!0}\ \mathbf{x}\ \mathbf{p}\ \mathbf{q})]) \rightharpoonup^{\mathrm{E}} (\varrho\,\langle\theta^{\mathrm{r}}, \mathrm{GO}\rangle.\ \mathbf{E}[\mathbf{p}]) \\
 & \quad \text{if } \mathbf{x} \in \mathit{dom}(\theta^{\mathrm{r}}),\ \theta^{\mathrm{r}}(\mathbf{x}) \neq 0 \\
[\textbf{if-false}] & (\varrho\,\langle\theta^{\mathrm{r}}, \mathrm{GO}\rangle.\ \mathbf{E}[(\mathrm{if!0}\ \mathbf{x}\ \mathbf{p}\ \mathbf{q})]) \rightharpoonup^{\mathrm{E}} (\varrho\,\langle\theta^{\mathrm{r}}, \mathrm{GO}\rangle.\ \mathbf{E}[\mathbf{q}])\ \text{if } \theta^{\mathrm{r}}(\mathbf{x}) = 0
\end{array}$$

Which, for this model, behaves like C's if, treating 0 as false and everything else as true. These rules requires the control variable to be GO as well, as picking a branch of the if!0 might break existing causality cycles, and therefore change the constructivity of the program.

5.2.1. Host language and Can

We must add several clauses to *Can* to handle shared variables and host language expressions(figure 41). The analysis of the shared from is like that of the signal form, except that there is no special case for when the shared variable cannot be written to. Because Esterel does not make control flow decisions based on the writability of shared variable, there is no need for the extra step. Writing to a shared variable behaves akin to emitting a signal: the return code is 0 and the variable is added to the sh set.

The last three clauses handle host language variables. No special analysis is done for these forms as Esterel does not link them into its concurrency mechanism.

$$
\begin{array}{ll}
\mathit{Can}(\text{(shared } \mathbf{s} := \mathbf{e}\ \mathbf{p}),\ \theta) & = \{\ \mathrm{S} = \mathit{Can}^{\mathrm{S}}(\mathbf{p}, \theta),\ \mathrm{K} = \mathit{Can}^{\mathrm{K}}(\mathbf{p}, \theta),\ \mathrm{sh} = \mathit{Can}^{\mathrm{sh}}(\mathbf{p}, \theta) \setminus \{\ \mathbf{s}\ \}\ \} \\
\mathit{Can}(\text{(+= } \mathbf{s}\ \mathbf{e}),\ \theta) & = \{\ \mathrm{S} = \emptyset,\ \mathrm{K} = \{\ 0\ \},\ \mathrm{sh} = \{\ \mathbf{s}\ \}\ \} \\
\mathit{Can}(\text{(var } \mathbf{x} := \mathbf{e}\ \mathbf{p}),\ \theta) & = \mathit{Can}(\mathbf{p}, \theta) \\
\mathit{Can}(\text{(:= } \mathbf{x}\ \mathbf{e}),\ \theta) & = \{\ \mathrm{S} = \emptyset,\ \mathrm{K} = \{\ 0\ \},\ \mathrm{sh} = \emptyset\ \} \\
\mathit{Can}(\text{(if!0 } \mathbf{x}\ \mathbf{p}\ \mathbf{q}),\ \theta) & = \{\ \mathrm{S} = \mathit{Can}^{\mathrm{S}}(\mathbf{p}, \theta) \cup \mathit{Can}^{\mathrm{S}}(\mathbf{q}, \theta), \\
 & \quad\ \mathrm{K} = \mathit{Can}^{\mathrm{K}}(\mathbf{p}, \theta) \cup \mathit{Can}^{\mathrm{K}}(\mathbf{q}, \theta), \\
 & \quad\ \mathrm{sh} = \mathit{Can}^{\mathrm{sh}}(\mathbf{p}, \theta) \cup \mathit{Can}^{\mathrm{sh}}(\mathbf{q}, \theta)\ \}
\end{array}
$$

Figure 41: Can and the Host Language

$$
\frac{}{\theta^{\mathrm{r}};\ \mathrm{WAIT};\ \mathbf{E} \vdash_{\mathrm{B}} (\text{+= } \mathbf{s}\ \mathbf{e})}\ [\text{set-shared-wait}]
$$

$$
\frac{\theta^{\mathrm{r}};\ \mathbf{A};\ \mathbf{E}[(\text{shared } \mathbf{s} := \mathbf{e}\ \mathbf{p})] \vdash_{\mathrm{e}} \mathbf{e}}{\theta^{\mathrm{r}};\ \mathbf{A};\ \mathbf{E} \vdash_{\mathrm{B}} (\text{shared } \mathbf{s} := \mathbf{e}\ \mathbf{p})}\ [\text{shared}]
\qquad
\frac{\theta^{\mathrm{r}};\ \mathbf{A};\ \mathbf{E}[(\text{+= } \mathbf{s}\ \mathbf{e})] \vdash_{\mathrm{e}} \mathbf{e}}{\theta^{\mathrm{r}};\ \mathbf{A};\ \mathbf{E} \vdash_{\mathrm{B}} (\text{+= } \mathbf{s}\ \mathbf{e})}\ [\text{set-shared}]
$$

$$
\frac{\theta^{\mathrm{r}};\ \mathbf{A};\ \mathbf{E}[(\text{var } \mathbf{x} := \mathbf{e}\ \mathbf{p})] \vdash_{\mathrm{e}} \mathbf{e}}{\theta^{\mathrm{r}};\ \mathbf{A};\ \mathbf{E} \vdash_{\mathrm{B}} (\text{var } \mathbf{x} := \mathbf{e}\ \mathbf{p})}\ [\text{var}]
\qquad
\frac{\theta^{\mathrm{r}};\ \mathbf{A};\ \mathbf{E}[(\text{:= } \mathbf{x}\ \mathbf{e})] \vdash_{\mathrm{e}} \mathbf{e}}{\theta^{\mathrm{r}};\ \mathbf{A};\ \mathbf{E} \vdash_{\mathrm{B}} (\text{:= } \mathbf{x}\ \mathbf{e})}\ [\text{set-seq}]
$$

Figure 42: The blocked judgment on terms using the host language

$$
\frac{\mathbf{s} \in \mathit{FV}(\mathbf{e}) \qquad \mathbf{s} \in \mathit{Can}^{\mathrm{sh}}_{Q}((_{Q}\ \langle \theta^{\mathrm{r}}, \mathbf{A} \rangle.\ \mathbf{p}), \{\})}{\theta^{\mathrm{r}};\ \mathbf{A};\ \mathbf{p} \vdash_{\mathrm{e}} \mathbf{e}}\ [\text{not ready}]
$$

Figure 43: The blocked judgment host language expressions

5.2.2. Host language and Blocked

The $\vdash_{\mathrm{B}}$ relation must be extended to forms that refer to the host language in figure 42. They are all base cases, and are either blocked because a write to a shared variable may not be performed due to the control variable ([**set-shared-wait**]), or because a host language expression is blocked. The blocked judgment for a host language expression (figure 43) says that the expression may not be evaluated if at least one of the shared variables that the expression references might still be written to by the full program.

$$\frac{\vdash_{\text{CB}} \mathbf{p}}{\vdash_{\text{CB}} (\text{shared } \mathbf{s} := \mathbf{e}\ \mathbf{p})}\,[\text{shared}] \quad \frac{}{\vdash_{\text{CB}} (\mathrel{+}= \mathbf{s}\ \mathbf{e})}\,[<=]$$

$$\frac{\vdash_{\text{CB}} \mathbf{p}}{\vdash_{\text{CB}} (\text{var } \mathbf{x} := \mathbf{e}\ \mathbf{p})}\,[\text{var}] \quad \frac{}{\vdash_{\text{CB}} (:= \mathbf{x}\ \mathbf{e})}\,[:=]$$

$$\frac{\vdash_{\text{CB}} \mathbf{p} \qquad \vdash_{\text{CB}} \mathbf{q}}{\vdash_{\text{CB}} (\text{if!0 } \mathbf{x}\ \mathbf{p}\ \mathbf{q})}\,[\text{if0}]$$

Figure 44: Correct Binding and the host language

5.2.3. Host language and Correct Binding

The extensions to correct-binding for the host language (figure 44) only require that all subforms have correct binding.

5.2.4. Leaving the host language out of the proofs

There are two reasons I have left the host language out of the proofs. The primary one is that the ground truth semantics I am using does not include these forms, thus writing proofs about them would involve extending that semantics in a non-trivial way. The second reason is that the host language forms reuse many of the mechanisms from the pure language. Therefore the proofs about the pure section of the language give some confidence for the correctness of the host language part.

The handling of shared variables is adapted from the COS, which uses the same extension to *Can*.

5.3. Future Instants

While both the calculus and the evaluator only handle single instants, we can still describe multiple instants. This is done via the inter-instant transition metafunction $\mathcal{E}$. This relies on the notion of a complete term:

$$\mathbf{p}^{\text{C}} ::= \mathbf{p}^{\text{D}} \mid (\varrho\ \langle \theta, \text{GO} \rangle.\ \mathbf{p}^{\text{D}})$$

Definition: $\mathcal{E}(\mathbf{p}^{\mathrm{C}})$

$$
\begin{array}{ll}
\mathcal{E} : \mathbf{p}^{\mathrm{C}} \to \mathbf{p} & \\
\mathcal{E}((\varrho\ \langle \theta^{\mathrm{r}}, \mathrm{GO} \rangle.\ \mathbf{p})) & = (\varrho\ \langle \lfloor \theta^{\mathrm{r}} \rfloor, \mathrm{WAIT} \rangle.\ \mathcal{E}(\mathbf{p})) \\
\mathcal{E}(\mathsf{pause}) & = \mathsf{nothing} \\
\mathcal{E}(\mathsf{nothing}) & = \mathsf{nothing} \\
\mathcal{E}((\overline{\mathsf{loop}}\ \mathbf{p}\ \mathbf{q})) & = (\mathsf{seq}\ \mathcal{E}(\mathbf{p})\ (\mathsf{loop}\ \mathbf{q})) \\
\mathcal{E}((\mathsf{seq}\ \mathbf{p}\ \mathbf{q})) & = (\mathsf{seq}\ \mathcal{E}(\mathbf{p})\ \mathbf{q}) \\
\mathcal{E}((\mathsf{par}\ \mathbf{p}\ \mathbf{q})) & = (\mathsf{par}\ \mathcal{E}(\mathbf{p})\ \mathcal{E}(\mathbf{q})) \\
\mathcal{E}((\mathsf{suspend}\ \mathbf{p}\ \mathbf{S})) & = (\mathsf{suspend}\ (\mathsf{seq}\ (\mathsf{if}\ \mathbf{S}\ \mathsf{pause}\ \mathsf{nothing})\ \mathcal{E}(\mathbf{p}))\ \mathbf{S}) \\
\mathcal{E}((\mathsf{trap}\ \mathbf{p})) & = (\mathsf{trap}\ \mathcal{E}(\mathbf{p})) \\
\mathcal{E}((\mathsf{exit}\ \mathbf{n})) & = (\mathsf{exit}\ \mathbf{n})
\end{array}
$$

Figure 45: The inter-instant transition function

which is a term which is either constructive or $\mathbf{p}^{\mathrm{D}}$. Such a term is either a program or a fragment of a program which has finished evaluating for the current instant. The $\mathcal{E}$ metafunction turns such a term into a term which will begin execution at the start of the next instant. The function is defined in figure 45.

This function walks down the program updating it so that it will unpause, replacing every pause which is an evaluation context with respect to the given term with nothing, allowing the program to resume from those points. In addition, the first clause uses the metafunction $\lfloor \cdot \rfloor$, which takes an environment and sets the status of every signal to $\bot$, and the status of every shared variable to old.

The metafunction $\mathcal{E}$ also modifies the forms $\overline{\mathsf{loop}}$ and suspend. A $\overline{\mathsf{loop}}$ is replaced with the traditional unfolding of a loop, because in the next instant the loop is allowed to restart. The suspend transformation is more complex. We want a term which entered a suspend to pause in that suspend if the given signal is 1 in the next instant. Therefore we transform an suspend to do exactly that: if the signal is 1 then pause, otherwise do nothing and resume executing the suspend's body.

5.3.1. Leaving instants out of the proofs

I have left instants out of the proofs because of how *Can* is defined in the calculus. The version of *Can* I give here assumes that the top of the program is where execution will occur, rather than execution starting from some pause. However I postulate that the calculus should still be correct for multi-instant execution. In addition to the tests, the inter-instant translation function $\mathcal{E}$ is nearly identical to the same function from Berry (2002)[1] which as been proven correct[2] in Coq (Berry and Rieg 2019), but with extensions to handle $\overline{\text{loop}}$ and ϱ.

5.4. Evidence via Testing

I have evidence the theorems I have proven should hold for the extensions in this chapter. This evidence comes in the form of random tests. To do this, I use the following

- **Redex COS model** I built a Redex (Felleisen et al. 2009) model of the COS semantics. The semantics is a rule-for-rule translation of the COS semantics from Potop-Butucaru et al. (2007), aside from some minor syntax differences. This provides an executable model of the COS semantics.

- **Redex calculus model** I have also build a Redex model of the calculus. This defines two relations: the core relation of the calculus $\rightharpoonup^{\text{E}}$, and a new relation $\rightharpoondown$ which gives an evaluation strategy for $\rightharpoonup^{\text{E}}$. The $\rightharpoondown$ relation and the $\mathcal{E}$ function is used to define a multi-instant evaluator for Esterel. This evaluator checks at every reduction step that the step taken by $\rightharpoondown$ is also in $\rightharpoonup^{\text{E}}$. The relation $\rightharpoondown$ is given in the appendix A.

- **Racket Frontend** The actual execution of this Redex model of the calculus is embedded into Racket, and may use Racket as its host language in addition to the numeric language the calculus comes equipped with. There is also a Racket frontend compiler which compiles Full Esterel into the Kernel used by the Redex model with the extensions to use Racket as its host language.

[1] Section 8.3, page 89 of the current draft

[2] Specifically, it is proven that, up to bisimilarity, a program passed through $\mathcal{E}$ under the Constructive Semantics remains the same program with respect to the State Semantics. See Theorem 5 of Berry and Rieg (2019).

- **Redex/Hiphop.js bridge** HipHop.js is an Esterel implementation embedded into Javascript. We built a library that can translate Redex expressions into Hiphop.js (Berry et al. 2011) programs and then evaluate them.[3] There is also a compiler form a subset of Hiphop.js to the Redex model of the calculus, allowing many of the Hiphop.js tests be run directly against the calculus. This translator does not accept all Hiphop.js programs, because Hiphop.js programs embed JavaScript code which the Redex model cannot evaluate.

- **Redex/Esterel v5 bridge** We also built a translator from Redex terms to Esterel v5 programs.

- **Redex circuit compiler** Finally I have built a compiler from pure Esterel (with loops) to circuits, which runs on top of the circuit solver.

I have run 1537900 random tests which on Full Esterel programs with loops. These tests check that the Hiphop.js, Esterel v5, the COS, the calculus, and the circuit compiler agree on the result of running programs for multiple instants.[4] These tests are to provide evidence for consistency and adequacy, not just against the circuit semantics but against real implementations as well. The real implementations are important because they accept Esterel terms that use host language expressions, which the circuit compiler does not. Therefore these tests in particular give evidence that adequacy holds in the presence of Full Esterel. Whenever the generated program contains host language variables or is not guaranteed to have no possible instantaneous loops the circuit compiler is skipped.

In addition I have run 11500 random tests which generate a random pure program (with loops), and apply all rules from the calculus (specifically from $\longrightarrow^{E}$, the compatible closure of $\rightharpoonup^{E}$), and then check that the circuits are equal using the Circuitous library. These tests provide evidence for soundness, and especially for the soundness with loops.

Together these tests took approximately six CPU weeks. The logs of these test cases and all of the code discussed above is stored at https://github.com/florence/esterel-calculus.

[3] Special thanks to Jesse Tov for helping out with this.
[4] Each test runs for a random number of instants.

CHAPTER 6

Related Work

This section gives existing work related to the Constructive Calculus. Most of this work is other Esterel semantics. In addition there are some related works on the development of calculi in general.

6.1. Other Esterel semantics

To show why a new semantics for Esterel contributes to the existing work on Esterel, this section covers some existing semantics, how they related to properties the calculus captures, and how they capture the notion of constructiveness. Many of the semantics in this section are given the label "Constructive". This is because earlier semantics for Esterel captured a slightly different language which accepted more programs. Those semantics are called "logical", such as the original semantics, given in Berry and Cosserat (1992). While some more recent work such as Tardieu (2007) use a logical semantics, they mostly have out of favor, and no modern Esterel implementation uses them. As Logical Esterel is a slightly different language than Constructive Esterel, I will not discuss logical semantics futher here, but rather focus on the constructive semantics, of which the Constructive Calculus is one.

6.1.1. Constructive Behavioral and State Behavioral Semantics

This section actually covers two semantics, the Constructive Behavioral Semantics (CBS), and the State Behavioral Semantics (SBS), as they are the same in all but one respect. Both are given in Berry (2002).

These semantics largely consist of two metafunctions: Must and Can. Must determines what code must be reached by execution, and therefore what signals must be emitted. Can determines what code might be reached, walking the causality graph and pruning branches that cannot be reached, and is used to set signals to absent. Constructive

programs are ones in which all signals in the program either *Must* or Cannot be emitted, and non-Constructive programs have signals which fall into neither set.

CBS tracks the state of pauses via an administrative reduction, reducing the program to one which will resume from the appropriate points. SBS does the same, but instead of rewriting the program it decorates pause statements to see which ones were reached, and uses these decorations to figure out where to resume. In essence, these reductions propagate the information from *Must* and *Can*.

These semantics both give a syntactic evaluator for Esterel: that is they give a function that gives the final result of several instants of execution. They are not equational theories. These semantics, as given in Berry (2002), only handle Pure Esterel.

6.1.2. Constructive Operational Semantics

The Constructive Operational Semantics (COS), in some sense, provides a bridge between the SBS and an actual implementation of Esterel. It replaces the *Must* metafunction with a reduction relation (which also doubles as the administrative reduction relation from the SBS). The core idea here is that the reduction relation represents running the program, and if something runs then it must happen.

In the COS, non-constructive programs are ones which get stuck during execution, e.g. programs that cannot reduce further and are not in a fully executed state. In general, this is because the reduction cannot make progress without executing some conditional, but the value of the signal being conditioned on is unknown and cannot be set to absent. This approach inspired how the Constructive Calculus handles constructiveness.

Like the CBS and CSS, the COS gives defines a syntactic evaluator for Esterel, and is not an equational theory. The COS is defined on all of Kernel Esterel.

6.1.3. Circuit Semantics

The circuit semantics gives meaning to Esterel programs by transforming them into circuits. This works by transforming the control flow part of Esterel programs into a dataflow problem where the data is encoded as power flowing down a wire. This is then combined with the original dataflow (e.g. signals) to give a full circuit. In essence, the circuit semantics treats the causality graph as a dataflow problem in the domain of circuits. A pauses is encoded as a register which passes on whether or not control reached a given point onto a future instant.

There are multiple circuit semantics. Potop-Butucaru (2002) splits the program into two circuits, the Surface, which handles the first instant a term is reached, and the Depth which handles future instants. Whether or not the Surface or Depth circuit is reached is controlled by the state of the registers within those terms.

The circuit semantics I prove the calculus equivalent to comes from Berry (2002), which is based on the original circuit compiler from Berry (1992). This compiler is discussed in depth in section 4.1.1.

In both semantics, the constructivity of Esterel programs is transformed into the constructivity of circuits (Shiple et al. 1996): an Esterel program is constructive on some inputs if and only if all wires in its circuit always settle to a single value in a bounded amount of time. Just as with causality graphs, circuits are non-constructive if some cycle in the circuit demands a value be settled on for some wire, and the value for that wire value depends on the state of the cycle.

The circuit semantics allow for local reasoning about equality between programs. It also provides an evaluator, through circuit simulation. In addition circuit semantics can be used to prove equivalences between programs, as we have a computable equality relation between circuits. However its reasoning is non-syntactic: the transformations done to a circuit may not result in a new circuit that can be transformed back into an Esterel program, and even when they can be, the reasoning used to explain why the circuit can be transformed in that way might not map cleanly back to Esterel. Therefor, it is not an equational theory. The circuit semantics in Berry (2002) is defined on only pure Esterel. The circuit semantics in Potop-Butucaru (2002) is extended to handle a host language.

6.1.4. The Axiomatic Semantics

Tini (2001) gives two semantics. The first is a labeled transition system that gives an evaluator for Esterel programs, and the second is a series of logical axioms which give an equality relation for Esterel programs, which they call the Axiomatic Semantics. Of all the semantics presented so far, these axioms are the closest to the goal that I have, as it is an equational theory. However it is built from fundamentally different techniques,[1] and it is not adequate to define an evaluator for Esterel. This is because it cannot reason about emits, as it lacks the control variable my calculus adds. However it is much stronger in other respects: in fact it is complete modulo bisimilarity on constructive programs. Adding the axioms of the Axiomatic Semantics to the Construtive Calculus would result in a much more powerful reasoning framework. The Axiomatic Semantics only handles Pure Esterel.

6.1.5. The Color Semantics

Berry and Rieg (2019) gives a a microstep semantics which I call here the Color Semantics.[2] It replaces both Must and Can with colors that propagate throughout the program, mimicking how 1 and 0 propagate through circuits. The control variables of the constructive calculus are based off this. The Color semantics is computationally adequate, and is not an equational theory. The Color Semantics handles only Pure Esterel.

6.1.6. Quartz

Quartz (Schneider 2001) is a variant of Esterel embedded into the interactive theorem prover HOL (Gordon and Melham 1993). Quartz is defined by a transformation to a set of control flow predicates and guarded commands. The full semantics is given by the conjunction of the logical formula these define. This allows properties of Quartz programs to be verified by both model checking and theorem proving using HOL. The generation of the guarded commands used for defining dataflow requires knowledge of the precondition for that command, which requires knowledge about the context. Like the circuit semantics, this means it is not an equational theory, but it still can prove equivalences between

[1] For instance, their notion of equality is based on bisimulation, whereas mine is based on contextual equivalence.

[2] This is called the "Microstep semantics" by Berry and Rieg (2019). I use a different name here to avoid confusion with other microstep semantics like the COS.

program using the underlying model it maps programs to. It is adequate, and in fact can be used for verified code generation. Quartz handles host language data, and also extends Kernel Esterel with forms such as delayed emission and non-deterministic choice.

6.2. Circuits

The handling of cyclic circuits is derived from the seminal work of Malik (1994). It uses the extensions for registers given by Shiple et al. (1996), which have been proven correct by Mendler et al. (2012). The notion of constructivity used here is what Shiple et al. (1996) call strong constructivity, as Malik's original definition of constructivity only demanded that interface wires be non-$\bot$, and allowed internal wires to take on any value.

6.3. Calculi

The Constructive Calculus for Esterel draws heavily from the State Calculus (Felleisen and Hieb 1992)—specifically in the usage of local maps and evaluation contexts (Felleisen and Friedman 1986) to track the state locally.

This calculus is the second draft, the first introduced in Florence et al. (2019). That calculus, however, was not constructive, as it allowed for local rewrites which could bypass signals who's value was not yet known. The local control variables A was introduced to solve this issue.

CHAPTER 7

Future Work

The constructive calculus presented here is the first calculus for Esterel which captures both Constructivity and which is Adequate. However, research is never complete—the proofs do not cover the entire language, the handling of emit makes the calculus somewhat weak, and there are stronger compilation guarantees one might wish to have proven. This section is meant to give a small starting point for any ambitious researcher who wants to tackle these problems.

7.1. Extending proofs to multiple instants, and guarding compilation

Future work may wish to extend the proofs for the Consistency, Soundness, and Adequacy of the Constructive Calculus to multiple instants. This should be possible with a tweak to the compilation of terms.

This tweak enforces an assumption that the compilation function makes: that GO and SEL are mutually exclusive. The circuits generated by $[\![\cdot]\!]$ do not behave properly if this condition is not met: in essence it is undefined behavior. This undefined behavior ruins many equalities that should hold, as having both GO and SEL true simultaneously can expose details of the internals of a term that are not observable in Esterel, but are observable in the circuit. Consider the equation:

```
(signal S1                                          (signal S1
  (seq                                                (seq pause (emit S1)))
   (if S1 (emit S2) nothing)    ≃^E
   (seq pause
        (emit S1))))
```

These two programs should be $\simeq^E$, as the signal S1 can never be emitted in the same instant in which it is conditioned on. However consider their circuit compilations: in the first program there will be a wire $S2 = GO \wedge S1$. However in the compilation of the second term there will be no S2 wire, therefore it will be taken to be 0. If, in the second cycle,

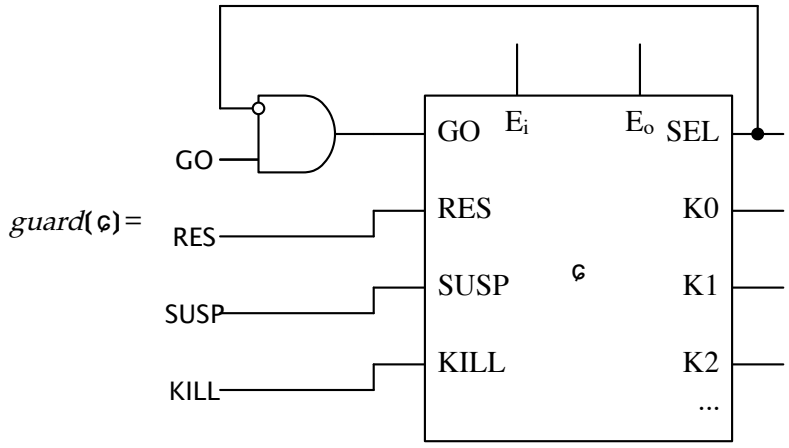


Figure 46: Guarding the top of a circuit to avoid protocol violations

GO is 1, (making it 1 at the same time as SEL) the wire S2 will get a 1, as both GO and S1 will be 1. But this differs from the second program! This means that the two circuits are not $\simeq^C$, which violates soundness.

No Esterel context would ever produce a violation like this, as this only occurs when the outer circuit context violates the protocol that GO and SEL are mutually exclusive. Therefore we can fix this by wrapping any compilation in a guard which, if this condition is violated, forces the circuit to have consistent behavior. Such a wrapper is given in figure 46. This guard suppresses GO if the protocol is violated, preventing the circuit from behavior from changing.

From here we can modify the statement of soundness to:

CONJECTURE 25 (SOUNDNESS OVER MULTIPLE INSTANTS)**.**

For all $\mathbf{p}^{\mathrm{P}}$ *and* $\mathbf{q}^{\mathrm{P}}$, *if* $\vdash_{\mathrm{CB}} \mathbf{p}^{\mathrm{P}}$ *and* $\mathbf{p}^{\mathrm{P}} \equiv^{\mathrm{E}} \mathbf{q}^{\mathrm{P}}$, *then* $\mathit{guard}(\llbracket \mathbf{p}^{\mathrm{P}} \rrbracket) \simeq^C \mathit{guard}(\llbracket \mathbf{q}^{\mathrm{P}} \rrbracket)$

Which removes the requirement that the SEL be 0, but adds in the guard. One should also prove that

CONJECTURE 26 (GUARD IS CORRECT)**.**

For all $\mathbf{p}^{\mathrm{P}}$, *if* $\llbracket \mathbf{p}^{\mathrm{P}} \rrbracket(\mathrm{GO}) \wedge \llbracket \mathbf{p}^{\mathrm{P}} \rrbracket(\mathrm{SEL}) \simeq 0$ *then* $\llbracket \mathbf{p}^{\mathrm{P}} \rrbracket \simeq^C \mathit{guard}(\llbracket \mathbf{p}^{\mathrm{P}} \rrbracket)$

Which shows that the guard never changes the program behavior if the protocol is never violated.

I believe that this is the only guard necessary. This belief comes from the proofs I have done so far: Other than proofs involving *Can*, the mutual exclusion of GO and SEL is the strongest precondition needed.[1]

The RES and SUSP wires do not need a similar guard procedure, because of lemma 76 (Activation Condition). While their is a similar protocol for them (e.g. GO and RES are mutually exclusive), this protocol only matters while SEL is 1, as RES and SUSP only have an effect on program resumption. Therefore the guard procedure above is enough to protect against errant uses of these wires by a (non-Esterel) outer context.

I suspect that the KILL wire should not need a guard either. This is because of the changes to the compilation of par from section 4.1.1. These changes remove the reliance on the protocol that return codes above 1 trigger the KILL wire, and therefore no protocol, and no guard, should be needed.

Adequacy must also be extended in a similar manner. However the new statement of Adequacy must be extended to involve the inter-instant translation function $\mathcal{E}$.

7.2. Removing θ from ρ

I suspect that there is a variant of my calculus which is both stronger (in the sense that it can prove more things equal) and does not require the θ^{r} portion of the environment. The idea behind is this that a θ^{r} can always be removed by running the existing [**emit**] and [**signal**] rules backwards, so why add it in the first place? Specifically, I believe that the [**emit**] and [**is-present**] rules can be replaced with:

[**emit**] $\quad \mathbf{E}[(\text{emit } \mathbf{S})] \rightharpoonup^{\mathrm{E}} (\text{par } (\text{emit } \mathbf{S})\ \mathbf{E}[\text{nothing}])$
[**is-present**] $(\varrho\ \mathrm{GO}.\ \mathbf{E}[(\text{if } \mathbf{S}\ \mathbf{p}\ \mathbf{q})]) \rightharpoonup^{\mathrm{E}} (\varrho\ \mathrm{GO}.\ \mathbf{E}[\mathbf{p}])$ if $\mathbf{E}'[(\text{emit } \mathbf{S})] = \mathbf{E}[(\text{if } \mathbf{S}\ \mathbf{p}\ \mathbf{q})]$, *for some* $\mathbf{E}'$

In this new system the [**is-present**] rule says that we may take the then branch of an if when the environment is GO and there is an emit for that signal in a relative evaluation context. The correctness of this rule can be validated by

[1] As we always take SEL to be 0, this condition is given by the existing preconditions on soundness and adequacy.

the current calculus (modulo the different environment shape), because the emit could be run, putting $\{ \mathbf{S} \mapsto 1 \}$ in the environment, and then the rule old [**is-present**] rule could take over.

The [**emit**] rule is where the extra power comes in. It lets us reshuffle emits arbitrarily within evaluation contexts. This would let us, for example, lift an emit out of a seq. We could prove equations like $(\text{par } (\text{emit } \mathbf{S})\ \mathbf{q}) \equiv^{\text{E}} (\text{seq } (\text{emit } \mathbf{S})\ \mathbf{q})$, and conjecture 5 (Lift Signal Emission (not provable)) which was discussed in section 3.4. The new [**emit**] rule cannot be proven by the current calculus, because in enables reasoning about emits when there is no GO, and when the binding form is not in an evaluation context with respect to the emit.

This new rule would require changing the formalization of lemma 57 (Strongly Canonicalizing), as the new [**emit**] rule could always execute, using $\mathbf{E} = \bigcirc$. However this does not seem like it would make a similar proof impossible, using a different formulation of the $\longrightarrow^{\text{S}}$ and $\longrightarrow^{\text{R}}$ relations. Or perhaps this could be solved by enforcing that the evaluation context never be empty. Both paths would be worth exploring.

The definition of *Can* would also need to change in this variant of the calculus. The *Can* function would likely still need an θ argument, therefore *Can* will need to add 1s to θ somehow. This could likely be done by replicating the relative evaluation context reasoning from the [**is-present**] rule, but it is not clear how this would work.

7.3. Fully Abstract Compilation

A future semanticist may wish to prove that the Esterel compiler (augmented with guard) is fully abstract. I believe that the Constructive Calculus gives the tools to do this. Specifically, the theorem to prove would be:

Conjecture 27 (Fully Abstract Compilation)**.**

For all $\mathbf{p}^{\text{P}}$ *and* $\mathbf{q}^{\text{P}}$, $\mathbf{p}^{\text{P}} \simeq^{E} \mathbf{q}^{\text{P}}$ *if and only if* $guard(\llbracket \mathbf{p}^{\text{P}} \rrbracket) \simeq^{C} guard(\llbracket \mathbf{p}^{\text{P}} \rrbracket)$

The reasons that such a proof may be within reach follows from the following chain of reasoning. First, the definition of $\simeq^{C}$ used here is defined by analyzing all of the possible inputs to the circuits. Second, the inputs to the circuits can be simulated using Esterel contexts. For example, if we have one input signal SI, we can simulate all inputs on it

using the contexts (signal SI ◯), (signal SI (par (emit SI) ◯)), and (signal SI (par (if SI (emit SI) nothing) ◯)), which correspond to 0, 1, and $\perp$ respectively. Third, we know that the evaluator given by the Constructive Calculus is equivalent to the circuit evaluator by theorem 30 (Computational Adequacy). Therefore if the contexts which simulate the inputs to the circuit are sufficient to decide $\simeq^E$, it must be that the notions of contextual equivalence between Esterel and Circuits is the same. I believe that these contexts, plus some which change **A**, are enough to decide $\simeq^E$. Formally, let these be defined as:

Definition: *input-contexts*(𝕊)

$$
\begin{aligned}
\textit{input-contexts}(\varnothing) &= \{\ (\varrho\ \langle\{\}, \mathrm{GO}\rangle.\ \bigcirc)\ ,\ (\varrho\ \langle\{\}, \mathrm{WAIT}\rangle.\ \bigcirc)\ \} \\
\textit{input-contexts}(\{\ \mathbf{S}\ \} \cup \mathbb{S}) &= \{\ (\text{signal}\ \mathbf{S}\ \mathbf{C}) \mid \mathbf{C} \in \textit{input-contexts}(\mathbb{S})\ \} \\
&\quad \cup \\
&\quad \{\ (\text{signal}\ \mathbf{S}\ (\text{par}\ (\text{emit}\ \mathbf{S})\ \mathbf{C})) \mid \mathbf{C} \in \textit{input-contexts}(\mathbb{S})\ \} \\
&\quad \cup \\
&\quad \{\ (\text{signal}\ \mathbf{S}\ (\text{par}\ (\text{if}\ \mathbf{S}\ (\text{emit}\ \mathbf{S})\ \text{nothing})\ \mathbf{C})) \mid \mathbf{C} \in \textit{input-contexts}(\mathbb{S})\ \}
\end{aligned}
$$

If one could prove:

Conjecture 28 (Signals decide contextual equivalence)**.**

For all $\mathbf{p}^{\mathrm{p}}$ *and* $\mathbf{q}^{\mathrm{p}}$, $\mathbf{p}^{\mathrm{p}} \simeq^E \mathbf{q}^{\mathrm{p}}$ *if and only if for all* $\mathbf{C} \in \textit{input-contexts}(\mathit{FV}(\mathbf{p}^{\mathrm{p}}) \cup \mathit{FV}(\mathbf{q}^{\mathrm{p}}))$, $\mathbf{C}[\mathbf{p}^{\mathrm{p}}] \equiv^{\mathrm{E}} \mathbf{C}[\mathbf{q}^{\mathrm{p}}]$

then the argument laid out here should be enough to complete a proof of fully abstract compilation. I do not know how this proof would proceed, however my intuition is that if there exists some context which shows that two terms are not $\simeq^E$, then one could inductively walk that context, and build a context which is in the input-contexts and which also shows that the two terms are not equivalent. Or perhaps the proof of Böhm's Theorem for the λ-calculus could provide inspiration.

Bibliography

H. Barendregt. The Lambda Calculus: its Syntax and Semantics. North Holland, 1984.

Albert Benveniste, Paul Caspi, Stephen A. Edwards, Nicolas Halbwachs, Paul Le Guernic, and Robert de Simone. The Synchronous Languages 12 Years Later. *Proceedings of the IEEE* 91(1), 2002.

Gérard Berry. Esterel on Hardware. *Philosophical Transactions Royal Society of London* 339, 1992.

Gérard Berry. The Constructive Semantics of Pure Esterel (Draft Version 3). 2002.

Gérard Berry, Amar Bouali, Xavier Fornari, Emmanuel Ledinot, Eric Nassor, and Robert de Simone. ESTEREL: A Formal Method Applied to Avionic Software Development. *Science of Computer Programming* 36(1), pp. 5–25, 2000.

Gérard Berry and Laurent Cosserat. The ESTEREL Synchronous Programming Language and its Mathematical Semantics. In *Proc. International Conference on Concurrency*, 1992.

Gérard Berry, Cyprien Nicolas, and Manuel Serrano. HipHop: A Synchronous Reactive Extension for Hop. In *Proc. PLASTIC*, 2011.

Gérard Berry and Lionel Rieg. Towards Coq-verified Esterel Semantics and Compiling. https://arxiv.org/abs/1909.12582v1, 2019.

Matthias Felleisen, Robert Bruce Findler, and Matthew Flatt. Semantics Engineering with PLT Redex. MIT Press, 2009.

Matthias Felleisen and Daniel P. Friedman. Control Operators, the SECD-machine, and the λ-calculus. In *Proc. Conference on Formal Descriptions of Programming Concepts Part III*, 1986.

Matthias Felleisen and Robert Hieb. The Revised Report on the Syntactic Theories of Sequential Control and State. *Theoretical Computer Science* 103(2), pp. 235–271, 1992.

Matthew Flatt and Matthias Felleisen. Units: Cool Modules for HOT Languages. In *Proc. Programming Language Design and Implementation (PLDI)*, 1998.

Matthew Flatt and PLT. Reference: Racket. PLT Inc., PLT-TR-2010-1, 2010. `http://racket-lang.org/tr1/`

Spencer P. Florence, Shu-Hung You, Jesse A. Tov, and Robert Bruce Findler. A Calculus for Esterel. *Proceedings of the ACM on Programming Languages* 3(POPL), 2019.

Michael J C Gordon and Tom F Melham. Introduction to HOL: A Theorem Proving Environment for Higher Order Logic. Cambridge University Press, 1993.

Sharad Malik. Analysis of Cyclic Combinational Circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 13(7), 1994.

Michael Mendler, Thomas R. Shiple, and Gérard Berry. Constructive Boolean Circuits and the Exactness of Timed Ternary Simulation. *Formal Methods in System Design* 40, pp. 283–329, 2012.

James Hiram Morris. Lambda-Calculus Models of Programming Languages. PhD dissertation, Massachusetts Institute of Technology, 1963.

Gordon Plotkin. Call-by-name, Call-by-value, and the λ-Calculus. *Theoretical Computer Science* 1(2), pp. 125–159, 1975.

Dumitru Potop-Butucaru. Optimizations for Faster Simulation of Esterel Programs. PhD dissertation, Ecole des Mines de Paris, 2002.

Dumitru Potop-Butucaru, Stephen A. Edwards, and Gérard Berry. Compiling Esterel. Springer, 2007.

Klaus Schneider. Embedding Imperative Synchronous Languages in Interactive Theorem Provers. In *Proc. Second International Conference on Application of Concurrency to System Design (ACSD)*, 2001.

Klaus Schneider and Michael Wenz. A New Method for Compiling Schizophrenic Synchronous Programs. In *Proc. International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES)*, 2001.

Thomas R. Shiple, Gérard Berry, and Hervé Touati. Constructive Analysis of Cycle Circuits. In *Proc. European design and test conference*, 1996.

Vincent St-Amour, Same Tobin-Hochstadt, and Matthias Felleisen. Optimization Coaching: Optimizers Learn to Communicate with Programmers. In *Proc. ACM international conference on Object oriented programming systems languages and applications (OOPSLA)*, 2012.

Olivier Tardieu. A Deterministic Logical Semantics for Pure Esterel. *ACM Transactions on Programming Languages and Systems* 29(2), 2007.

Simone Tini. An Axiomatic Semantics for Esterel. *Theoretical Computer Science* 296, pp. 231–282, 2001.

Emina Torlak and Rastislav Bodik. Growing Solver-Aided Languages with ROSETTE. In *Proc. Onward!*, 2013.

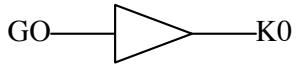
APPENDIX A

Definitions

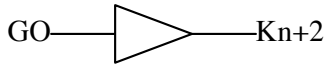
A.1. Circuits

Definition: $[\![\mathbf{p}^{p}]\!]$

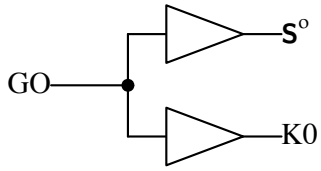
$[\![$nothing$]\!]$=



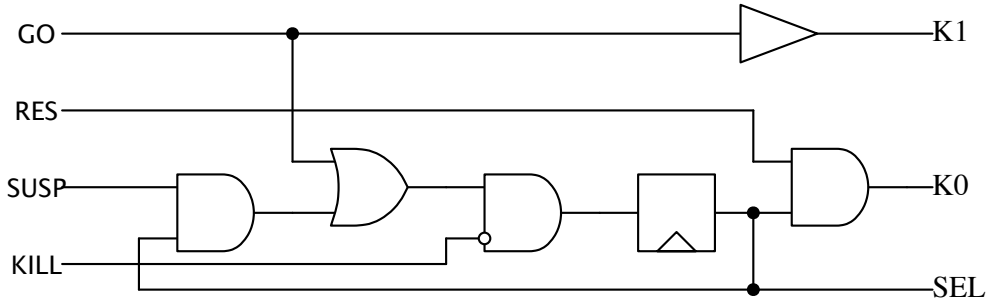
$[\![$(exit **n**)$]\!]$=



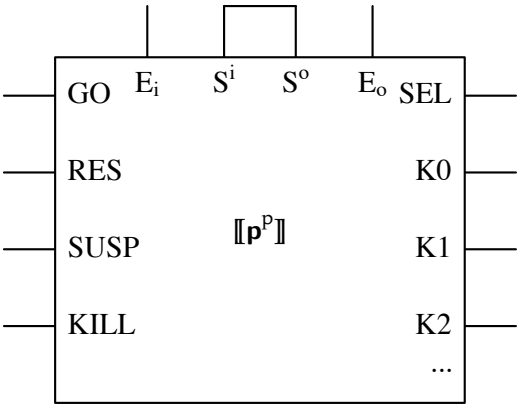
$[\![$(emit **S**)$]\!]$=



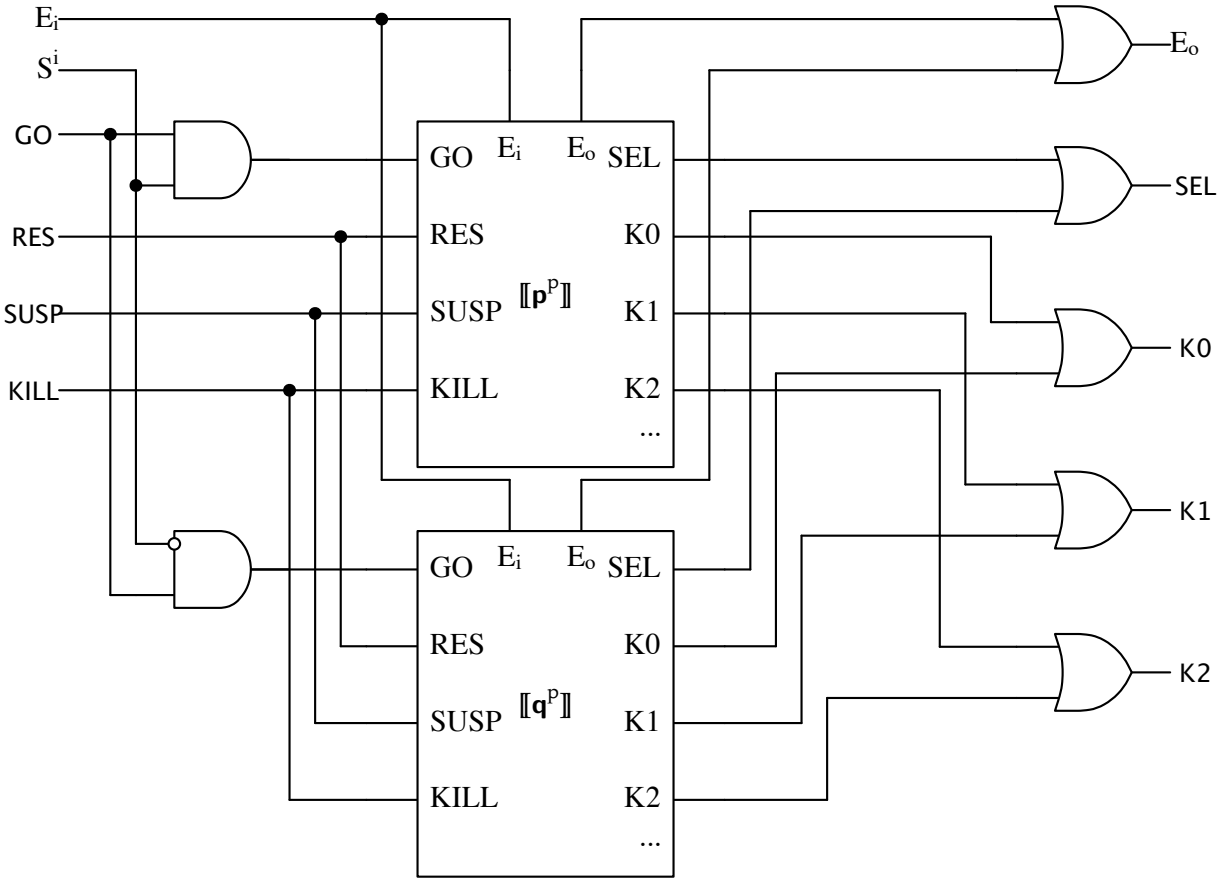
$[\![$pause$]\!]$=



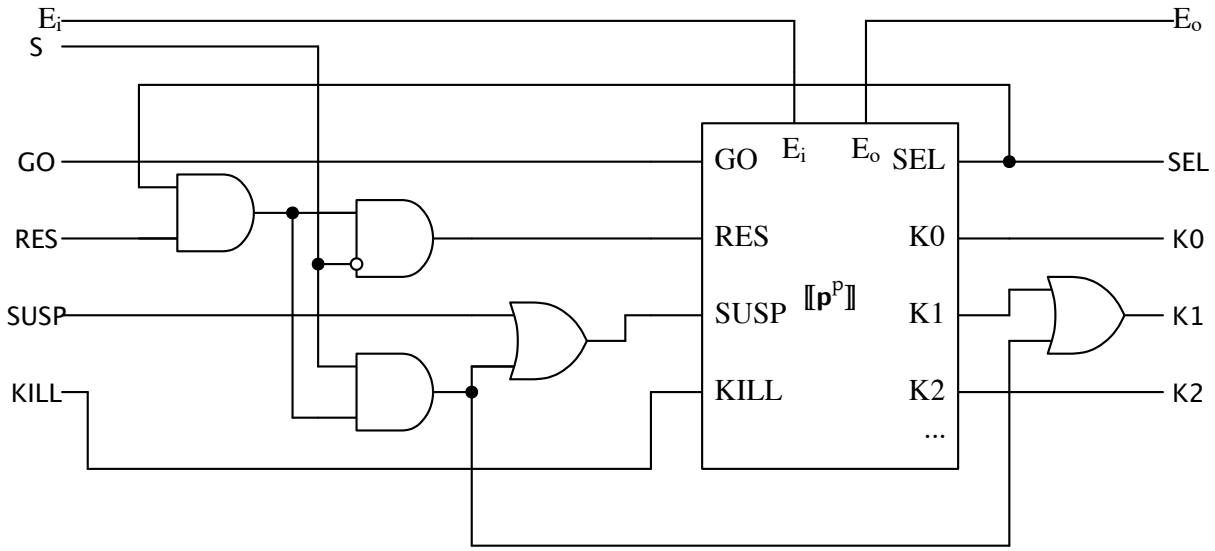
$[\![(\text{signal } \mathbf{S}\ \mathbf{p}^p)]\!]=$



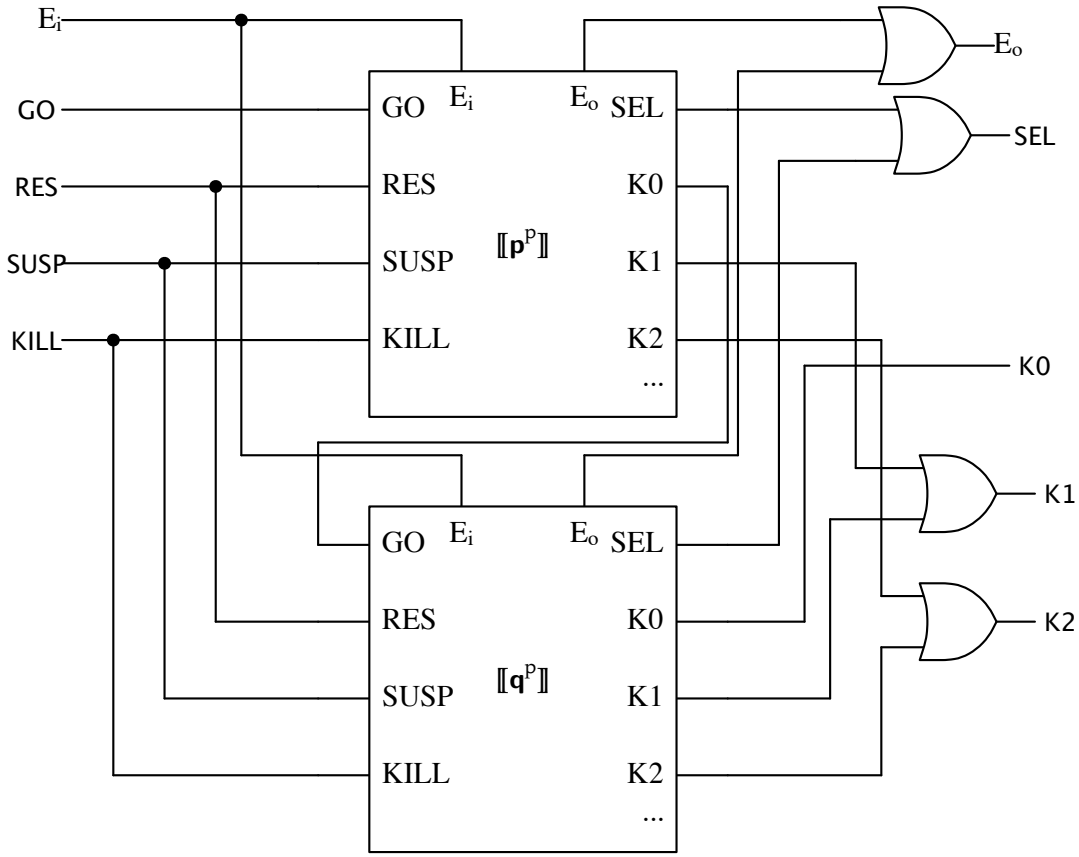
$[\![(\text{if } \mathbf{S}\ \mathbf{p}^p\ \mathbf{q}^p)]\!]=$



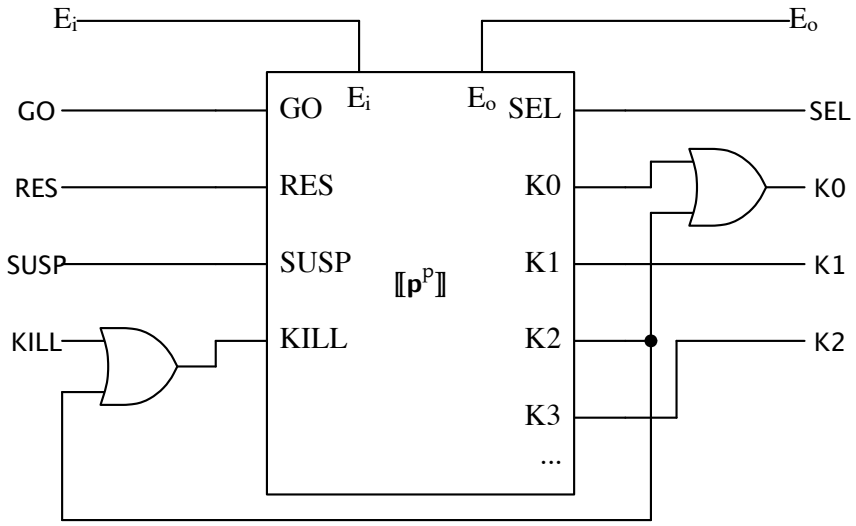
⟦(suspend **p**^{p} **S**)⟧=



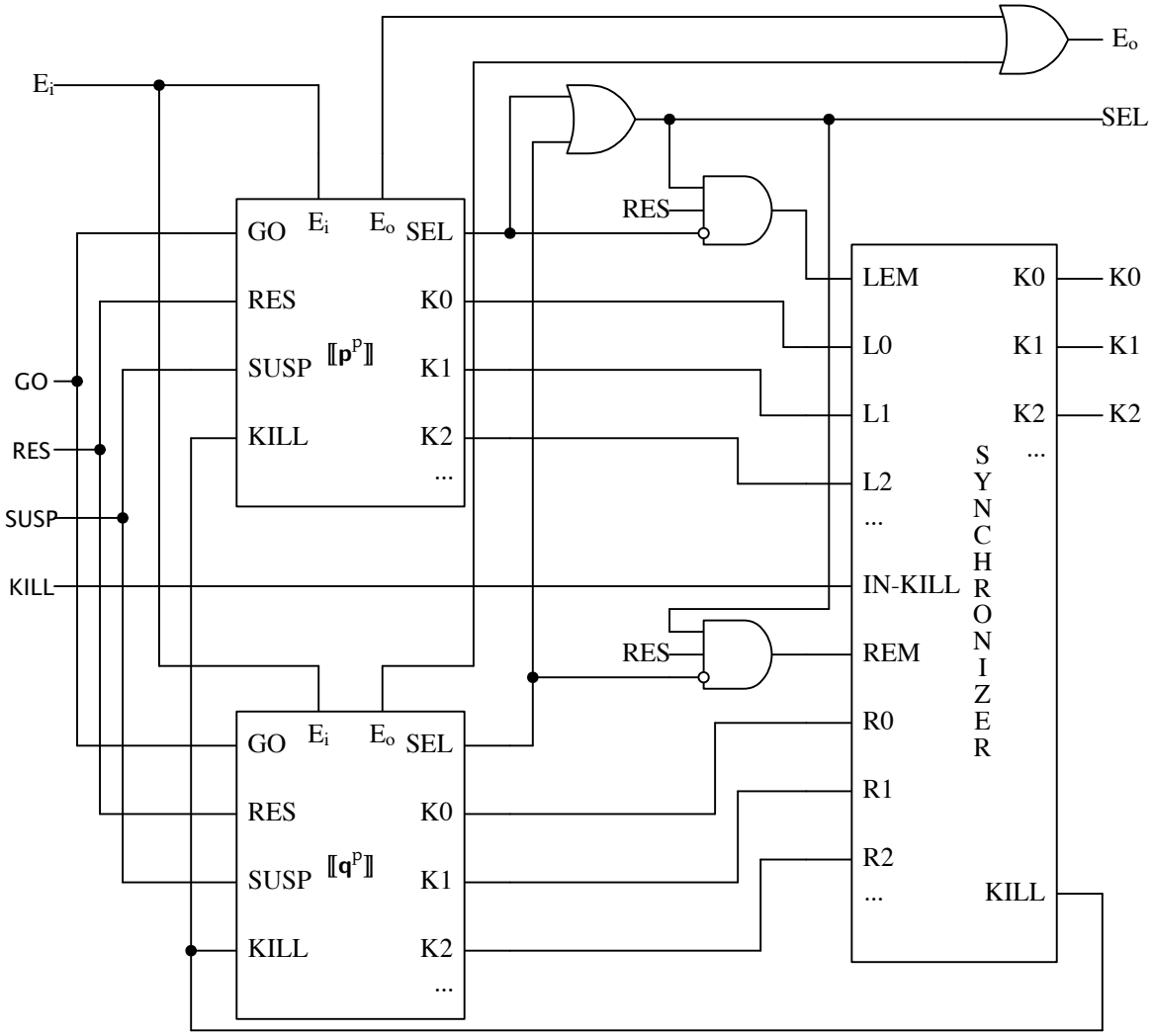
⟦(seq **p**^{p} **q**^{p})⟧=



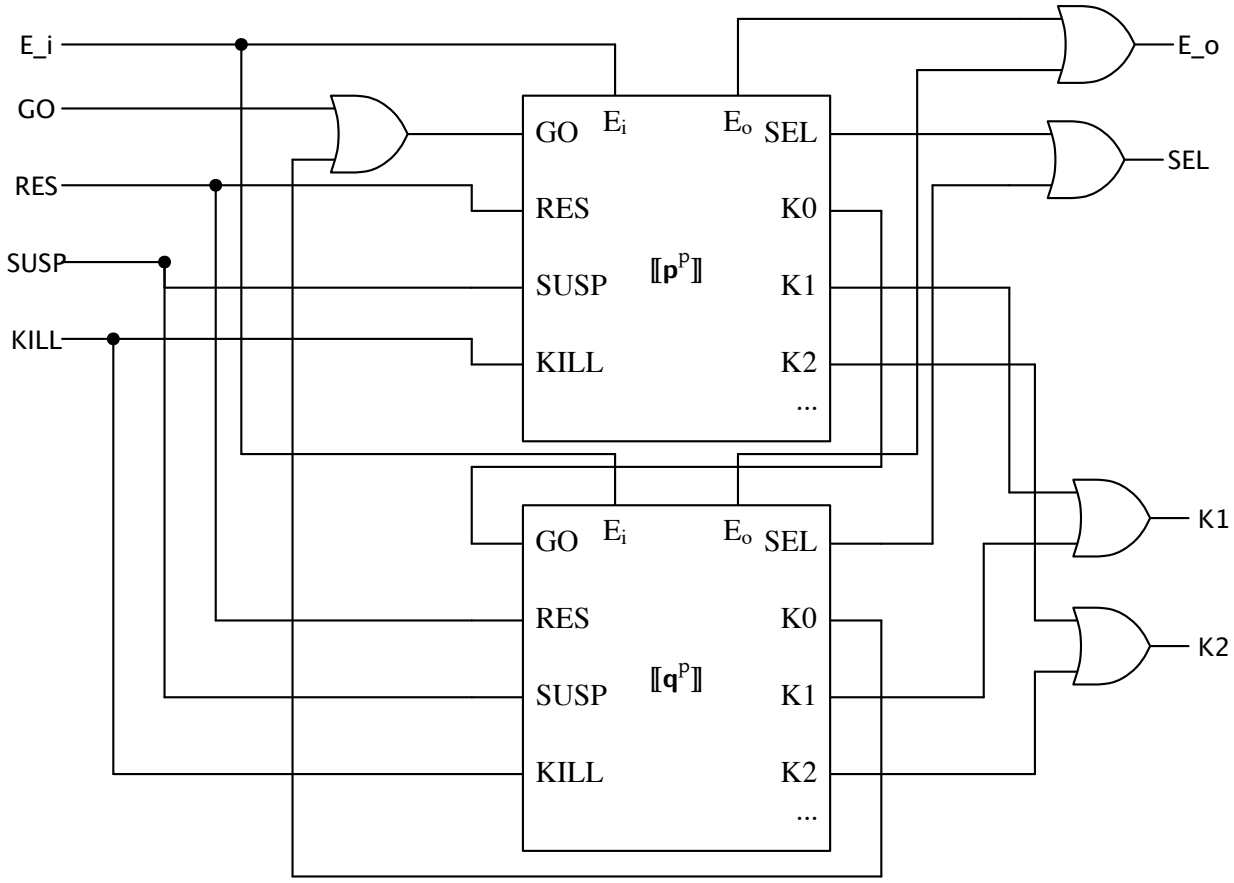
$[\![(\text{trap } \mathbf{p}^{p})]\!]=$



$[\![(\text{par } \mathbf{p}^p \ \mathbf{q}^p)]\!]=$

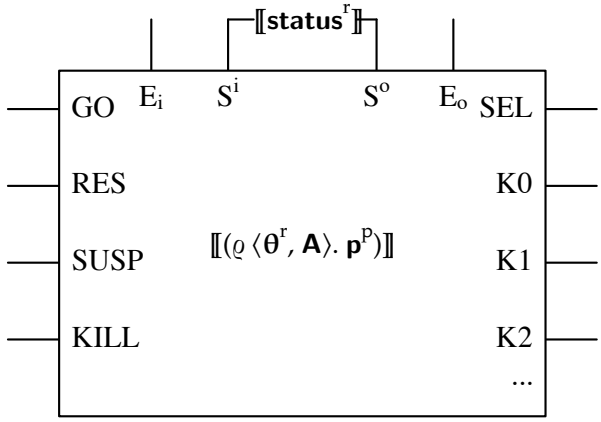


$[\![(\overline{\text{loop}}\ \mathbf{p}^p\ \mathbf{q}^p)]\!]=$

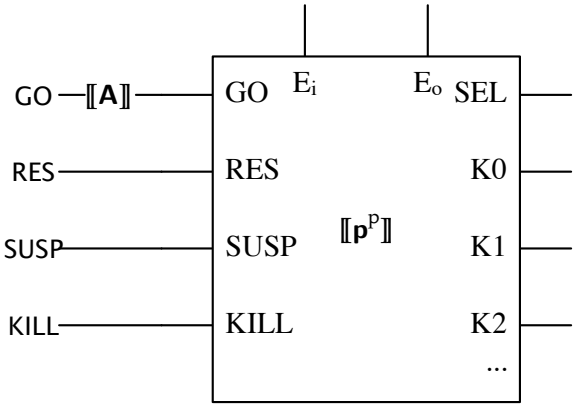


$[\![(\text{loop}\ \mathbf{p}^p)]\!]=$
$[\![(\overline{\text{loop}}\ \mathbf{p}^p\ \mathbf{p}^p)]\!]$

$[\![(\varrho\ \langle \theta^{r} \leftarrow \{\ \mathbf{S} \mapsto \mathbf{status}^{r}\ \},\ \mathbf{A}\rangle.\ \mathbf{p}^{p})]\!]=$



$[\![(\varrho\ \langle\{\},\ \mathbf{A}\rangle.\ \mathbf{p}^{p})]\!]=$



Definition: $[\![\mathbf{p}^{p}]\!] \setminus \theta$

read as: θ binds $[\![\mathbf{p}^{p}]\!]$

$[\![\mathbf{p}^{p}]\!] \setminus \theta$ *if and only if* $\forall\ \mathbf{S} \in \mathit{dom}(\theta),\ \theta(\mathbf{S}) = 1 \Leftrightarrow [\![\mathbf{p}^{p}]\!](\mathbf{S}^{i}) \simeq 1$, *and* $\theta(\mathbf{S}) = 0 \Leftrightarrow [\![\mathbf{p}^{p}]\!](\mathbf{S}^{i}) \simeq 0$.

Definition: $[\![\mathbf{p}^{p}]\!] \setminus \mathbf{A}$

read as: $\mathbf{A}$ binds $[\![\mathbf{p}^{p}]\!]$

$[\![\mathbf{p}^{p}]\!] \setminus \mathbf{A}$ *if and only if* $\mathbf{A} = \mathrm{GO}$ *implies that* $[\![\mathbf{p}^{p}]\!](\mathrm{GO}) = 1$.

Definition: $\mathfrak{C}(\mathbf{w}) \simeq \mathbf{e}$

$\mathfrak{C}$ is contextually equivalent to a circuit in which the definition of the wire $\mathbf{w}$ is replace by $\mathbf{e}$.

A.2. Calculus

Definition: $\mathbf{p}$, $\mathbf{q}$

$$
\begin{array}{l}
\mathbf{p}, \mathbf{q} ::= \text{nothing} \mid (\text{exit } \mathbf{n}) \mid (\text{emit } \mathbf{S}) \mid \text{pause} \\
\quad \mid (\text{signal } \mathbf{S}\ \mathbf{p}) \mid (\text{seq } \mathbf{p}\ \mathbf{q}) \mid (\text{if } \mathbf{S}\ \mathbf{p}\ \mathbf{q}) \mid (\text{par } \mathbf{p}\ \mathbf{q}) \\
\quad \mid (\text{loop } \mathbf{p}) \mid (\text{suspend } \mathbf{p}\ \mathbf{S}) \mid (\text{trap } \mathbf{p}) \\
\quad \mid (\text{shared } \mathbf{s} := \mathbf{e}\ \mathbf{p}) \mid (\mathrel{+}= \mathbf{s}\ \mathbf{e}) \mid (\text{var } \mathbf{x} := \mathbf{e}\ \mathbf{p}) \mid (:= \mathbf{x}\ \mathbf{e}) \mid (\text{if!0 } \mathbf{x}\ \mathbf{p}\ \mathbf{q}) \\
S \in \text{signal variables} \qquad x \in \text{sequential variables} \\
s \in \text{shared variables} \qquad e \in \text{host expressions}
\end{array}
$$

Definition: $\mathbf{p}^{\mathrm{p}}$, $\mathbf{q}^{\mathrm{p}}$

$$
\begin{array}{l}
\mathbf{p}^{\mathrm{p}}, \mathbf{q}^{\mathrm{p}} ::= \text{nothing} \\
\quad \mid \text{pause} \\
\quad \mid (\text{seq } \mathbf{p}^{\mathrm{p}}\ \mathbf{p}^{\mathrm{p}}) \\
\quad \mid (\text{par } \mathbf{p}^{\mathrm{p}}\ \mathbf{p}^{\mathrm{p}}) \\
\quad \mid (\text{trap } \mathbf{p}^{\mathrm{p}}) \\
\quad \mid (\text{exit } \mathbf{n}) \\
\quad \mid (\text{signal } \mathbf{S}\ \mathbf{p}^{\mathrm{p}}) \\
\quad \mid (\text{suspend } \mathbf{p}^{\mathrm{p}}\ \mathbf{S}) \\
\quad \mid (\text{if } \mathbf{S}\ \mathbf{p}^{\mathrm{p}}\ \mathbf{p}^{\mathrm{p}}) \\
\quad \mid (\text{emit } \mathbf{S}) \\
\quad \mid (\text{loop } \mathbf{p}^{\mathrm{p}}) \\
\quad \mid (\overline{\text{loop}}\ \mathbf{p}^{\mathrm{p}}\ \mathbf{q}^{\mathrm{p}}) \\
\quad \mid (\varrho\ \langle \theta^{\mathrm{r}}, \text{WAIT} \rangle.\ \mathbf{p}^{\mathrm{p}}) \\
S \in \text{signal variables}
\end{array}
$$

Definition: $\mathbf{p} \rightharpoonup^{E} \mathbf{q}$

signals	[**signal**]	$(\text{signal } \mathbf{S}\ \mathbf{p}) \rightharpoonup^{E} (\varrho\ \langle \{\ \mathbf{S} \mapsto \bot\ \}, \text{WAIT}\rangle.\ \mathbf{p})$
	[**emit**]	$(\varrho\ \langle \theta^{r}, \text{GO}\rangle.\ \mathbf{E}[(\text{emit } \mathbf{S})]) \rightharpoonup^{E} (\varrho\ \langle (\theta^{r} \leftarrow \{\ \mathbf{S} \mapsto 1\ \}), \text{GO}\rangle.\ \mathbf{E}[\text{nothing}])$ if $\mathbf{S} \in \mathit{dom}(\theta^{r})$
	[**is-present**]	$(\varrho\ \langle \theta^{r}, \mathbf{A}\rangle.\ \mathbf{E}[(\text{if } \mathbf{S}\ \mathbf{p}\ \mathbf{q})]) \rightharpoonup^{E} (\varrho\ \langle \theta^{r}, \mathbf{A}\rangle.\ \mathbf{E}[\mathbf{p}])$ if $\theta^{r}(\mathbf{S}) = 1$
	[**is-absent**]	$(\varrho\ \langle \theta^{r}, \mathbf{A}\rangle.\ \mathbf{E}[(\text{if } \mathbf{S}\ \mathbf{p}\ \mathbf{q})]) \rightharpoonup^{E} (\varrho\ \langle \theta^{r}, \mathbf{A}\rangle.\ \mathbf{E}[\mathbf{q}])$ if $\mathbf{S} \in \mathit{dom}(\theta^{r}), \theta^{r}(\mathbf{S}) = \bot, \mathbf{S} \notin \mathit{Can}^{S}_{\varrho}((\varrho\ \langle \theta^{r}, \mathbf{A}\rangle.\ \mathbf{E}[(\text{if } \mathbf{S}\ \mathbf{p}\ \mathbf{q})]), \{\})$
shared variables	[**shared**]	$(\varrho\ \langle \theta^{r}, \mathbf{A}\rangle.\ \mathbf{E}[(\text{shared } \mathbf{s} := \mathbf{e}\ \mathbf{p})]) \rightharpoonup^{E} (\varrho\ \langle \theta^{r}, \mathbf{A}\rangle.\ \mathbf{E}[(\varrho\ \langle \{\ \mathbf{s} \mapsto \langle \mathbf{n}, \text{old}\rangle\ \}, \text{WAIT}\rangle.\ \mathbf{p})])$ if $\mathit{FV}(\mathbf{e}) \subseteq \mathit{dom}(\theta^{r}), \forall\ \mathbf{s} \in \mathit{FV}(\mathbf{e}).\ \mathbf{s} \notin \mathit{Can}^{sh}_{\varrho}((\varrho\ \langle \theta^{r}, \mathbf{A}\rangle\ \mathbf{E}[(\text{shared } \mathbf{s} := \mathbf{e}\ \mathbf{p})]), \cdot),$ $\mathbf{n} = \mathit{eval}^{H}(\mathbf{e}, \theta^{r})$
	[**set-old**]	$(\varrho\ \langle \theta^{r}, \text{GO}\rangle.\ \mathbf{E}[(+= \mathbf{s}\ \mathbf{e})]) \rightharpoonup^{E} (\varrho\ \langle (\theta^{r} \leftarrow \{\ \mathbf{s} \mapsto \langle \mathit{eval}^{H}(\mathbf{e}, \theta^{r}), \text{new}\rangle\ \}), \text{GO}\rangle.\ \mathbf{E}[\text{nothing}])$ if $\theta^{r}(\mathbf{s}) = \langle \_, \text{old}\rangle, \mathit{FV}(\mathbf{e}) \subseteq \mathit{dom}(\theta^{r}), \forall\ \mathbf{s} \in \mathit{FV}(\mathbf{e}).\ \mathbf{s} \notin \mathit{Can}^{sh}_{\varrho}((\varrho\ \langle \theta^{r}, \text{GO}\rangle\ \mathbf{E}[(+= \mathbf{s}\ \mathbf{e})]), \cdot)$
	[**set-new**]	$(\varrho\ \langle \theta^{r}, \text{GO}\rangle.\ \mathbf{E}[(+= \mathbf{s}\ \mathbf{e})]) \rightharpoonup^{E} (\varrho\ \langle (\theta^{r} \leftarrow \{\ \mathbf{s} \mapsto \langle \mathbf{n} + \mathit{eval}^{H}(\mathbf{e}, \theta^{r}), \text{new}\rangle\ \}), \text{GO}\rangle.\ \mathbf{E}[\text{nothing}])$ if $\mathit{FV}(\mathbf{e}) \subseteq \mathit{dom}(\theta^{r}), \theta^{r}(\mathbf{s}) = \langle \mathbf{n}, \text{new}\rangle,$ $\forall\ \mathbf{s} \in \mathit{FV}(\mathbf{e}).\ \mathbf{s} \notin \mathit{Can}^{sh}_{\varrho}((\varrho\ \langle \theta^{r}, \text{GO}\rangle\ \mathbf{E}[(+= \mathbf{s}\ \mathbf{e})]), \cdot)$
sequential variables	[**var**]	$(\varrho\ \langle \theta^{r}, \mathbf{A}\rangle.\ \mathbf{E}[(\text{var } \mathbf{x} := \mathbf{e}\ \mathbf{p})]) \rightharpoonup^{E} (\varrho\ \langle \theta^{r}, \mathbf{A}\rangle.\ \mathbf{E}[(\varrho\ \langle \{\ \mathbf{x} \mapsto \mathit{eval}^{H}(\mathbf{e}, \theta^{r})\ \}, \text{WAIT}\rangle.\ \mathbf{p})])$ if $\mathit{FV}(\mathbf{e}) \subseteq \mathit{dom}(\theta^{r}), \forall\ \mathbf{s} \in \mathit{FV}(\mathbf{e}).\ \mathbf{s} \notin \mathit{Can}^{sh}_{\varrho}((\varrho\ \langle \theta^{r}, \mathbf{A}\rangle\ \mathbf{E}[(\text{var } \mathbf{x} := \mathbf{e}\ \mathbf{p})]), \cdot)$
	[**set-var**]	$(\varrho\ \langle \theta^{r}, \mathbf{A}\rangle.\ \mathbf{E}[(:= \mathbf{x}\ \mathbf{e})]) \rightharpoonup^{E} (\varrho\ \langle (\theta^{r} \leftarrow \{\ \mathbf{x} \mapsto \mathit{eval}^{H}(\mathbf{e}, \theta^{r})\ \}), \mathbf{A}\rangle.\ \mathbf{E}[\text{nothing}])$ if $\mathbf{x} \in \mathit{dom}(\theta^{r}), \mathit{FV}(\mathbf{e}) \subseteq \mathit{dom}(\theta^{r}), \forall\ \mathbf{s} \in \mathit{FV}(\mathbf{e}).\ \mathbf{s} \notin \mathit{Can}^{sh}_{\varrho}((\varrho\ \langle \theta^{r}, \mathbf{A}\rangle\ \mathbf{E}[(:= \mathbf{x}\ \mathbf{e})]), \cdot)$
	[**if-true**]	$(\varrho\ \langle \theta^{r}, \text{GO}\rangle.\ \mathbf{E}[(\text{if!0 } \mathbf{x}\ \mathbf{p}\ \mathbf{q})]) \rightharpoonup^{E} (\varrho\ \langle \theta^{r}, \text{GO}\rangle.\ \mathbf{E}[\mathbf{p}])$ if $\mathbf{x} \in \mathit{dom}(\theta^{r}), \theta^{r}(\mathbf{x}) \neq 0$
	[**if-false**]	$(\varrho\ \langle \theta^{r}, \text{GO}\rangle.\ \mathbf{E}[(\text{if!0 } \mathbf{x}\ \mathbf{p}\ \mathbf{q})]) \rightharpoonup^{E} (\varrho\ \langle \theta^{r}, \text{GO}\rangle.\ \mathbf{E}[\mathbf{q}])$ if $\theta^{r}(\mathbf{x}) = 0$
ϱ	[**merge**]	$(\varrho\ \langle \theta^{r}{}_{1}, \mathbf{A}_{1}\rangle.\ \mathbf{E}[(\varrho\ \langle \theta^{r}{}_{2}, \mathbf{A}_{2}\rangle.\ \mathbf{p})]) \rightharpoonup^{E} (\varrho\ \langle (\theta^{r}{}_{1} \leftarrow \theta^{r}{}_{2}), \mathbf{A}_{1}\rangle.\ \mathbf{E}[\mathbf{p}])$ if $\mathbf{A}_{1} \geq \mathbf{A}_{2}$
seq	[**seq-done**]	$(\text{seq nothing } \mathbf{q}) \rightharpoonup^{E} \mathbf{q}$
	[**seq-exit**]	$(\text{seq } (\text{exit } \mathbf{n})\ \mathbf{q}) \rightharpoonup^{E} (\text{exit } \mathbf{n})$
trap	[**trap**]	$(\text{trap } \mathbf{p}^{S}) \rightharpoonup^{E} \downarrow^{p} \mathbf{p}^{S}$
par	[**par-nothing**]	$(\text{par nothing } \mathbf{p}^{D}) \rightharpoonup^{E} \mathbf{p}^{D}$
	[**par-1exit**]	$(\text{par } (\text{exit } \mathbf{n})\ \hat{\mathbf{p}}) \rightharpoonup^{E} (\text{exit } \mathbf{n})$
	[**par-2exit**]	$(\text{par } (\text{exit } \mathbf{n}_{1})\ (\text{exit } \mathbf{n}_{2})) \rightharpoonup^{E} (\text{exit } \mathit{max}(\mathbf{n}_{1}, \mathbf{n}_{2}))$
	[**par-swap**]	$(\text{par } \mathbf{p}\ \mathbf{q}) \rightharpoonup^{E} (\text{par } \mathbf{q}\ \mathbf{p})$
	[**suspend**]	$(\text{suspend } \mathbf{p}^{S}\ \mathbf{S}) \rightharpoonup^{E} \mathbf{p}^{S}$
loop	[**loop**]	$(\text{loop } \mathbf{p}) \rightharpoonup^{E} (\overline{\text{loop}}\ \mathbf{p}\ \mathbf{p})$
	[**loop^stop-exit**]	$(\overline{\text{loop}}\ (\text{exit } \mathbf{n})\ \mathbf{q}) \rightharpoonup^{E} (\text{exit } \mathbf{n})$

Definition: $\mathbf{p} \longrightarrow \mathbf{q}$

The compatible closure of $\rightharpoonup^{\mathrm{E}}$.

Definition: $\mathbf{p} \equiv^{\mathrm{E}} \mathbf{q}$

The transitive, reflexive, symmetric, compatible closure of $\rightharpoonup^{\mathrm{E}}$.

Definition: $\mathit{eval}/^{\mathrm{E}}(\mathbf{O}, \mathbf{p})$

$$\begin{array}{l}
\mathit{eval}/^{\mathrm{E}} : \mathbf{O}\ \mathbf{p} \to \langle \theta\ , \mathbf{bool} \rangle \\
\mathit{eval}/^{\mathrm{E}}(\mathbf{O}, (\varrho\ \langle \theta^{\mathrm{r}}{}_1, \mathrm{GO} \rangle.\ \mathbf{p}^{\mathrm{p}})) \ = \ \langle \mathit{restrict}(\theta^{\mathrm{r}}{}_2, \mathbf{O}, (\varrho\ \langle \theta^{\mathrm{r}}{}_2, \mathrm{GO} \rangle.\ \mathbf{p}^{\mathrm{D}}))\ ,\ \mathit{tt} \rangle \\
\quad \text{if } (\varrho\ \langle \theta^{\mathrm{r}}{}_1, \mathrm{GO} \rangle.\ \mathbf{p}^{\mathrm{p}}) \equiv^{\mathrm{E}} (\varrho\ \langle \theta^{\mathrm{r}}{}_2, \mathrm{GO} \rangle.\ \mathbf{p}^{\mathrm{D}}),\ \mathit{complete\text{-}wrt}(\theta^{\mathrm{r}}{}_2, \mathbf{p}^{\mathrm{D}}) \\
\mathit{eval}/^{\mathrm{E}}(\mathbf{O}, (\varrho\ \langle \theta^{\mathrm{r}}{}_1, \mathrm{GO} \rangle.\ \mathbf{p}^{\mathrm{p}})) \ = \ \langle \mathit{restrict}(\theta^{\mathrm{r}}{}_2, \mathbf{O}, (\varrho\ \langle \theta^{\mathrm{r}}{}_2, \mathrm{GO} \rangle.\ \mathbf{q}^{\mathrm{p}}))\ ,\ \mathit{ff} \rangle \\
\quad \text{if } (\varrho\ \langle \theta^{\mathrm{r}}{}_1, \mathrm{GO} \rangle.\ \mathbf{p}^{\mathrm{p}}) \equiv^{\mathrm{E}} (\varrho\ \langle \theta^{\mathrm{r}}{}_2, \mathrm{GO} \rangle.\ \mathbf{q}^{\mathrm{p}}),\ \theta^{\mathrm{r}}{}_2; \mathrm{GO}; \bigcirc \vdash_{\mathrm{B}} \mathbf{q}^{\mathrm{p}}
\end{array}$$

Definition: $\mathit{restrict}(\theta, \mathbf{O}, \mathbf{p})$

read as: Restrict θ to signals in $\mathbf{O}$, given their values as determined by the program $\mathbf{p}$.

$$\mathit{restrict}(\theta, \mathbf{O}, \mathbf{p})(\mathbf{S}) = \begin{cases} 0 & \text{where } \mathbf{S} \in \mathbf{O},\ \theta(\mathbf{S}) = \bot, \text{ and } \mathbf{S} \notin \mathit{Can}^{\mathbf{S}}_{\varrho}(\mathbf{p}, \{\}) \\ \theta(\mathbf{S}) & \text{where } \mathbf{S} \in \mathbf{O} \end{cases}$$

Definition: $\theta^{r}; \mathbf{A}; \mathbf{E}^{p} \vdash_{B} \mathbf{p}^{p}$

$$\frac{\theta^{r}(\mathbf{S}) = \bot \qquad \mathbf{S} \in \mathit{Can}^{S}_{Q}\big((_{Q}\,\langle \theta^{r}, \mathbf{A}\rangle.\ \mathbf{E}[(\text{if } \mathbf{S}\ \mathbf{p}\ \mathbf{q})]), \{\}\big)}{\theta^{r}; \mathbf{A}; \mathbf{E} \vdash_{B} (\text{if } \mathbf{S}\ \mathbf{p}\ \mathbf{q})}\,[\text{if}] \quad \frac{}{\theta^{r}; \text{WAIT}; \mathbf{E} \vdash_{B} (\text{emit } \mathbf{S})}\,[\text{emit-wait}]$$

$$\frac{\theta^{r}; \mathbf{A}; \mathbf{E}[(\text{suspend} \bigcirc \mathbf{S})] \vdash_{B} \mathbf{p}}{\theta^{r}; \mathbf{A}; \mathbf{E} \vdash_{B} (\text{suspend}\ \mathbf{p}\ \mathbf{S})}\,[\text{suspend}] \quad \frac{\theta^{r}; \mathbf{A}; \mathbf{E}[(\text{trap} \bigcirc)] \vdash_{B} \mathbf{p}}{\theta^{r}; \mathbf{A}; \mathbf{E} \vdash_{B} (\text{trap}\ \mathbf{p})}\,[\text{trap}]$$

$$\frac{\theta^{r}; \mathbf{A}; \mathbf{E}[(\text{seq} \bigcirc \mathbf{q})] \vdash_{B} \mathbf{p}}{\theta^{r}; \mathbf{A}; \mathbf{E} \vdash_{B} (\text{seq}\ \mathbf{p}\ \mathbf{q})}\,[\text{seq}]$$

$$\frac{\theta^{r}; \mathbf{A}; \mathbf{E}[(\text{par} \bigcirc \mathbf{p}^{D})] \vdash_{B} \mathbf{p}}{\theta^{r}; \mathbf{A}; \mathbf{E} \vdash_{B} (\text{par}\ \mathbf{p}\ \mathbf{p}^{D})}\,[\text{parl}] \quad \frac{\theta^{r}; \mathbf{A}; \mathbf{E}[(\text{par}\ \mathbf{p}^{D} \bigcirc)] \vdash_{B} \mathbf{q}}{\theta^{r}; \mathbf{A}; \mathbf{E} \vdash_{B} (\text{par}\ \mathbf{p}^{D}\ \mathbf{q})}\,[\text{parr}]$$

$$\frac{\theta^{r}; \mathbf{A}; \mathbf{E}[(\text{par} \bigcirc \mathbf{q})] \vdash_{B} \mathbf{p} \qquad \theta^{r}; \mathbf{A}; \mathbf{E}[(\text{par}\ \mathbf{p} \bigcirc)] \vdash_{B} \mathbf{q}}{\theta^{r}; \mathbf{A}; \mathbf{E} \vdash_{B} (\text{par}\ \mathbf{p}\ \mathbf{q})}\,[\text{par-both}]$$

Definition: $Can(\mathbf{p}, \theta)$

$\boldsymbol{\kappa} \in$ Natural Numbers

$$
\begin{array}{|l}
\hline
\downarrow^{\kappa} : \boldsymbol{\kappa} \to \boldsymbol{\kappa} \\
\downarrow^{\kappa} 0 = 0 \\
\downarrow^{\kappa} 1 = 1 \\
\downarrow^{\kappa} 2 = 0 \\
\downarrow^{\kappa} \mathbf{n} = \mathbf{n}\text{-}1 \\
\text{if } \mathbf{n} > 2 \\
\hline
\end{array}
$$

$$
\begin{array}{ll}
\mathit{Can} : \mathbf{p}\ \theta \to \{ \text{S: } \mathbb{S}, \text{K: } \mathbb{k}, \text{sh: } \mathbb{s} \} & \\
\mathit{Can}(\text{nothing}, \theta) & = \{ \text{S} = \varnothing, \text{K} = \{ 0 \}, \text{sh} = \varnothing \} \\
\mathit{Can}(\text{pause}, \theta) & = \{ \text{S} = \varnothing, \text{K} = \{ 1 \}, \text{sh} = \varnothing \} \\
\mathit{Can}((\text{exit } \mathbf{n}), \theta) & = \{ \text{S} = \varnothing, \text{K} = \{ \mathbf{n} + 2 \}, \text{sh} = \varnothing \} \\
\mathit{Can}((\text{emit } \mathbf{S}), \theta) & = \{ \text{S} = \{ \mathbf{S} \}, \text{K} = \{ 0 \}, \text{sh} = \varnothing \} \\
\mathit{Can}((\text{if } \mathbf{S}\ \mathbf{p}\ \mathbf{q}), \theta) & = \mathit{Can}(\mathbf{p}, \theta) \\
\text{if } \theta(\mathbf{S}) = 1 & \\
\mathit{Can}((\text{if } \mathbf{S}\ \mathbf{p}\ \mathbf{q}), \theta) & = \mathit{Can}(\mathbf{q}, \theta) \\
\text{if } \theta(\mathbf{S}) = 0 & \\
\mathit{Can}((\text{if } \mathbf{S}\ \mathbf{p}\ \mathbf{q}), \theta) & = \{\ \text{S} = \mathit{Can}^{\text{S}}(\mathbf{p}, \theta) \cup \mathit{Can}^{\text{S}}(\mathbf{q}, \theta), \\
 & \quad \text{K} = \mathit{Can}^{\text{K}}(\mathbf{p}, \theta) \cup \mathit{Can}^{\text{K}}(\mathbf{q}, \theta), \\
 & \quad \text{sh} = \mathit{Can}^{\text{sh}}(\mathbf{p}, \theta) \cup \mathit{Can}^{\text{sh}}(\mathbf{q}, \theta)\ \} \\
\mathit{Can}((\text{suspend } \mathbf{p}\ \mathbf{S}), \theta) & = \mathit{Can}(\mathbf{p}, \theta) \\
\mathit{Can}((\text{seq } \mathbf{p}\ \mathbf{q}), \theta) & = \mathit{Can}(\mathbf{p}, \theta) \\
\text{if } 0 \notin \mathit{Can}^{\text{K}}(\mathbf{p}, \theta) & \\
\mathit{Can}((\text{seq } \mathbf{p}\ \mathbf{q}), \theta) & = \{\ \text{S} = \mathit{Can}^{\text{S}}(\mathbf{p}, \theta) \cup \mathit{Can}^{\text{S}}(\mathbf{q}, \theta), \\
 & \quad \text{K} = \mathit{Can}^{\text{K}}(\mathbf{p}, \theta) \setminus \{ 0 \} \cup \mathit{Can}^{\text{K}}(\mathbf{q}, \theta), \\
 & \quad \text{sh} = \mathit{Can}^{\text{sh}}(\mathbf{p}, \theta) \cup \mathit{Can}^{\text{sh}}(\mathbf{q}, \theta)\ \} \\
\mathit{Can}((\text{loop } \mathbf{p}), \theta) & = \mathit{Can}(\mathbf{p}, \theta) \\
\mathit{Can}((\overline{\text{loop}}\ \mathbf{p}\ \mathbf{q}), \theta) & = \mathit{Can}(\mathbf{p}, \theta) \\
\mathit{Can}((\text{par } \mathbf{p}\ \mathbf{q}), \theta) & = \{\ \text{S} = \mathit{Can}^{\text{S}}(\mathbf{p}, \theta) \cup \mathit{Can}^{\text{S}}(\mathbf{q}, \theta), \\
 & \quad \text{K} = \{\ \mathit{max}(\boldsymbol{\kappa}_1, \boldsymbol{\kappa}_2) \mid \kappa_1 \in \mathit{Can}^{\text{K}}(\mathbf{p}, \theta)\ , \kappa_2 \in \mathit{Can}^{\text{K}}(\mathbf{q}, \theta)\ \}, \\
 & \quad \text{sh} = \mathit{Can}^{\text{sh}}(\mathbf{p}, \theta) \cup \mathit{Can}^{\text{sh}}(\mathbf{q}, \theta)\ \} \\
\mathit{Can}((\text{trap } \mathbf{p}), \theta) & = \{ \text{S} = \mathit{Can}^{\text{S}}(\mathbf{p}, \theta), \text{K} = \{ \downarrow^{\kappa} \text{x} \mid \text{x} \in \mathit{Can}^{\text{K}}(\mathbf{p}, \theta) \}, \text{sh} = \mathit{Can}^{\text{sh}}(\mathbf{p}, \theta)\ \} \\
\mathit{Can}((\text{signal } \mathbf{S}\ \mathbf{p}), \theta) & = \{\ \text{S} = \mathit{Can}^{\text{S}}(\mathbf{p}, \theta \leftarrow \{ \mathbf{S} \mapsto 0 \}) \setminus \{ \mathbf{S} \}, \\
 & \quad \text{K} = \mathit{Can}^{\text{K}}(\mathbf{p}, \theta \leftarrow \{ \mathbf{S} \mapsto 0 \}), \\
 & \quad \text{sh} = \mathit{Can}^{\text{sh}}(\mathbf{p}, \theta \leftarrow \{ \mathbf{S} \mapsto 0 \})\ \} \\
\text{if } \mathbf{S} \notin \mathit{Can}^{\text{S}}(\mathbf{p}, \theta \leftarrow \{ \mathbf{S} \mapsto \bot \}) & \\
\mathit{Can}((\text{signal } \mathbf{S}\ \mathbf{p}), \theta) & = \{ \text{S} = \mathit{Can}^{\text{S}}(\mathbf{p}, \theta_2) \setminus \{ \mathbf{S} \}, \text{K} = \mathit{Can}^{\text{K}}(\mathbf{p}, \theta_2), \text{sh} = \mathit{Can}^{\text{sh}}(\mathbf{p}, \theta_2)\ \} \\
\text{if } \theta_2 = \theta \leftarrow \{ \mathbf{S} \mapsto \bot \} & \\
\mathit{Can}((\varrho\ \langle \theta_1, \mathbf{A} \rangle .\ \mathbf{p}), \theta_2) & = \{\ \text{S} = \mathit{Can}^{\text{S}}_{\varrho}((\varrho\ \langle \theta_1, \mathbf{A} \rangle .\ \mathbf{p}), \theta_2) \setminus \mathit{dom}(\theta_1), \\
 & \quad \text{K} = \mathit{Can}^{\text{K}}_{\varrho}((\varrho\ \langle \theta_1, \mathbf{A} \rangle .\ \mathbf{p}), \theta_2), \\
 & \quad \text{sh} = \mathit{Can}^{\text{sh}}_{\varrho}((\varrho\ \langle \theta_1, \mathbf{A} \rangle .\ \mathbf{p}), \theta_2) \setminus \mathit{dom}(\theta_1)\ \} \\
\mathit{Can}((\text{shared } \mathbf{s} := \mathbf{e}\ \mathbf{p}), \theta) & = \{ \text{S} = \mathit{Can}^{\text{S}}(\mathbf{p}, \theta), \text{K} = \mathit{Can}^{\text{K}}(\mathbf{p}, \theta), \text{sh} = \mathit{Can}^{\text{sh}}(\mathbf{p}, \theta) \setminus \{ \mathbf{s} \}\ \} \\
\mathit{Can}((\mathrel{+}= \mathbf{s}\ \mathbf{e}), \theta) & = \{ \text{S} = \varnothing, \text{K} = \{ 0 \}, \text{sh} = \{ \mathbf{s} \}\ \} \\
\mathit{Can}((\text{var } \mathbf{x} := \mathbf{e}\ \mathbf{p}), \theta) & = \mathit{Can}(\mathbf{p}, \theta) \\
\mathit{Can}((:= \mathbf{x}\ \mathbf{e}), \theta) & = \{ \text{S} = \varnothing, \text{K} = \{ 0 \}, \text{sh} = \varnothing \} \\
\mathit{Can}((\text{if!0 } \mathbf{x}\ \mathbf{p}\ \mathbf{q}), \theta) & = \{\ \text{S} = \mathit{Can}^{\text{S}}(\mathbf{p}, \theta) \cup \mathit{Can}^{\text{S}}(\mathbf{q}, \theta), \\
 & \quad \text{K} = \mathit{Can}^{\text{K}}(\mathbf{p}, \theta) \cup \mathit{Can}^{\text{K}}(\mathbf{q}, \theta), \\
 & \quad \text{sh} = \mathit{Can}^{\text{sh}}(\mathbf{p}, \theta) \cup \mathit{Can}^{\text{sh}}(\mathbf{q}, \theta)\ \}
\end{array}
$$

Definition: $\mathit{Can}_\varrho(\!|(\varrho\,\langle\theta_1, \mathbf{A}\rangle.\ \mathbf{p}), \theta_2|\!)$

$$
\begin{array}{l}
\mathit{Can}_\varrho : (\varrho\,\langle\theta, \mathbf{A}\rangle.\ \mathbf{p})\ \theta \to \{\,\mathrm{S}: \mathbb{S}, \mathrm{K}: \mathbb{k}, \mathrm{sh}: \mathbb{s}\,\} \\
\mathit{Can}_\varrho(\!|(\varrho\,\langle\theta, \mathbf{A}\rangle.\ \mathbf{p}), \theta_2|\!) \;=\; \mathit{Can}_\varrho(\!|(\varrho\,\langle(\theta \setminus \{\mathbf{S}\}), \mathbf{A}\rangle.\ \mathbf{p}), \theta_2 \leftarrow \{\,\mathbf{S} \mapsto 0\,\}|\!) \\
\text{if } \mathbf{S} \in \mathit{dom}(\!|\theta|\!), \\
\quad \theta(\mathbf{S}) = \bot, \\
\quad \mathbf{S} \notin \overset{\mathrm{S}}{\mathit{Can}}_\varrho(\!|(\varrho\,\langle(\theta \setminus \{\mathbf{S}\}), \mathbf{A}\rangle.\ \mathbf{p}), \theta_2 \leftarrow \{\,\mathbf{S} \mapsto \bot\,\}|\!) \\
\mathit{Can}_\varrho(\!|(\varrho\,\langle\theta, \mathbf{A}\rangle.\ \mathbf{p}), \theta_2|\!) \;=\; \mathit{Can}_\varrho(\!|(\varrho\,\langle(\theta \setminus \{\mathbf{S}\}), \mathbf{A}\rangle.\ \mathbf{p}), \theta_2 \leftarrow \{\,\mathbf{S} \mapsto \theta(\mathbf{S})\,\}|\!) \\
\text{if } \mathbf{S} \in \mathit{dom}(\!|\theta|\!) \\
\mathit{Can}_\varrho(\!|(\varrho\,\langle\theta_1, \mathbf{A}\rangle.\ \mathbf{p}), \theta_2|\!) \;=\; \mathit{Can}(\!|\mathbf{p}, \theta_2|\!)
\end{array}
$$

Definition: $\mathit{complete\text{-}wrt}(\!|\theta^{\mathrm{r}}, \mathbf{p}^{\mathrm{D}}|\!)$

For all $\mathbf{S} \in \mathit{dom}(\!|\theta^{\mathrm{r}}|\!)$, *either* $\theta^{\mathrm{r}}(\mathbf{S}) = 1$ *or* $\theta^{\mathrm{r}}(\mathbf{S}) = \bot$ *and* $\mathbf{S} \notin \overset{\mathrm{S}}{\mathit{Can}}_\varrho(\!|(\varrho\,\langle\theta^{\mathrm{r}}, \mathrm{GO}\rangle.\ \mathbf{p}^{\mathrm{D}}), \{\}|\!)$.

A.3. Auxiliary

Definition: $\mathcal{E}(\!|\mathbf{p}^{\mathrm{C}}|\!)$

$$
\begin{array}{lcl}
\mathcal{E} : \mathbf{p}^{\mathrm{C}} \to \mathbf{p} & & \\
\mathcal{E}(\!|(\varrho\,\langle\theta^{\mathrm{r}}, \mathrm{GO}\rangle.\ \mathbf{p})|\!) & = & (\varrho\,\langle\lfloor\theta^{\mathrm{r}}\rfloor, \mathrm{WAIT}\rangle.\ \mathcal{E}(\!|\mathbf{p}|\!)) \\
\mathcal{E}(\!|\mathrm{pause}|\!) & = & \mathrm{nothing} \\
\mathcal{E}(\!|\mathrm{nothing}|\!) & = & \mathrm{nothing} \\
\mathcal{E}(\!|(\overline{\mathrm{loop}}\ \mathbf{p}\ \mathbf{q})|\!) & = & (\mathrm{seq}\ \mathcal{E}(\!|\mathbf{p}|\!)\ (\mathrm{loop}\ \mathbf{q})) \\
\mathcal{E}(\!|(\mathrm{seq}\ \mathbf{p}\ \mathbf{q})|\!) & = & (\mathrm{seq}\ \mathcal{E}(\!|\mathbf{p}|\!)\ \mathbf{q}) \\
\mathcal{E}(\!|(\mathrm{par}\ \mathbf{p}\ \mathbf{q})|\!) & = & (\mathrm{par}\ \mathcal{E}(\!|\mathbf{p}|\!)\ \mathcal{E}(\!|\mathbf{q}|\!)) \\
\mathcal{E}(\!|(\mathrm{suspend}\ \mathbf{p}\ \mathbf{S})|\!) & = & (\mathrm{suspend}\ (\mathrm{seq}\ (\mathrm{if}\ \mathbf{S}\ \mathrm{pause}\ \mathrm{nothing})\ \mathcal{E}(\!|\mathbf{p}|\!))\ \mathbf{S}) \\
\mathcal{E}(\!|(\mathrm{trap}\ \mathbf{p})|\!) & = & (\mathrm{trap}\ \mathcal{E}(\!|\mathbf{p}|\!)) \\
\mathcal{E}(\!|(\mathrm{exit}\ \mathbf{n})|\!) & = & (\mathrm{exit}\ \mathbf{n})
\end{array}
$$

Definition: $\mathscr{S}(\mathbf{p})$

read as: An upper bound in the number of $\longrightarrow^{\mathrm{R}}$ steps $\mathbf{p}$ may take. (The name is a very bad pun on the physics notation for an Action.)

$$
\begin{array}{ll}
\mathscr{S}: \mathbf{p}^{\mathrm{p}}_{\mathrm{GO}} \to \mathbf{n} & \\
\mathscr{S}(\text{nothing}) & = 0 \\
\mathscr{S}(\text{pause}) & = 0 \\
\mathscr{S}((\text{exit } \mathbf{n})) & = 0 \\
\mathscr{S}((\text{emit } \mathbf{S})) & = 1 \\
\mathscr{S}((\text{signal } \mathbf{S}\ \mathbf{p}^{\mathrm{p}}_{\mathrm{GO}})) & = 2 + \mathscr{S}(\mathbf{p}^{\mathrm{p}}_{\mathrm{GO}}) \\
\mathscr{S}((\text{if } \mathbf{S}\ \mathbf{p}^{\mathrm{p}}_{\mathrm{GO}}\ \mathbf{q}^{\mathrm{p}}_{\mathrm{GO}})) & = 1 + \mathscr{S}(\mathbf{p}^{\mathrm{p}}_{\mathrm{GO}}) + \mathscr{S}(\mathbf{q}^{\mathrm{p}}_{\mathrm{GO}}) \\
\mathscr{S}((\text{par } \mathbf{p}^{\mathrm{p}}_{\mathrm{GO}}\ \mathbf{q}^{\mathrm{p}}_{\mathrm{GO}})) & = 1 + \mathscr{S}(\mathbf{p}^{\mathrm{p}}_{\mathrm{GO}}) + \mathscr{S}(\mathbf{q}^{\mathrm{p}}_{\mathrm{GO}}) \\
\mathscr{S}((\text{seq } \mathbf{p}^{\mathrm{p}}_{\mathrm{GO}}\ \mathbf{q}^{\mathrm{p}}_{\mathrm{GO}})) & = 1 + \mathscr{S}(\mathbf{p}^{\mathrm{p}}_{\mathrm{GO}}) + \mathscr{S}(\mathbf{q}^{\mathrm{p}}_{\mathrm{GO}}) \\
\mathscr{S}((\text{trap } \mathbf{p}^{\mathrm{p}}_{\mathrm{GO}})) & = 1 + \mathscr{S}(\mathbf{p}^{\mathrm{p}}_{\mathrm{GO}}) \\
\mathscr{S}((\text{suspend } \mathbf{p}^{\mathrm{p}}_{\mathrm{GO}}\ \mathbf{S})) & = 1 + \mathscr{S}(\mathbf{p}^{\mathrm{p}}_{\mathrm{GO}}) \\
\mathscr{S}((\varrho\ \langle \theta^{\mathrm{r}}, \mathbf{A} \rangle.\ \mathbf{p}^{\mathrm{p}}_{\mathrm{GO}})) & = 1 + \mathscr{S}(\mathbf{p}^{\mathrm{p}}_{\mathrm{GO}}) \\
\mathscr{S}((\text{loop } \mathbf{p}^{\mathrm{p}}_{\mathrm{GO}})) & = 2 + \mathscr{S}(\mathbf{p}^{\mathrm{p}}_{\mathrm{GO}}) + \mathscr{S}(\mathbf{p}^{\mathrm{p}}_{\mathrm{GO}}) \\
\mathscr{S}((\overline{\text{loop}}\ \mathbf{p}^{\mathrm{p}}_{\mathrm{GO}}\ \mathbf{q}^{\mathrm{p}}_{\mathrm{GO}})) & = 1 + \mathscr{S}(\mathbf{p}^{\mathrm{p}}_{\mathrm{GO}}) + \mathscr{S}(\mathbf{q}^{\mathrm{p}}_{\mathrm{GO}})
\end{array}
$$

Definition: $\mathit{closed}(\mathbf{p}^{\mathrm{p}}_{\mathrm{GO}})$

$$
\frac{\mathcal{FV}((\varrho\ \langle \theta^{\mathrm{r}}, \mathrm{GO} \rangle.\ \mathbf{q}^{\mathrm{p}})) = \varnothing}{\mathit{closed}((\varrho\ \langle \theta^{\mathrm{r}}, \mathrm{GO} \rangle.\ \mathbf{q}^{\mathrm{p}}))}
$$

Definition: $\mathit{nc}(\mathbf{p}^{\mathrm{p}}, \theta, \theta^{\mathfrak{c}})$

$$
\frac{\mathit{nc}\text{-}\kappa(\mathbf{p}^{\mathrm{p}}, \theta, \theta^{\mathfrak{c}}) \qquad \mathit{nc}\text{-}\mathcal{S}(\mathbf{p}^{\mathrm{p}}, \theta, \theta^{\mathfrak{c}}) \qquad \mathit{nc}\text{-}\mathcal{N}(\mathbf{p}^{\mathrm{p}}, \theta, \theta^{\mathfrak{c}})}{\mathit{nc}(\mathbf{p}^{\mathrm{p}}, \theta, \theta^{\mathfrak{c}})}
$$

Definition: $\mathit{nc}\text{-}\mathcal{S}(\mathbf{p}^{\mathrm{p}}, \theta, \theta^{\mathfrak{c}})$

$$\frac{\forall\, \mathbf{S} \in \mathit{Can}^{\mathrm{S}}(\mathbf{p}^{\mathrm{p}}, \theta),\ \theta(\mathbf{S}) = \bot \Rightarrow \theta^{\varsigma}(\mathbf{S}^{\mathrm{i}}) = \theta^{\varsigma}(\mathbf{S}^{\mathrm{o}}) = \bot}{\mathit{nc}\text{-}\mathcal{S}(\mathbf{p}^{\mathrm{p}}, \theta, \theta^{\varsigma})}$$

Definition: $\mathit{nc}\text{-}\kappa(\mathbf{p}^{\mathrm{p}}, \theta, \theta^{\varsigma})$

$$\frac{\forall\, \mathbf{n} \in \mathit{Can}^{\mathrm{K}}(\mathbf{p}^{\mathrm{p}}, \theta),\ \theta^{\varsigma}(\mathrm{K}\mathbf{n}) = \bot}{\mathit{nc}\text{-}\kappa(\mathbf{p}^{\mathrm{p}}, \theta, \theta^{\varsigma})}$$

Definition: $\mathit{sub}(\mathbf{p}^{\mathrm{p}}, \mathbf{q}^{\mathrm{p}}, \theta^{\varsigma})$

When ς *is the compilation of* $\mathbf{p}^{\mathrm{p}}$*, get the substate of* θ^{ς} *corresponding to the subterm* $\mathbf{q}^{\mathrm{p}}$.

Definition: *nc-r*($\mathbf{p}^{\mathrm{p}}$, θ, θ^{c})

$$\frac{}{\mathit{nc\text{-}r}(\text{nothing}, \theta, \theta^{\mathsf{c}})}\,[\text{nothing}] \quad \frac{}{\mathit{nc\text{-}r}((\text{exit } \mathbf{n}), \theta, \theta^{\mathsf{c}})}\,[\text{exit}]$$

$$\frac{}{\mathit{nc\text{-}r}((\text{emit } \mathbf{S}), \theta, \theta^{\mathsf{c}})}\,[\text{emit}] \quad \frac{}{\mathit{nc\text{-}r}(\text{pause}, \theta, \theta^{\mathsf{c}})}\,[\text{pause}]$$

$$\frac{\mathit{nc}(\mathbf{p}^{\mathsf{p}}, \theta, \mathit{sub}((\text{trap } \mathbf{p}^{\mathsf{p}}), \mathbf{p}^{\mathsf{p}}, \theta^{\mathsf{c}}))}{\mathit{nc\text{-}r}((\text{trap } \mathbf{p}^{\mathsf{p}}), \theta, \theta^{\mathsf{c}})}\,[\text{trap}] \quad \frac{\mathit{nc}(\mathbf{p}^{\mathsf{p}}, \theta, \mathit{sub}((\text{suspend } \mathbf{p}^{\mathsf{p}}\ \mathbf{S}), \mathbf{p}^{\mathsf{p}}, \theta^{\mathsf{c}}))}{\mathit{nc\text{-}r}((\text{suspend } \mathbf{p}^{\mathsf{p}}\ \mathbf{S}), \theta, \theta^{\mathsf{c}})}\,[\text{suspend}]$$

$$\frac{\theta(\mathbf{S}) = 0 \qquad \mathit{nc}(\mathbf{q}^{\mathsf{p}}, \theta, \mathit{sub}((\text{if } \mathbf{S}\ \mathbf{p}^{\mathsf{p}}\ \mathbf{q}^{\mathsf{p}}), \mathbf{q}^{\mathsf{p}}, \theta^{\mathsf{c}}))}{\mathit{nc\text{-}r}((\text{if } \mathbf{S}\ \mathbf{p}^{\mathsf{p}}\ \mathbf{q}^{\mathsf{p}}), \theta, \theta^{\mathsf{c}})}\,[\text{if-0}] \quad \frac{\theta(\mathbf{S}) = 1 \qquad \mathit{nc}(\mathbf{p}^{\mathsf{p}}, \theta, \mathit{sub}((\text{if } \mathbf{S}\ \mathbf{p}^{\mathsf{p}}\ \mathbf{q}^{\mathsf{p}}), \mathbf{p}^{\mathsf{p}}, \theta^{\mathsf{c}}))}{\mathit{nc\text{-}r}((\text{if } \mathbf{S}\ \mathbf{p}^{\mathsf{p}}\ \mathbf{q}^{\mathsf{p}}), \theta, \theta^{\mathsf{c}})}\,[\text{if-1}]$$

$$\frac{\theta(\mathbf{S}) = \bot \qquad \mathit{nc}(\mathbf{p}^{\mathsf{p}}, \theta, \mathit{sub}((\text{if } \mathbf{S}\ \mathbf{p}^{\mathsf{p}}\ \mathbf{q}^{\mathsf{p}}), \mathbf{p}^{\mathsf{p}}, \theta^{\mathsf{c}})) \qquad \mathit{nc}(\mathbf{q}^{\mathsf{p}}, \theta, \mathit{sub}((\text{if } \mathbf{S}\ \mathbf{p}^{\mathsf{p}}\ \mathbf{q}^{\mathsf{p}}), \mathbf{q}^{\mathsf{p}}, \theta^{\mathsf{c}}))}{\mathit{nc\text{-}r}((\text{if } \mathbf{S}\ \mathbf{p}^{\mathsf{p}}\ \mathbf{q}^{\mathsf{p}}), \theta, \theta^{\mathsf{c}})}\,[\text{if-}\bot]$$

$$\frac{0 \notin \mathit{Can}^{\mathrm{K}}(\mathbf{p}^{\mathsf{p}}, \theta) \qquad \mathit{nc}(\mathbf{p}^{\mathsf{p}}, \theta, \mathit{sub}((\text{seq } \mathbf{p}^{\mathsf{p}}\ \mathbf{q}^{\mathsf{p}}), \mathbf{p}^{\mathsf{p}}, \theta^{\mathsf{c}}))}{\mathit{nc\text{-}r}((\text{seq } \mathbf{p}^{\mathsf{p}}\ \mathbf{q}^{\mathsf{p}}), \theta, \theta^{\mathsf{c}})}\,[\text{seq-}\neg 0]$$

$$\frac{0 \in \mathit{Can}^{\mathrm{K}}(\mathbf{p}^{\mathsf{p}}, \theta) \qquad \mathit{nc}(\mathbf{p}^{\mathsf{p}}, \theta, \mathit{sub}((\text{seq } \mathbf{p}^{\mathsf{p}}\ \mathbf{q}^{\mathsf{p}}), \mathbf{p}^{\mathsf{p}}, \theta^{\mathsf{c}})) \qquad \mathit{nc}(\mathbf{q}^{\mathsf{p}}, \theta, \mathit{sub}((\text{seq } \mathbf{p}^{\mathsf{p}}\ \mathbf{q}^{\mathsf{p}}), \mathbf{q}^{\mathsf{p}}, \theta^{\mathsf{c}}))}{\mathit{nc\text{-}r}((\text{seq } \mathbf{p}^{\mathsf{p}}\ \mathbf{q}^{\mathsf{p}}), \theta, \theta^{\mathsf{c}})}\,[\text{seq-0}]$$

$$\frac{\mathit{nc}(\mathbf{p}^{\mathsf{p}}, \theta, \mathit{sub}((\text{par } \mathbf{p}^{\mathsf{p}}\ \mathbf{q}^{\mathsf{p}}), \mathbf{p}^{\mathsf{p}}, \theta^{\mathsf{c}})) \qquad \mathit{nc}(\mathbf{q}^{\mathsf{p}}, \theta, \mathit{sub}((\text{par } \mathbf{p}^{\mathsf{p}}\ \mathbf{q}^{\mathsf{p}}), \mathbf{q}^{\mathsf{p}}, \theta^{\mathsf{c}}))}{\mathit{nc\text{-}r}((\text{par } \mathbf{p}^{\mathsf{p}}\ \mathbf{q}^{\mathsf{p}}), \theta, \theta^{\mathsf{c}})}\,[\text{par}]$$

$$\frac{\mathit{nc\text{-}r}(\mathbf{p}^{\mathsf{p}}, \theta, \mathit{sub}((\varrho\ \langle\{\}, \text{WAIT}\rangle.\ \mathbf{p}^{\mathsf{p}}), \mathbf{p}^{\mathsf{p}}, \theta^{\mathsf{c}}))}{\mathit{nc\text{-}r}((\varrho\ \langle\{\}, \text{WAIT}\rangle.\ \mathbf{p}^{\mathsf{p}}), \theta, \theta^{\mathsf{c}})}\,[\rho\text{-}\{\}]$$

$$\frac{\mathbf{S} \in \mathit{dom}(\theta) \qquad \theta^{\mathrm{r}}(\mathbf{S}) = \bot \qquad \mathbf{S} \notin \mathit{Can}^{\mathrm{S}}_{\varrho}((\varrho\ \langle(\theta^{\mathrm{r}} \setminus \{\mathbf{S}\}), \text{WAIT}\rangle.\ \mathbf{p}^{\mathsf{p}}), \theta \leftarrow \{\ \mathbf{S} \mapsto \bot\ \}) \qquad \mathit{nc\text{-}r}((\varrho\ \langle(\theta^{\mathrm{r}} \setminus \{\mathbf{S}\}), \text{WAIT}\rangle.\ \mathbf{p}^{\mathsf{p}}), \theta \leftarrow \{\ \mathbf{S} \mapsto 0\ \}, \mathit{sub}((\varrho\ \langle\theta^{\mathrm{r}}, \text{WAIT}\rangle.\ \mathbf{p}^{\mathsf{p}}), (\varrho\ \langle(\theta^{\mathrm{r}} \setminus \{\mathbf{S}\}), \text{WAIT}\rangle.\ \mathbf{p}^{\mathsf{p}}), \theta^{\mathsf{c}}))}{\mathit{nc\text{-}r}((\varrho\ \langle\theta^{\mathrm{r}}, \text{WAIT}\rangle.\ \mathbf{p}^{\mathsf{p}}), \theta, \theta^{\mathsf{c}})}\,[\rho\text{-}0]$$

$$\frac{\mathbf{S} \in \mathit{dom}(\theta) \qquad \theta(\mathbf{S}) \neq \bot \qquad \mathit{nc\text{-}r}((\varrho\ \langle(\theta^{\mathrm{r}} \setminus \{\mathbf{S}\}), \text{WAIT}\rangle.\ \mathbf{p}^{\mathsf{p}}), \theta \leftarrow \{\ \mathbf{S} \mapsto 1\ \}, \mathit{sub}((\varrho\ \langle\theta^{\mathrm{r}}, \text{WAIT}\rangle.\ \mathbf{p}^{\mathsf{p}}), (\varrho\ \langle(\theta^{\mathrm{r}} \setminus \{\mathbf{S}\}), \text{WAIT}\rangle.\ \mathbf{p}^{\mathsf{p}}), \theta^{\mathsf{c}}))}{\mathit{nc\text{-}r}((\varrho\ \langle\theta^{\mathrm{r}}, \text{WAIT}\rangle.\ \mathbf{p}^{\mathsf{p}}), \theta, \theta^{\mathsf{c}})}\,[\rho\text{-}\neg\bot]$$

Definition: $\vdash_{\mathrm{CB}} \mathbf{p}$

$$\frac{}{\vdash_{\mathrm{CB}} \text{nothing}}\,[\text{nothing}] \quad \frac{}{\vdash_{\mathrm{CB}} \text{pause}}\,[\text{pause}] \quad \frac{}{\vdash_{\mathrm{CB}} (\text{emit } \mathbf{S})}\,[\text{emit}]$$

$$\frac{}{\vdash_{\mathrm{CB}} (\text{exit } \mathbf{n})}\,[\text{exit}] \quad \frac{\vdash_{\mathrm{CB}} \mathbf{p}}{\vdash_{\mathrm{CB}} (\text{trap } \mathbf{p})}\,[\text{trap}]$$

$$\frac{\vdash_{\mathrm{CB}} \mathbf{p}}{\vdash_{\mathrm{CB}} (\text{signal } \mathbf{S}\ \mathbf{p})}\,[\text{signal}] \quad \frac{\vdash_{\mathrm{CB}} \mathbf{p} \qquad \vdash_{\mathrm{CB}} \mathbf{q}}{\vdash_{\mathrm{CB}} (\text{if } \mathbf{S}\ \mathbf{p}\ \mathbf{q})}\,[\text{if}]$$

$$\frac{\mathcal{BV}(\mathbf{p}) \cap \mathcal{FV}(\mathbf{q}) = \varnothing \qquad \vdash_{\mathrm{CB}} \mathbf{p} \qquad \vdash_{\mathrm{CB}} \mathbf{q}}{\vdash_{\mathrm{CB}} (\text{seq } \mathbf{p}\ \mathbf{q})}\,[\text{seq}] \quad \frac{\{\,\mathbf{S}\,\} \cap \mathcal{BV}(\mathbf{p}) = \varnothing \qquad \vdash_{\mathrm{CB}} \mathbf{p}}{\vdash_{\mathrm{CB}} (\text{suspend } \mathbf{p}\ \mathbf{S})}\,[\text{suspend}] \quad \frac{\vdash_{\mathrm{CB}} \mathbf{p}}{\vdash_{\mathrm{CB}} (\varrho\,\langle \theta^{\mathrm{r}}, \mathbf{A}\rangle.\ \mathbf{p})}\,[\rho]$$

$$\frac{\mathcal{BV}(\mathbf{p}) \cap \mathcal{BV}(\mathbf{q}) = \varnothing \qquad \mathcal{FV}(\mathbf{p}) \cap \mathcal{BV}(\mathbf{q}) = \varnothing \qquad \mathcal{BV}(\mathbf{p}) \cap \mathcal{FV}(\mathbf{q}) = \varnothing}{\vdash_{\mathrm{CB}} (\text{par } \mathbf{p}\ \mathbf{q})}\,[\text{par}]$$

A.4. Reduction Strategy

Definition: $\rightharpoonup$

signals	[**signal**]	$(\varrho\ \langle \theta, \mathbf{A}\rangle.\ \mathbf{E}[(\text{signal } \mathbf{S}\ \mathbf{p})]) \rightarrow^{E} (\varrho\ \langle \theta, \mathbf{A}\rangle.\ \mathbf{E}[(\varrho\ \langle \{\, \mathbf{S} \mapsto \bot \,\}, \text{WAIT}\rangle.\ \mathbf{p})])$ if $\mathit{leftmost}(\theta, \mathbf{A}, (\text{signal } \mathbf{S}\ \mathbf{p}), \mathbf{E})$
	[**emit**]	$(\varrho\ \langle \theta, \text{GO}\rangle.\ \mathbf{E}[(\text{emit } \mathbf{S})]) \rightarrow^{E} (\varrho\ \langle (\theta \leftarrow \{\, \mathbf{S} \mapsto 1 \,\}), \text{GO}\rangle.\ \mathbf{E}[\text{nothing}])$ if $\mathit{leftmost}(\theta, \text{GO}, (\text{emit } \mathbf{S}), \mathbf{E})$, $\mathbf{S} \in \mathit{dom}(\theta)$
	[**is-present**]	$(\varrho\ \langle \theta, \mathbf{A}\rangle.\ \mathbf{E}[(\text{if } \mathbf{S}\ \mathbf{p}\ \mathbf{q})]) \rightarrow^{E} (\varrho\ \langle \theta, \mathbf{A}\rangle.\ \mathbf{E}[\mathbf{p}])$ if $\mathit{leftmost}(\theta, \mathbf{A}, (\text{if } \mathbf{S}\ \mathbf{p}\ \mathbf{q}), \mathbf{E})$, $\theta(\mathbf{S}) = 1$
	[**is-absent**]	$(\varrho\ \langle \theta, \mathbf{A}\rangle.\ \mathbf{E}[(\text{if } \mathbf{S}\ \mathbf{p}\ \mathbf{q})]) \rightarrow^{E} (\varrho\ \langle \theta, \mathbf{A}\rangle.\ \mathbf{E}[\mathbf{q}])$ if $\mathit{leftmost}(\theta, \mathbf{A}, (\text{if } \mathbf{S}\ \mathbf{p}\ \mathbf{q}), \mathbf{E})$, $\mathbf{S} \in \mathit{dom}(\theta)$, $\theta(\mathbf{S}) = \bot$, $\mathbf{S} \notin \mathit{Can}^{S}_{\varrho}((\varrho\ \langle \theta, \mathbf{A}\rangle.\ \mathbf{E}[(\text{if } \mathbf{S}\ \mathbf{p}\ \mathbf{q})]), \{\})$
shared variables	[**shared**]	$(\varrho\ \langle \theta, \mathbf{A}\rangle.\ \mathbf{E}[(\text{shared } \mathbf{s} := \mathbf{e}\ \mathbf{p})]) \rightarrow^{E} (\varrho\ \langle \theta, \mathbf{A}\rangle.\ \mathbf{E}[(\varrho\ \langle \{\, \mathbf{s} \mapsto \langle \mathbf{n}, \text{old}\rangle \,\}, \text{WAIT}\rangle.\ \mathbf{p})])$ if $\mathit{leftmost}(\theta, \mathbf{A}, (\text{shared } \mathbf{s} := \mathbf{e}\ \mathbf{p}), \mathbf{E})$, $\mathit{FV}(\mathbf{e}) \subseteq \mathit{dom}(\theta)$, $\forall\, \mathbf{s} \in \mathit{FV}(\mathbf{e}).\ \mathbf{s} \notin \mathit{Can}^{sh}_{\varrho}((\varrho\ \langle \theta, \mathbf{A}\rangle\ \mathbf{E}[(\text{shared } \mathbf{s} := \mathbf{e}\ \mathbf{p})]), \cdot)$, $\mathbf{n} = \mathit{eval}^{H}(\mathbf{e}, \theta)$
	[**set-old**]	$(\varrho\ \langle \theta, \text{GO}\rangle.\ \mathbf{E}[(\mathrel{+}= \mathbf{s}\ \mathbf{e})]) \rightarrow^{E} (\varrho\ \langle (\theta \leftarrow \{\, \mathbf{s} \mapsto \langle \mathit{eval}^{H}(\mathbf{e}, \theta), \text{new}\rangle \,\}), \text{GO}\rangle.\ \mathbf{E}[\text{nothing}])$ if $\mathit{leftmost}(\theta, \text{GO}, (\mathrel{+}= \mathbf{s}\ \mathbf{e}), \mathbf{E})$, $\theta(\mathbf{s}) = \langle \_, \text{old}\rangle$, $\mathit{FV}(\mathbf{e}) \subseteq \mathit{dom}(\theta)$, $\forall\, \mathbf{s} \in \mathit{FV}(\mathbf{e}).\ \mathbf{s} \notin \mathit{Can}^{sh}_{\varrho}((\varrho\ \langle \theta, \text{GO}\rangle\ \mathbf{E}[(\mathrel{+}= \mathbf{s}\ \mathbf{e})]), \cdot)$
	[**set-new**]	$(\varrho\ \langle \theta, \text{GO}\rangle.\ \mathbf{E}[(\mathrel{+}= \mathbf{s}\ \mathbf{e})]) \rightarrow^{E} (\varrho\ \langle (\theta \leftarrow \{\, \mathbf{s} \mapsto \langle \mathbf{n} + \mathit{eval}^{H}(\mathbf{e}, \theta), \text{new}\rangle \,\}), \text{GO}\rangle.\ \mathbf{E}[\text{nothing}])$ if $\mathit{leftmost}(\theta, \text{GO}, (\mathrel{+}= \mathbf{s}\ \mathbf{e}), \mathbf{E})$, $\theta(\mathbf{s}) = \langle \mathbf{n}, \text{new}\rangle$, $\mathit{FV}(\mathbf{e}) \subseteq \mathit{dom}(\theta)$, $\forall\, \mathbf{s} \in \mathit{FV}(\mathbf{e}).\ \mathbf{s} \notin \mathit{Can}^{sh}_{\varrho}((\varrho\ \langle \theta, \text{GO}\rangle\ \mathbf{E}[(\mathrel{+}= \mathbf{s}\ \mathbf{e})]), \cdot)$
sequential variables	[**var**]	$(\varrho\ \langle \theta, \mathbf{A}\rangle.\ \mathbf{E}[(\text{var } \mathbf{x} := \mathbf{e}\ \mathbf{p})]) \rightarrow^{E} (\varrho\ \langle \theta, \mathbf{A}\rangle.\ \mathbf{E}[(\varrho\ \langle \{\, \mathbf{x} \mapsto \mathit{eval}^{H}(\mathbf{e}, \theta) \,\}, \text{WAIT}\rangle.\ \mathbf{p})])$ if $\mathit{leftmost}(\theta, \mathbf{A}, (\text{var } \mathbf{x} := \mathbf{e}\ \mathbf{p}), \mathbf{E})$, $\mathit{FV}(\mathbf{e}) \subseteq \mathit{dom}(\theta)$, $\forall\, \mathbf{s} \in \mathit{FV}(\mathbf{e}).\ \mathbf{s} \notin \mathit{Can}^{sh}_{\varrho}((\varrho\ \langle \theta, \mathbf{A}\rangle\ \mathbf{E}[(\text{var } \mathbf{x} := \mathbf{e}\ \mathbf{p})]), \cdot)$
	[**set-var**]	$(\varrho\ \langle \theta, \mathbf{A}\rangle.\ \mathbf{E}[(:= \mathbf{x}\ \mathbf{e})]) \rightarrow^{E} (\varrho\ \langle (\theta \leftarrow \{\, \mathbf{x} \mapsto \mathit{eval}^{H}(\mathbf{e}, \theta) \,\}), \mathbf{A}\rangle.\ \mathbf{E}[\text{nothing}])$ if $\mathit{leftmost}(\theta, \mathbf{A}, (:= \mathbf{x}\ \mathbf{e}), \mathbf{E})$, $\mathbf{x} \in \mathit{dom}(\theta)$, $\mathit{FV}(\mathbf{e}) \subseteq \mathit{dom}(\theta)$, $\forall\, \mathbf{s} \in \mathit{FV}(\mathbf{e}).\ \mathbf{s} \notin \mathit{Can}^{sh}_{\varrho}((\varrho\ \langle \theta, \mathbf{A}\rangle\ \mathbf{E}[(:= \mathbf{x}\ \mathbf{e})]), \cdot)$
	[**if-true**]	$(\varrho\ \langle \theta, \text{GO}\rangle.\ \mathbf{E}[(\text{if!0 } \mathbf{x}\ \mathbf{p}\ \mathbf{q})]) \rightarrow^{E} (\varrho\ \langle \theta, \text{GO}\rangle.\ \mathbf{E}[\mathbf{p}])$ if $\mathit{leftmost}(\theta, \text{GO}, (\text{if!0 } \mathbf{x}\ \mathbf{p}\ \mathbf{q}), \mathbf{E})$, $\mathbf{x} \in \mathit{dom}(\theta)$, $\theta(\mathbf{x}) \neq 0$
	[**if-false**]	$(\varrho\ \langle \theta, \text{GO}\rangle.\ \mathbf{E}[(\text{if!0 } \mathbf{x}\ \mathbf{p}\ \mathbf{q})]) \rightarrow^{E} (\varrho\ \langle \theta, \text{GO}\rangle.\ \mathbf{E}[\mathbf{q}])$ if $\mathit{leftmost}(\theta, \text{GO}, (\text{if!0 } \mathbf{x}\ \mathbf{p}\ \mathbf{q}), \mathbf{E})$, $\theta(\mathbf{x}) = 0$
ϱ	[**merge**]	$(\varrho\ \langle \theta_1, \mathbf{A}_1\rangle.\ \mathbf{E}[(\varrho\ \langle \theta_2, \mathbf{A}_2\rangle.\ \mathbf{p})]) \rightarrow^{E} (\varrho\ \langle (\theta_1 \leftarrow \theta_2), \mathbf{A}_1\rangle.\ \mathbf{E}[\mathbf{p}])$ if $\mathit{leftmost}(\theta_1, \mathbf{A}_1, (\varrho\ \langle \theta_2, \mathbf{A}_2\rangle.\ \mathbf{p}), \mathbf{E})$, $\mathbf{A}_1 \geq \mathbf{A}_2$
seq	[**seq-done**]	$(\varrho\ \langle \theta, \mathbf{A}\rangle.\ \mathbf{E}[(\text{seq nothing } \mathbf{q})]) \rightarrow^{E} (\varrho\ \langle \theta, \mathbf{A}\rangle.\ \mathbf{E}[\mathbf{q}])$ if $\mathit{leftmost}(\theta, \mathbf{A}, (\text{seq nothing } \mathbf{q}), \mathbf{E})$
	[**seq-exit**]	$(\varrho\ \langle \theta, \mathbf{A}\rangle.\ \mathbf{E}[(\text{seq (exit } \mathbf{n})\ \mathbf{q})]) \rightarrow^{E} (\varrho\ \langle \theta, \mathbf{A}\rangle.\ \mathbf{E}[(\text{exit } \mathbf{n})])$ if $\mathit{leftmost}(\theta, \mathbf{A}, (\text{seq (exit } \mathbf{n})\ \mathbf{q}), \mathbf{E})$
trap	[**trap**]	$(\varrho\ \langle \theta, \mathbf{A}\rangle.\ \mathbf{E}[(\text{trap } \mathbf{p}^{S})]) \rightarrow^{E} (\varrho\ \langle \theta, \mathbf{A}\rangle.\ \mathbf{E}[\downarrow^{p} \mathbf{p}^{S}])$ if $\mathit{leftmost}(\theta, \mathbf{A}, (\text{trap } \mathbf{p}^{S}), \mathbf{E})$
par	[**parr**]	$(\varrho\ \langle \theta, \mathbf{A}\rangle.\ \mathbf{E}[(\text{par } \mathbf{p}^{S}\ \mathbf{p}^{D})]) \rightarrow^{E} (\varrho\ \langle \theta, \mathbf{A}\rangle.\ \mathbf{E}[\mathbf{p}^{S} \sqcap_{\parallel} \mathbf{p}^{D}])$ if $\mathit{leftmost}(\theta, \mathbf{A}, (\text{par } \mathbf{p}^{S}\ \mathbf{p}^{D}), \mathbf{E})$
	[**parl**]	$(\varrho\ \langle \theta, \mathbf{A}\rangle.\ \mathbf{E}[(\text{par } \hat{\mathbf{p}}\ \mathbf{p}^{S})]) \rightarrow^{E} (\varrho\ \langle \theta, \mathbf{A}\rangle.\ \mathbf{E}[\hat{\mathbf{p}} \sqcap_{\parallel} \mathbf{p}^{S}])$ if $\mathit{leftmost}(\theta, \mathbf{A}, (\text{par } \hat{\mathbf{p}}\ \mathbf{p}^{S}), \mathbf{E})$
	[**suspend**]	$(\varrho\ \langle \theta, \mathbf{A}\rangle.\ \mathbf{E}[(\text{suspend } \mathbf{p}^{S}\ \mathbf{S})]) \rightarrow^{E} (\varrho\ \langle \theta, \mathbf{A}\rangle.\ \mathbf{E}[\mathbf{p}^{S}])$ if $\mathit{leftmost}(\theta, \mathbf{A}, (\text{suspend } \mathbf{p}^{S}\ \mathbf{S}), \mathbf{E})$
loop	[**loop**]	$(\varrho\ \langle \theta, \mathbf{A}\rangle.\ \mathbf{E}[(\text{loop } \mathbf{p})]) \rightarrow^{E} (\varrho\ \langle \theta, \mathbf{A}\rangle.\ \mathbf{E}[(\overline{\text{loop}}\ \mathbf{p}\ \mathbf{p})])$ if $\mathit{leftmost}(\theta, \mathbf{A}, (\text{loop } \mathbf{p}), \mathbf{E})$
	[**loop^stop-exit**]	$(\varrho\ \langle \theta, \mathbf{A}\rangle.\ \mathbf{E}[(\overline{\text{loop}}\ (\text{exit } \mathbf{n})\ \mathbf{q})]) \rightarrow^{E} (\varrho\ \langle \theta, \mathbf{A}\rangle.\ \mathbf{E}[(\text{exit } \mathbf{n})])$ if $\mathit{leftmost}(\theta, \mathbf{A}, (\overline{\text{loop}}\ (\text{exit } \mathbf{n})\ \mathbf{q}), \mathbf{E})$

Definition: $\mathit{leftmost}(\theta^{r}, \mathbf{A}, \mathbf{p}, \mathbf{E})$

$$\frac{\mathit{leftmost}^{*}(\theta, \mathbf{A}, \mathbf{p}, \bigcirc, \mathbf{E})}{\mathit{leftmost}(\theta, \mathbf{A}, \mathbf{p}, \mathbf{E})}$$

Definition: $\mathit{leftmost}^{*}(\theta^{r}, \mathbf{A}, \mathbf{p}, \mathbf{E}_1, \mathbf{E}_2)$

$$\frac{}{\mathit{leftmost}^{*}(\theta, \mathbf{A}, \mathbf{p}_o, \mathbf{E}, \bigcirc)}\ [\text{hole}]$$

$$\frac{\mathit{leftmost}^{*}(\theta, \mathbf{A}, \mathbf{p}_o, \mathbf{E}[(\text{seq} \bigcirc \mathbf{q})], \mathbf{E})}{\mathit{leftmost}^{*}(\theta, \mathbf{A}, \mathbf{p}_o, \mathbf{E}, (\text{seq } \mathbf{E}\ \mathbf{q}))}\ [\text{seq}] \qquad \frac{\mathit{leftmost}^{*}(\theta, \mathbf{A}, \mathbf{p}_o, \mathbf{E}[(\overline{\text{loop}} \bigcirc \mathbf{q})], \mathbf{E})}{\mathit{leftmost}^{*}(\theta, \mathbf{A}, \mathbf{p}_o, \mathbf{E}, (\overline{\text{loop}}\ \mathbf{E}\ \mathbf{q}))}\ [\text{loop\^{}stop}]$$

$$\frac{\mathit{leftmost}^{*}(\theta, \mathbf{A}, \mathbf{p}_o, \mathbf{E}[(\text{par} \bigcirc \mathbf{q})], \mathbf{E})}{\mathit{leftmost}^{*}(\theta, \mathbf{A}, \mathbf{p}_o, \mathbf{E}, (\text{par } \mathbf{E}\ \mathbf{q}))}\ [\text{parl}] \qquad \frac{\mathit{leftmost}^{*}(\theta, \mathbf{A}, \mathbf{p}_o, \mathbf{E}[(\text{par } \mathbf{p}^{D} \bigcirc)], \mathbf{E})}{\mathit{leftmost}^{*}(\theta, \mathbf{A}, \mathbf{p}_o, \mathbf{E}, (\text{par } \mathbf{p}^{D}\ \mathbf{E}))}\ [\text{par-done}]$$

$$\frac{\mathit{leftmost}^{*}(\theta, \mathbf{A}, \mathbf{p}_o, \mathbf{E}[(\text{par } \mathbf{p} \bigcirc)], \mathbf{E}) \qquad \theta; \mathbf{A}; \mathbf{E}[(\text{par} \bigcirc \mathbf{E}[\mathbf{p}_o])] \vdash_{B} \mathbf{p}}{\mathit{leftmost}^{*}(\theta, \mathbf{A}, \mathbf{p}_o, \mathbf{E}, (\text{par } \mathbf{p}\ \mathbf{E}))}\ [\text{par-blocked}]$$

$$\frac{\mathit{leftmost}^{*}(\theta, \mathbf{A}, \mathbf{p}_o, \mathbf{E}[(\text{suspend} \bigcirc \mathbf{S})], \mathbf{E})}{\mathit{leftmost}^{*}(\theta, \mathbf{A}, \mathbf{p}_o, \mathbf{E}, (\text{suspend } \mathbf{E}\ \mathbf{S}))}\ [\text{suspend}] \qquad \frac{\mathit{leftmost}^{*}(\theta, \mathbf{A}, \mathbf{p}_o, \mathbf{E}[(\text{trap} \bigcirc)], \mathbf{E})}{\mathit{leftmost}^{*}(\theta, \mathbf{A}, \mathbf{p}_o, \mathbf{E}, (\text{trap } \mathbf{E}))}\ [\text{trap}]$$

APPENDIX B

Proofs

B.1. Core Theorems

This section contains the core proofs to justify soundness, adequacy, and consistency of the calculus with respect to the compilation function. The core theorem for soundness is theorem 29 (Soundness), however the most informative theorem is theorem 34 (Soundness of Step). The core theorem for adequacy is theorem 30 (Computational Adequacy), but the most informative theorems are lemma 57 (Strongly Canonicalizing), lemma 60 (Esterel Value is Circuit Value), and lemma 67 (Adequacy of Can). The theorem for consistency is theorem 31 (Consistency of Eval), which in this case is essentially a corollary of theorem 30 (Computational Adequacy). Some proofs are proved using Circuitous in Jypter notebooks using the Racket kernel: https://github.com/rmculpepper/iracket. These Notebooks may be found in the repository for this Dissertation.

Theorem 29 (Soundness)**.**

For all $\mathbf{p}^{\mathrm{P}}$ *and* $\mathbf{q}^{\mathrm{P}}$, *if* $\vdash_{\mathrm{CB}} \mathbf{p}^{\mathrm{P}}$, $\mathbf{p}^{\mathrm{P}} \equiv^{\mathrm{E}} \mathbf{q}^{\mathrm{P}}$, $[\![\mathbf{p}^{\mathrm{P}}]\!](\mathrm{SEL}) \simeq 0$, *and* $[\![\mathbf{q}^{\mathrm{P}}]\!](\mathrm{SEL}) \simeq 0$ *then* $[\![\mathbf{p}^{\mathrm{P}}]\!] \simeq^{C} [\![\mathbf{q}^{\mathrm{P}}]\!]$

Interpretation. This theorem says that, at least for the first instant/cycle $\equiv^{\mathrm{E}}$ agrees with $\simeq^{C}$. Therefore any change to a program which can be proven correct by $\equiv^{\mathrm{E}}$ is correct under $\simeq^{C}$.

Proof.

Cases of $\equiv^{\mathrm{E}}$:

Case 1:sym

In this case we have $\mathbf{p}^{\mathsf{p}} \equiv^{\mathrm{E}} \mathbf{q}^{\mathsf{p}}$ because $\mathbf{q}^{\mathsf{p}} \equiv^{\mathrm{E}} \mathbf{p}^{\mathsf{p}}$. This case follows by induction and by lemma 35 (Symmetry of circuit contextual equality).

Case 2:trans

In this case we have $\mathbf{p}^{\mathsf{p}} \equiv^{\mathrm{E}} \mathbf{q}^{\mathsf{p}}$ there exists some $\mathbf{r}^{\mathsf{p}}$ where $\mathbf{p}^{\mathsf{p}} \equiv^{\mathrm{E}} \mathbf{r}^{\mathsf{p}}$ and $\mathbf{r}^{\mathsf{p}} \equiv^{\mathrm{E}} \mathbf{q}^{\mathsf{p}}$. This case case follows induction and by lemma 36 (Transitivity of circuit contextual equality).

Case 3:refl

In this case we have $\mathbf{p}^{\mathsf{p}} \equiv^{\mathrm{E}} \mathbf{q}^{\mathsf{p}}$ because $\mathbf{p}^{\mathsf{p}} = \mathbf{q}^{\mathsf{p}}$. This case follows by lemma 37 (Reflexivity of circuit contextual equality).

Case 4:ctx

$\mathbf{p}^{\mathsf{p}} \equiv^{\mathrm{E}} \mathbf{q}^{\mathsf{p}}$ because $\mathbf{p}^{\mathsf{p}} = \mathbf{C}^{\mathsf{p}}[\mathbf{p}^{\mathsf{p}}{}_i]$, $\mathbf{q}^{\mathsf{p}} = \mathbf{C}^{\mathsf{p}}[\mathbf{q}^{\mathsf{p}}{}_i]$, and $\mathbf{p}^{\mathsf{p}}{}_i \equiv^{\mathrm{E}} \mathbf{q}^{\mathsf{p}}{}_i$.

This case follows by lemma 33 (Soundness of context closure), and induction on $\mathbf{p}^{\mathsf{p}}{}_i \equiv^{\mathrm{E}} \mathbf{q}^{\mathsf{p}}{}_i$.

Case 5:step

In this case we have $\mathbf{p}^{\mathsf{p}} \equiv^{\mathrm{E}} \mathbf{q}^{\mathsf{p}}$ because $\mathbf{p}^{\mathsf{p}} \rightharpoonup^{\mathrm{E}} \mathbf{q}^{\mathsf{p}}$. This case is given by theorem 34 (Soundness of Step).

□

Theorem 30 (Computational Adequacy)**.**

For all $\mathbf{p}^{\mathsf{p}}$, $\mathbf{O}$, *if* $\mathit{closed}(\mathbf{p}^{\mathsf{p}}_{\mathrm{GO}})$ *and* $[\![\mathbf{p}^{\mathsf{p}}_{\mathrm{GO}}]\!](\mathrm{SEL}) \simeq 0$ *then*
$\mathit{eval}^{\mathrm{E}}(\mathbf{O}, \mathbf{p}^{\mathsf{p}}_{\mathrm{GO}}) = \langle \theta\ , \mathbf{bool} \rangle$ *if and only if* $\mathit{eval}^{\mathrm{C}}(\mathbf{O}, [\![\mathbf{p}^{\mathsf{p}}_{\mathrm{GO}}]\!]) = \langle \theta\ , \mathbf{bool} \rangle$

Interpretation. This theorem states that the calculus can defined an evaluator which is the same an evaluator for Esterel which we take as the ground-truth semantics.

Proof.

1. By lemma 57 (Strongly Canonicalizing) and lemma 69 (Non-stepping terms are values) we the fact that $\longrightarrow^{E}$ is a subrelation of $\equiv^{E}$, and the fact that $\mathbf{p}^{p}_{GO}$ is closed, we can conclude that there exists some $\mathbf{q} = (\varrho\ \langle \theta^{r}, GO\rangle.\ \mathbf{r}^{p})$, where $\mathbf{p} \equiv^{E} \mathbf{q}$, such that either $\mathbf{r}^{p} \in \mathbf{p}^{D}$ which (by $\mathsf{can_S}$-done, is *complete-wrt*$(\theta^{r}, \mathbf{r}^{p})$, or we have $\theta^{r}; GO; \bigcirc \vdash_{B} \mathbf{r}^{p}$.

2. Cases of (1):

Case 1:$\mathbf{r}^{p} \in \mathbf{p}^{D}$

By the definition of $eval^{E}$, it must return tt for the constructiveness of $\mathbf{p}^{p}$. By lemma 61 (Done is Constructive), $eval^{C}$ must do the same.

By theorem 29 (Soundness), both evaluators must agree on the value of the signal wires, and thus give back the same θ.

Case 2:$\theta^{r}; GO; \bigcirc \vdash_{B} \mathbf{r}^{p}$

The constructiveness of both evaluators follows direclty from lemma 62 (Blocked terms are non-constructive). By theorem 29 (Soundness), both evaluators must agree on the value of the signal wires, and thus give back the same θ.

□

Theorem 31 (Consistency of Eval)**.**

For all $\mathbf{p}^{p}_{GO}$ *and* $\mathbf{O}$, *if* *closed*$(\mathbf{p}^{p}_{GO})$, $eval^{E}(\mathbf{O}, \mathbf{p}^{p}_{GO}) = \langle \theta_1, \mathbf{bool}_1\rangle$,
and $eval^{E}(\mathbf{O}, \mathbf{p}^{p}_{GO}) = \langle \theta_2, \mathbf{bool}_2\rangle$,
then $\langle \theta_1, \mathbf{bool}_1\rangle = \langle \theta_2, \mathbf{bool}_2\rangle$.

Interpretation. This theorem states that $eval^{E}$ is a function.

Proof.

This follows directly from theorem 30 (Computational Adequacy). As theorem 30 (Computational Adequacy) states that the relations defined by $eval/^{\mathrm{E}}$ and $eval/^{\mathrm{C}} \circ [\![\cdot]\!]$, thus as $eval/^{\mathrm{C}}$ and $[\![\cdot]\!]$ are functions, $eval/^{\mathrm{E}}$ must be as well. □

B.2. Soundness

This section contains some of the basic lemma's needed for soundness.

Theorem 32 (Soundness of guarded terms)**.**

For all $\mathbf{p}^{\mathrm{p}}$ *and* $\mathbf{q}^{\mathrm{p}}$, *if* $\vdash_{\mathrm{CB}} \mathbf{p}^{\mathrm{p}}$, $\mathbf{p}^{\mathrm{p}} \equiv^{\mathrm{E}} \mathbf{q}^{\mathrm{p}}$, $[\![\mathbf{p}^{\mathrm{p}}]\!](\mathrm{SEL}) \simeq 0$, $[\![\mathbf{q}^{\mathrm{p}}]\!](\mathrm{SEL}) \simeq 0$, *and then* $[\![\mathbf{p}]\!] \simeq^{C} [\![\mathbf{q}]\!]$

Interpretation. This theorem says that, at least for the first instant/cycle $\equiv^{\mathrm{E}}$ agrees with $\simeq^{C}$. Therefore any change to a program which can be proven correct by $\equiv^{\mathrm{E}}$ is correct under $\simeq^{C}$.

Proof.

Cases of $\equiv^{\mathrm{E}}$:

Case 1:sym

In this case we have $\mathbf{p}^{\mathrm{p}} \equiv^{\mathrm{E}} \mathbf{q}^{\mathrm{p}}$ because $\mathbf{q}^{\mathrm{p}} \equiv^{\mathrm{E}} \mathbf{p}^{\mathrm{p}}$. This case follows by induction and by lemma 35 (Symmetry of circuit contextual equality).

Case 2:trans

In this case we have $\mathbf{p}^{\mathrm{p}} \equiv^{\mathrm{E}} \mathbf{q}^{\mathrm{p}}$ there exists some $\mathbf{r}^{\mathrm{p}}$ where $\mathbf{p}^{\mathrm{p}} \equiv^{\mathrm{E}} \mathbf{r}^{\mathrm{p}}$ and $\mathbf{r}^{\mathrm{p}} \equiv^{\mathrm{E}} \mathbf{q}^{\mathrm{p}}$. This case case follows induction and by lemma 36 (Transitivity of circuit contextual equality).

Case 3:refl

In this case we have $\mathbf{p}^{\mathrm{p}} \equiv^{\mathrm{E}} \mathbf{q}^{\mathrm{p}}$ because $\mathbf{p}^{\mathrm{p}} = \mathbf{q}^{\mathrm{p}}$. This case follows by lemma 37 (Reflexivity of circuit contextual equality).

Case 4:ctx

$\mathbf{p}^{\mathsf{p}} \equiv^{\mathrm{E}} \mathbf{q}^{\mathsf{p}}$ because $\mathbf{p}^{\mathsf{p}} = \mathbf{C}^{\mathsf{p}}[\mathbf{p}^{\mathsf{p}}{}_i]$, $\mathbf{q}^{\mathsf{p}} = \mathbf{C}^{\mathsf{p}}[\mathbf{q}^{\mathsf{p}}{}_i]$, and $\mathbf{p}^{\mathsf{p}}{}_i \equiv^{\mathrm{E}} \mathbf{q}^{\mathsf{p}}{}_i$.

This case follows by lemma 33 (Soundness of context closure), and induction on $\mathbf{p}^{\mathsf{p}}{}_i \equiv^{\mathrm{E}} \mathbf{q}^{\mathsf{p}}{}_i$.

Case 5:step

In this case we have $\mathbf{p}^{\mathsf{p}} \equiv^{\mathrm{E}} \mathbf{q}^{\mathsf{p}}$ because $\mathbf{p}^{\mathsf{p}} \rightharpoonup^{\mathrm{E}} \mathbf{q}^{\mathsf{p}}$. This case is given by theorem 34 (Soundness of Step).

□

Lemma 33 (Soundness of context closure)**.**

For all $\mathbf{C}^{\mathsf{p}}$, $\mathbf{p}^{\mathsf{p}}$ *and* $\mathbf{q}^{\mathsf{p}}$, *if* $[\![\mathbf{p}^{\mathsf{p}}]\!] \simeq^{C} [\![\mathbf{q}^{\mathsf{p}}]\!]$, *then* $[\![\mathbf{C}^{\mathsf{p}}[\mathbf{p}^{\mathsf{p}}]]\!] \simeq^{C} [\![\mathbf{C}^{\mathsf{p}}[\mathbf{q}^{\mathsf{p}}]]\!]$

Proof.

This proof follows direclty by induction on $\mathbf{C}^{\mathsf{p}}$, as $[\![\cdot]\!]$ is defined inductively on the same structure, and will add the same out circuit contexts to each term. □

Theorem 34 (Soundness of Step)**.**

For all $\mathbf{p}^{\mathsf{p}}$ *and* $\mathbf{q}^{\mathsf{p}}$, *if* $\vdash_{\mathrm{CB}} \mathbf{p}^{\mathsf{p}}$, $\mathbf{p}^{\mathsf{p}} \rightharpoonup^{\mathrm{E}} \mathbf{q}^{\mathsf{p}}$, $[\![\mathbf{p}^{\mathsf{p}}]\!](\mathrm{SEL}) = 0$, *and* $[\![\mathbf{q}^{\mathsf{p}}]\!](\mathrm{SEL}) = 0$ *then* $[\![\mathbf{p}^{\mathsf{p}}]\!] \simeq^{C} [\![\mathbf{q}^{\mathsf{p}}]\!]$

Interpretation. This theorem says that, at least for the first instant/cycle $\rightharpoonup^{\mathrm{E}}$ agrees with $\simeq^{C}$.

Proof.

Cases of $\mathbf{p}^{\mathsf{p}} \rightharpoonup^{\mathrm{E}} \mathbf{q}^{\mathsf{p}}$:

Case 1:[**par-swap**]

In this case we have $(\mathrm{par}\ \mathbf{p}^{\mathsf{p}}\ \mathbf{q}^{\mathsf{p}}) \rightharpoonup^{\mathrm{E}} (\mathrm{par}\ \mathbf{q}^{\mathsf{p}}\ \mathbf{p}^{\mathsf{p}})$

This is given by lemma 38 (par-swap is sound).

Case 2:[**par-nothing**]

In this case we have $(\text{par nothing } \mathbf{p}^{D}) \rightarrowtail^{E} \mathbf{p}^{D}$

This is given by lemma 39 (par-nothing is sound).

Case 3:[**par-1exit**]

In this case we have $(\text{par } (\text{exit } \mathbf{n})\ \hat{\mathbf{p}}) \rightarrowtail^{E} (\text{exit } \mathbf{n})$

This is given by lemma 44 (par1-exit is sound).

Case 4:[**par-2exit**]

In this case we have $(\text{par } (\text{exit } \mathbf{n}_1)\ (\text{exit } \mathbf{n}_2)) \rightarrowtail^{E} (\text{exit } \mathit{max}(\mathbf{n}_1, \mathbf{n}_2))$

This is given by lemma 43 (par2-exit is sound).

Case 5:[**is-present**]

In this case we have $(\varrho\ \langle \theta^{r}, \mathbf{A} \rangle.\ \mathbf{E}^{p}[(\text{if } \mathbf{S}\ \mathbf{p}^{p}\ \mathbf{q}^{p})]) \rightarrowtail^{E} (\varrho\ \langle \theta^{r}, \mathbf{A} \rangle.\ \mathbf{E}^{p}[\mathbf{p}^{p}])$

where

$\theta^{r}(\mathbf{S}) = 1$

This is given by lemma 48 (is-present is sound).

Case 6:[**is-absent**]

In this case we have $(\varrho\ \langle \theta^{r}, \mathbf{A} \rangle.\ \mathbf{E}^{p}[(\text{if } \mathbf{S}\ \mathbf{p}^{p}\ \mathbf{q}^{p})]) \rightarrowtail^{E} (\varrho\ \langle \theta^{r}, \mathbf{A} \rangle.\ \mathbf{E}^{p}[\mathbf{q}^{p}])$

where

$\mathbf{S} \in \mathit{dom}(\theta^{r})$

$\theta^{r}(\mathbf{S}) = \bot$

$\mathbf{S} \notin \mathit{Can}^{S}_{\varrho}((\varrho\ \langle \theta^{r}, \mathbf{A} \rangle.\mathbf{E}^{p}[(\text{if } \mathbf{S}\mathbf{p}^{p}\mathbf{q}^{p})]), \{\})$

This is given by lemma 49 (is-absent is sound).

Case 7:[**seq-done**]

In this case we have $(\text{seq nothing } \mathbf{q}^{p}) \rightarrowtail^{E} \mathbf{q}^{p}$

This is given by lemma 42 (seq-done is sound).

CASE 8:[**seq-exit**]

In this case we have $(\text{seq}\ (\text{exit}\ \mathbf{n})\ \mathbf{q}^{\mathrm{p}}) \rightharpoonup^{\mathrm{E}} (\text{exit}\ \mathbf{n})$

This is given by lemma 45 (SEQ-EXIT IS SOUND).

CASE 9:[**suspend**]

In this case we have $(\text{suspend}\ \mathbf{p}^{\mathrm{S}}\ \mathbf{S}) \rightharpoonup^{\mathrm{E}} \mathbf{p}^{\mathrm{S}}$

This is given by lemma 41 (SUSPEND IS SOUND).

CASE 10:[**trap**]

In this case we have $(\text{trap}\ \mathbf{p}^{\mathrm{S}}) \rightharpoonup^{\mathrm{E}} \downarrow^{\mathrm{p}} \mathbf{p}^{\mathrm{S}}$

This is given by lemma 40 (TRAP IS SOUND).

CASE 11:[**signal**]

In this case we have $(\text{signal}\ \mathbf{S}\ \mathbf{p}^{\mathrm{p}}) \rightharpoonup^{\mathrm{E}} (\varrho\ \langle \{\ \mathbf{S} \mapsto \bot\ \}, \text{WAIT}\rangle.\ \mathbf{p}^{\mathrm{p}})$

This is given by lemma 46 (SIGNAL IS SOUND).

CASE 12:[**merge**]

In this case we have $(\varrho\ \langle \theta^{\mathrm{r}}{}_1, \mathbf{A}_1\rangle.\ \mathbf{E}^{\mathrm{p}}[(\varrho\ \langle \theta^{\mathrm{r}}{}_2, \mathbf{A}_2\rangle.\ \mathbf{p}^{\mathrm{p}})]) \rightharpoonup^{\mathrm{E}} (\varrho\ \langle (\theta^{\mathrm{r}}{}_1 \leftarrow \theta^{\mathrm{r}}{}_2), \mathbf{A}_1\rangle.\ \mathbf{E}^{\mathrm{p}}[\mathbf{p}^{\mathrm{p}}])$

where

$\mathbf{A}_1 \geq \mathbf{A}_2$

This is given by lemma 50 (MERGE IS SOUND).

CASE 13:[**emit**]

In this case we have $(\varrho\ \langle \theta^{\mathrm{r}}, \text{GO}\rangle.\ \mathbf{E}^{\mathrm{p}}[(\text{emit}\ \mathbf{S})]) \rightharpoonup^{\mathrm{E}} (\varrho\ \langle (\theta^{\mathrm{r}} \leftarrow \{\ \mathbf{S} \mapsto 1\ \}), \text{GO}\rangle.\ \mathbf{E}^{\mathrm{p}}[\text{nothing}])$

where

$\mathbf{S} \in \mathit{dom}(\theta^{\mathrm{r}})$

This is given by lemma 47 (EMIT IS SOUND).

□

LEMMA 35 (SYMMETRY OF CIRCUIT CONTEXTUAL EQUALITY)**.**

For all $\mathcal{C}_1, \mathcal{C}_2$*, if* $\mathcal{C}_1 \simeq^C \mathcal{C}_2$ *then* $\mathcal{C}_2 \simeq^C \mathcal{C}_1$

PROOF.

As $\simeq^C$ is defined, at its core, on equality of sequences of booleans, and that equality is symmetric, $\simeq^C$ must be as well. □

LEMMA 36 (TRANSITIVITY OF CIRCUIT CONTEXTUAL EQUALITY)**.**

For all $\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3$ *if* $\mathcal{C}_1 \simeq^C \mathcal{C}_2$ *and* $\mathcal{C}_2 \simeq^C \mathcal{C}_3$ *then* $\mathcal{C}_1 \simeq^C \mathcal{C}_3$

PROOF.

As $\simeq^C$ is defined, at its core, on equality of sequences of Booleans, and that equality is transitive, $\simeq^C$ must be as well. □

LEMMA 37 (REFLEXIVITY OF CIRCUIT CONTEXTUAL EQUALITY)**.**

For all $\mathcal{C}$, $\mathcal{C} \simeq^C \mathcal{C}$

PROOF.

This follows directly from the definition of $\simeq^C$, which relies on running the two circuits. As, in this case, the two circuits are the same then they will behave the same on all inputs. □

B.3. Reduction Relation Properties

This section contains lemmas and proofs that justify that the reduction relation $\rightharpoonup^{E}$ is sound with respect to the compilation function.

LEMMA 38 (PAR-SWAP IS SOUND)**.**

For all $\mathbf{p}^{P}$ *and* $\mathbf{q}^{P}$, $[\![(\text{par } \mathbf{p}^{P}\ \mathbf{q}^{P})]\!] \simeq^{C} [\![(\text{par } \mathbf{q}^{P}\ \mathbf{p}^{P})]\!]$

PROOF.

This can be seen trivally, as the graphs of $[\![(\text{par } \mathbf{p}^{P}\ \mathbf{q}^{P})]\!]$ and $[\![(\text{par } \mathbf{q}^{P}\ \mathbf{p}^{P})]\!]$ are symmetric. □

LEMMA 39 (PAR-NOTHING IS SOUND)**.**

For all $\mathbf{p}^{D}$, $[\![(\text{par nothing } \mathbf{p}^{D})]\!] \simeq^{C} [\![\mathbf{p}^{D}]\!]$

PROOF.

This proof is given in the notebook [par-done], which actually shows the more general

$[\![(\text{par nothing } \mathbf{p}^{P})]\!] \simeq^{C} [\![\mathbf{p}^{P}]\!]$. □

LEMMA 40 (TRAP IS SOUND)**.**

For all $\mathbf{p}^{S}$, $[\![(\text{trap } \mathbf{p}^{S})]\!] \simeq^{C} [\![\downarrow^{P} \mathbf{p}^{S}]\!]$

PROOF.

Cases of $\mathbf{p}^{S}$:

CASE 1:$\mathbf{p}^{S}$=nothing

Example:

```
> (assert-same
   (compile-esterel (term (trap nothing)))
   (compile-esterel (term (harp nothing))))
```

.

CASE 2:$\mathbf{p}^{S}$=(exit 0)

Example:

```
> (assert-same
   (compile-esterel (term (trap (exit 0))))
   (compile-esterel (term (harp (exit 0)))))
```

.

CASE 3:$\mathbf{p}^{S}$=(exit **n**)

Where $\mathbf{n} > 0$.

In this case, $\downarrow^{p}$ (exit **n**) = (exit **n**-1).

If we draw the circuit for $[\![(\text{exit } \mathbf{n})]\!]$ and $[\![(\text{exit } \mathbf{n}\text{-}1)]\!]$, we see that they give us the same graph.

□

LEMMA 41 (SUSPEND IS SOUND)**.**

For all $\mathbf{p}^{D}$ *and* $\mathbf{S}$, $[\![(\text{suspend } \mathbf{p}^{D}\ \mathbf{S})]\!] \simeq^{C} [\![\mathbf{p}^{D}]\!]$

PROOF.

This is proved in the [suspend] notebook. □

LEMMA 42 (SEQ-DONE IS SOUND)**.**

For all $\mathbf{q}^{\mathsf{p}}$, $[\![(\text{seq nothing } \mathbf{q}^{\mathsf{p}})]\!] \simeq^{C} [\![\mathbf{q}^{\mathsf{p}}]\!]$

PROOF.

$[\![(\text{seq nothing } \mathbf{q}^{\mathsf{p}})]\!]$ just connections the GO wire to $[\![\mathbf{q}^{\mathsf{p}}]\!]$(GO), which is exactly $[\![\mathbf{q}^{\mathsf{p}}]\!]$. Thus the two circuits are identical. □

LEMMA 43 (PAR2-EXIT IS SOUND)**.**

For all $\mathbf{n}_1$ *and* $\mathbf{n}_2$, $[\![(\text{par (exit } \mathbf{n}_1) \text{ (exit } \mathbf{n}_2))]\!] \simeq^{C} [\![(\text{exit } max(\mathbf{n}_1, \mathbf{n}_2))]\!]$

PROOF.

Cases of $\mathbf{n}_1 = \mathbf{n}_2$, $\mathbf{n}_1 > \mathbf{n}_2$, and $\mathbf{n}_1 < \mathbf{n}_2$:

CASE 1:$\mathbf{n}_1 = \mathbf{n}_2$

Induction on $\mathbf{n}_1$:

CASE 1.I:$\mathbf{n}_1$=0

See [par-2exit] notebook

CASE 1.II:$\mathbf{n}_1$=1+$\mathbf{m}$

Note that in this case the lem-n wire in the the synchronizer will be equal to lem, as all other exit codes will be 0, and therefore lem-n = lem $\vee$ 0 .... The same will hold for rem-n. We now can see that we have a synchronizer of the same shape as in the previous subcase. Thus the remainder of this proof proceeds in the same way.

CASE 2:$\mathbf{n}_1 > \mathbf{n}_2$

Induction on $\mathbf{n}_2$:

CASE 2.I:$\mathbf{n}_2$=0

Note that all ln up to l2+n_1 must be 0. Therefore all kn up to that point must be 0. The notebook [par-2exit] shows that the remainder of the synchronizer behaves as $[\![(\text{exit } \mathbf{n}_1)]\!]$.

CASE 2.II:$\mathbf{n}_2$=1+$\mathbf{m}$

All kn up to kn_2 must be zero as there are no corresponding ln or rn wires. From this point we can use analogous reasoning to the previous subcase.

CASE 3:$\mathbf{n}_1 < \mathbf{n}_2$

This case analogous to the previous case, as the synchronizer (and par) are symmetric.

□

LEMMA 44 (PAR1-EXIT IS SOUND)**.**

For all $\mathbf{n}$ *and* $\hat{\mathbf{p}}$, $[\![(\text{par } (\text{exit } \mathbf{n}) \hat{\mathbf{p}})]\!] \simeq^C [\![(\text{exit } \mathbf{n})]\!]$

PROOF.

The proof of this is given in the [par1-exit] notebook. □

LEMMA 45 (SEQ-EXIT IS SOUND)**.**

For all $\mathbf{n}$ *and* $\mathbf{q}^p$, *if* $[\![(\text{seq } (\text{exit } \mathbf{n}) \mathbf{q}^p)]\!](\text{SEL}) \simeq 0$ *then* $[\![(\text{seq } (\text{exit } \mathbf{n}) \mathbf{q}^p)]\!] \simeq^C [\![(\text{exit } \mathbf{n})]\!]$

PROOF.

By $[\![(\text{seq } (\text{exit } \mathbf{n}) \mathbf{q}^p)]\!](\text{SEL}) \simeq 0$, it must be that $[\![\mathbf{q}^p]\!](\text{SEL}) \simeq 0$. Thus by lemma 76 (ACTIVATION CONDITION) all output wires of $[\![\mathbf{q}^p]\!]$ are 0. Thus the only wire which can be true is K2+n, which in this case will be equal to $[\![(\text{exit } \mathbf{n})]\!](\text{K2+n})$. In addition by lemma 78 (CONSTRUCTIVE UNLESS ACTIVATED) $[\![\mathbf{q}^p]\!]$ never exhibits non-constructive behavior, thus this circuit is always constructive. Thus the two circuits are equal. □

LEMMA 46 (SIGNAL IS SOUND)**.**

For all $\mathbf{S}$ *and* $\mathbf{p}^{\mathrm{p}}$, $[\![(\mathrm{signal}\ \mathbf{S}\ \mathbf{p}^{\mathrm{p}})]\!] \simeq^{C} [\![(\varrho\ \langle\{\ \mathbf{S} \mapsto \bot\ \}, \mathrm{WAIT}\rangle.\ \mathbf{p}^{\mathrm{p}})]\!]$

PROOF.

$[\![(\mathrm{signal}\ \mathbf{S}\ \mathbf{p}^{\mathrm{p}})]\!]$ connects the input and output $\mathbf{S}$ wires to each other, and passes GO along unchanged. $[\![(\varrho\ \langle\{\ \mathbf{S} \mapsto \bot\ \}, \mathrm{WAIT}\rangle.\ \mathbf{p}^{\mathrm{p}})]\!]$ does the same, therefore the two circuits are identical. □

LEMMA 47 (EMIT IS SOUND)**.**

For all $\mathbf{r}^{\mathrm{p}} = (\varrho\ \langle\theta^{\mathrm{r}}, \mathrm{GO}\rangle.\ \mathbf{E}^{\mathrm{p}}[(\mathrm{emit}\ \mathbf{S})])$,
$[\![(\varrho\ \langle\theta^{\mathrm{r}}, \mathrm{GO}\rangle.\ \mathbf{E}^{\mathrm{p}}[(\mathrm{emit}\ \mathbf{S})])]\!] \simeq^{C} [\![(\varrho\ \langle(\theta^{\mathrm{r}} \leftarrow \{\ \mathbf{S} \mapsto 1\ \}), \mathrm{GO}\rangle.\ \mathbf{E}^{\mathrm{p}}[\mathrm{nothing}])]\!]$

PROOF.

Induction on $\mathbf{E}^{\mathrm{p}}$:

CASE 1:$\mathbf{E}^{\mathrm{p}}=\bigcirc$

This follows trivially, as an empty context connects GO directly so the signal, which is forced to be 1 by our environment.

CASE 2:$\mathbf{E}^{\mathrm{p}}=\mathbf{E1}^{\mathrm{p}}[\mathbf{E}^{\mathrm{p}}{}_{i}]$

Note that the right hand side of the reduction forces $[\![\theta^{\mathrm{r}}(\mathbf{S})]\!]$ to compile as $[\![1]\!]$ and it replaces $[\![(\mathrm{emit}\ \mathbf{S})]\!]$ a circuit that sets $[\![(\mathrm{emit}\ \mathbf{S})]\!](\mathbf{S}_o) = 0$. Nothing else is changed. By lemma 82 (S OUTPUT IRRELEVANT) any $\mathbf{S}_o$ is only read by its corresponding binder, which in this case is θ by lemma 79 (S IS MAINTAINED ACROSS E). Finally we know that the $[\![(\mathrm{emit}\ \mathbf{S})]\!](\mathbf{S}_o) \simeq [\![\mathbf{p}^{\mathrm{p}}]\!](\mathrm{GO})$ by lemma 80 (GO IS MAINTAINED ACROSS E). Therefore we change the value of no wires, so the circuits are the same.

□

Lemma 48 (is-present is sound)**.**

For all $\mathbf{r}^{\mathsf{p}} = (\varrho\ \langle \theta, \mathbf{A}\rangle.\ \mathbf{E}^{\mathsf{p}}[(\text{if } \mathbf{S}\ \mathbf{p}^{\mathsf{p}}\ \mathbf{q}^{\mathsf{p}})])$,

if $[\![(\varrho\ \langle \theta, \mathbf{A}\rangle.\ \mathbf{E}^{\mathsf{p}}[(\text{if } \mathbf{S}\ \mathbf{p}^{\mathsf{p}}\ \mathbf{q}^{\mathsf{p}})])]\!](\text{SEL}) \simeq 0$ *and* $\theta(\mathbf{S}) = 1$ *then,*

$[\![(\varrho\ \langle \theta, \mathbf{A}\rangle.\ \mathbf{E}^{\mathsf{p}}[(\text{if } \mathbf{S}\ \mathbf{p}^{\mathsf{p}}\ \mathbf{q}^{\mathsf{p}})])]\!] \simeq^{C} [\![(\varrho\ \langle \theta, \mathbf{A}\rangle.\ \mathbf{E}^{\mathsf{p}}[\mathbf{p}^{\mathsf{p}}])]\!]$

Proof.

As $[\![\theta]\!]$ will force the $\mathbf{S}^{\mathsf{i}}$ wire to be 1, by lemma 79 (S is maintained across E) we know that

$[\![(\text{if } \mathbf{S}\ \mathbf{p}^{\mathsf{p}}\ \mathbf{q}^{\mathsf{p}})]\!](\mathbf{S}^{\mathsf{i}}) \simeq 1$. Thus it suffices to show that $[\![(\text{if } \mathbf{S}\ \mathbf{p}^{\mathsf{p}}\ \mathbf{q}^{\mathsf{p}})]\!] \simeq^{C} [\![\mathbf{p}^{\mathsf{p}}]\!]$ under this condition. This proof is given in the [is-present] notebook. □

Lemma 49 (is-absent is sound)**.**

For all $\mathbf{r}^{\mathsf{p}} = (\varrho\ \langle \theta, \mathbf{A}\rangle.\ \mathbf{E}^{\mathsf{p}}[(\text{if } \mathbf{S}\ \mathbf{p}^{\mathsf{p}}\ \mathbf{q}^{\mathsf{p}})])$,

if $\theta(\mathbf{S}) = \bot$,

$\mathbf{S} \notin \mathit{Can}^{S}_{\varrho}(\!|(\varrho\ \langle \theta, \mathbf{A}\rangle.\ \mathbf{E}^{\mathsf{p}}[(\text{if } \mathbf{S}\ \mathbf{p}^{\mathsf{p}}\ \mathbf{q}^{\mathsf{p}})]), \{\}|\!)$ *and,*

$[\![(\varrho\ \langle \theta, \mathbf{A}\rangle.\ \mathbf{E}^{\mathsf{p}}[(\text{if } \mathbf{S}\ \mathbf{p}^{\mathsf{p}}\ \mathbf{q}^{\mathsf{p}})])]\!](\text{SEL}) \simeq 0$,

then

$[\![(\varrho\ \langle \theta, \mathbf{A}\rangle.\ \mathbf{E}^{\mathsf{p}}[(\text{if } \mathbf{S}\ \mathbf{p}^{\mathsf{p}}\ \mathbf{q}^{\mathsf{p}})])]\!] \simeq^{C} [\![(\varrho\ \langle \theta, \mathbf{A}\rangle.\ \mathbf{E}^{\mathsf{p}}[\mathbf{q}^{\mathsf{p}}])]\!]$

Proof.

Let $\mathbf{p}_{outer}$ be $(\varrho\ \langle \theta, \mathbf{A}\rangle.\ \mathbf{E}^{\mathsf{p}}[(\text{if } \mathbf{S}\ \mathbf{p}^{\mathsf{p}}\ \mathbf{q}^{\mathsf{p}})])$, the left hand side of the reduction. This can be proved by the following steps:

1. By lemma 79 (S is maintained across E) and lemma 80 (GO is maintained across E) we know that $[\![\mathbf{p}^{\mathsf{p}}{}_{outer}]\!](\mathbf{S}^{\mathsf{i}}) \simeq [\![(\text{if } \mathbf{S}\ \mathbf{p}^{\mathsf{p}}\ \mathbf{q}^{\mathsf{p}})]\!](\mathbf{S}$ and $[\![\mathbf{p}^{\mathsf{p}}{}_{outer}]\!](\text{GO}) \simeq [\![(\text{if } \mathbf{S}\ \mathbf{p}^{\mathsf{p}}\ \mathbf{q}^{\mathsf{p}})]\!](\text{GO})$

2. By lemma 71 (Can S is sound) and our premise that $[\![\mathbf{p}^{\mathsf{p}}{}_{outer}]\!](\text{SEL}) \simeq 0$, we know that $[\![\mathbf{p}^{\mathsf{p}}{}_{outer}]\!](\mathbf{S}^{\mathsf{o}}) \simeq 0$.

3. By the definition of $[\![\cdot]\!]$ on ϱ, we know that $[\![(\text{if } \mathbf{S}\ \mathbf{p}^{\mathrm{p}}\ \mathbf{q}^{\mathrm{p}})]\!](\mathbf{S}^{\mathrm{i}}) \simeq [\![(\text{if } \mathbf{S}\ \mathbf{p}^{\mathrm{p}}\ \mathbf{q}^{\mathrm{p}})]\!](\mathbf{S}^{\mathrm{o}})$

4. By (1), (2) & (3), we know that $[\![(\text{if } \mathbf{S}\ \mathbf{p}^{\mathrm{p}}\ \mathbf{q}^{\mathrm{p}})]\!](\mathbf{S}^{\mathrm{i}}) \simeq 0$.

5. By lemma 81 (Selection Definition), $[\![\mathbf{p}^{\mathrm{p}}{}_{outer}]\!](\mathrm{SEL}) \simeq [\![\mathbf{p}^{\mathrm{p}}]\!](\mathrm{SEL}) \vee [\![\mathbf{q}^{\mathrm{p}}]\!](\mathrm{SEL}) \vee \mathbf{w}_{others} \ldots$

6. By (5) and our premise that $[\![\mathbf{p}^{\mathrm{p}}{}_{outer}]\!](\mathrm{SEL}) \simeq 0$, we know that $[\![(\text{if } \mathbf{S}\ \mathbf{p}^{\mathrm{p}}\ \mathbf{q}^{\mathrm{p}})]\!](\mathrm{SEL}) \simeq 0$

7. Under (6) and lemma 76 (Activation Condition) we can show that $[\![(\text{if } \mathbf{S}\ \mathbf{p}^{\mathrm{p}}\ \mathbf{q}^{\mathrm{p}})]\!] \simeq^{C} [\![\mathbf{q}^{\mathrm{p}}]\!]$. This is done in the [is-absent] notebook.

□

Lemma 50 (merge is sound)**.**

For all $\mathbf{r}^{\mathrm{p}} = (\varrho\ \langle \theta^{\mathrm{r}}{}_1, \mathbf{A}_1 \rangle.\ \mathbf{E}^{\mathrm{p}}[(\varrho\ \langle \theta^{\mathrm{r}}{}_2, \mathbf{A}_2 \rangle.\ \mathbf{p}^{\mathrm{p}})])$ *if* $\mathbf{A}_1 \geq \mathbf{A}_2$ *and* $\vdash_{\mathrm{CB}}\ (\varrho\ \langle \theta^{\mathrm{r}}{}_1, \mathbf{A}_1 \rangle.\ \mathbf{E}^{\mathrm{p}}[(\varrho\ \langle \theta^{\mathrm{r}}{}_2, \mathbf{A}_2 \rangle.\ \mathbf{p}^{\mathrm{p}})])$ *then* $[\![(\varrho\ \langle \theta^{\mathrm{r}}{}_1, \mathbf{A}_1 \rangle.\ \mathbf{E}^{\mathrm{p}}[(\varrho\ \langle \theta^{\mathrm{r}}{}_2, \mathbf{A}_2 \rangle.\ \mathbf{p}^{\mathrm{p}})])]\!] \simeq^{C} [\![(\varrho\ \langle \theta^{\mathrm{r}}{}_1 \leftarrow \theta^{\mathrm{r}}{}_2, \mathbf{A}_1 \rangle.\ \mathbf{E}^{\mathrm{p}}[\mathbf{p}^{\mathrm{p}}])]\!]$

Proof.

This is a direct consequence of lemma 52 (Can Lift Environments) and lemma 51 (Merge Adjacent Environments). □

Lemma 51 (Merge Adjacent Environments)**.**

For all $\mathbf{p}^{\mathrm{p}}$, $\theta^{\mathrm{r}}{}_1$, $\theta^{\mathrm{r}}{}_2$, $\mathbf{A}_1$ *and* $\mathbf{A}_2$, *if* $\mathbf{A}_1 \geq \mathbf{A}_2$ *then* $[\![(\varrho\ \langle \theta^{\mathrm{r}}{}_1, \mathbf{A}_1 \rangle.\ (\varrho\ \langle \theta^{\mathrm{r}}{}_2, \mathbf{A}_2 \rangle.\ \mathbf{p}^{\mathrm{p}}))]\!] \simeq^{C} [\![(\varrho\ \langle \theta^{\mathrm{r}}{}_1 \leftarrow \theta^{\mathrm{r}}{}_2, \mathbf{A}_1 \rangle.\ \mathbf{p}^{\mathrm{p}})]\!]$

Proof.

Sketch: The compilation of ϱ only changes the outputs of its inner circuit in that it closes some of the signal wires, and that it only changes input values of some signals and the GO wire. Thus, we can argue that that equivalence base on three facts. First, that $\theta^{\mathrm{r}}{}_1 \leftarrow \theta^{\mathrm{r}}{}_2$ closes the same signals as the two nested environments. Second that these signals are closed in the same way: that is they input part of the signal will receive the same value. Third, that the value of the

GO wire does not change. In a sense, this means that the only effect of this rule is to move wires from one ”spot” in the circuit to another, without changing their connections.

1. By the definition of $\leftarrow$, $dom(\theta^r_1 \leftarrow \theta^r_2) = dom(\theta^r_1) \cup dom(\theta^r_2)$.

2. by the definition of $[\![\cdot]\!]$, compiling a ϱ closes only the wires in its θ^r's domain, we can see that the same wires are closed both expressions.

3. By (1) and (2), $inputs([\![(\varrho\ \langle\theta^r_1, \mathbf{A}_1\rangle.\ (\varrho\ \langle\theta^r_2, \mathbf{A}_2\rangle.\ \mathbf{p}^p))]\!]) = inputs([\![(\varrho\ \langle\theta^r_1 \leftarrow \theta^r_2, \mathbf{A}_1\rangle.\ \mathbf{p}^p)]\!])$ and $outputs([\![(\varrho\ \langle\theta^r_1, \mathbf{A}_1\rangle.\ (\varrho\ \langle\theta^r_2, \mathbf{A}_2\rangle.\ \mathbf{p}^p))]\!]) = outputs([\![(\varrho\ \langle\theta^r_1 \leftarrow \theta^r_2, \mathbf{A}_1\rangle.\ \mathbf{p}^p)]\!])$

4. The compilation of $(\varrho\ \langle\theta^r_2, \mathbf{A}_2\rangle.\ \bigcirc)$ will prevent the compilation of $(\varrho\ \langle\theta^r_1, \mathbf{A}_1\rangle.\ \bigcirc)$ from modifying any signals in the domain of θ_2, meaning those signals will get values as specified by the compilation of θ^r_2. In addition $\theta^r_1 \leftarrow \theta^r_2$ keep the value of any signal in θ^r_2, therefore those signals will compile the same way. Thus the value of no input signal is changed.

5. By (3) and (4), we know that for all $\mathbf{S} \in inputs([\![\mathbf{p}^p]\!])$, in $[\![(\varrho\ \langle\theta^r_1, \mathbf{A}_1\rangle.\ (\varrho\ \langle\theta^r_2, \mathbf{A}_2\rangle.\ \mathbf{p}^p))]\!]\ [\![\mathbf{p}^p]\!] \setminus \theta^r_1 \leftarrow \theta^r_2$, and in $[\![(\varrho\ \langle\theta^r_1 \leftarrow \theta^r_2, \mathbf{A}_1\rangle.\ \mathbf{p}^p)]\!], [\![\mathbf{p}^p]\!] \setminus \theta^r_1 \leftarrow \theta^r_2$.

6. As $\mathbf{A}_1 \geq \mathbf{A}_2$ we know that either both are GO, both are WAIT, or $\mathbf{A}_1 = \text{GO}$ and $\mathbf{A}_2 = \text{WAIT}$. In each case we can see that the actual value on $[\![\mathbf{p}^p]\!](\text{GO})$ remains the same. That is, in both cases, $[\![\mathbf{p}^p]\!] \setminus \mathbf{A}_1$.

7. The compilation of ϱ does not change the other control inputs and outputs of $[\![\mathbf{p}^p]\!]$.

8. By (5), (6), and (7), the inputs and outputs of $[\![\mathbf{p}^p]\!]$ are not changed, thus the behavior of the circuit is not changed.

□

LEMMA 52 (CAN LIFT ENVIRONMENTS)**.**

For all $\mathbf{p}, \mathbf{E}, \theta$, *and* $\mathbf{A}$, *if either* $\mathbf{A} = \text{WAIT}$ *or* $\mathbf{A} = \text{GO}$ *and* $[\![\mathbf{E}^p[(\varrho\ \langle\theta, \mathbf{A}\rangle.\ \mathbf{p}^p)]]\!](\text{GO}) = 1$, *and* $\vdash_{\text{CB}}\ \mathbf{E}^p[(\varrho\ \langle\theta, \mathbf{A}\rangle.\ \mathbf{p}^p)]$, *then*

$[\![\mathbf{E}^{\mathsf{p}}[(\varrho\ \langle\theta, \mathbf{A}\rangle.\ \mathbf{p}^{\mathsf{p}})]]\!] \simeq^{C} [\![(\varrho\ \langle\theta, \mathbf{A}\rangle.\ \mathbf{E}^{\mathsf{p}}[\mathbf{p}^{\mathsf{p}}])]\!]$

Proof.

This proof proceeds in two parts. First, by lemma 80 (GO is maintained across E), we know that lifting $\mathbf{A}$ across won't change the value of the GO wire of any subcircuit because either $\mathbf{A} = \text{WAIT}$, in which case its compilation does not change the GO wire at all, or $\mathbf{A} = \text{GO}$, in which case it will force the GO wire to be 1. But in this second case our hypothesis states that the GO wire was already 1, so nothing has changed.

Second, by $\vdash_{\text{CB}} \mathbf{E}^{\mathsf{p}}[(\varrho\ \langle\theta, \mathbf{A}\rangle.\ \mathbf{p}^{\mathsf{p}})]$ we know that the free variables of $\mathbf{E}$ and the bound variables of $(\varrho\ \langle\theta, \mathbf{A}\rangle.\ \mathbf{p}^{\mathsf{p}})$ are distinct. Thus lifting θ will not capture any new variables, therefore by lemma 83 (Free Variables are Input/Outputs), the compilation of θ will connect the exact same wires resulting in a circuit that is structurally the same after the lift. Thus lifting the signal environment also changes nothing. □

Lemma 53 (Correct binding is preserved)**.**

For all $\mathbf{p}$, $\mathbf{q}$, *if* $\mathbf{p} \rightharpoonup^{\text{E}} \mathbf{q}$ *and* $\vdash_{\text{CB}} \mathbf{p}$ *them* $\vdash_{\text{CB}} \mathbf{q}$

Proof.

Cases of $\mathbf{p} \rightharpoonup^{\text{E}} \mathbf{q}$:

Case 1:[**par-nothing**]

In this case we have $(\text{par nothing } \mathbf{p}^{\text{D}}) \rightharpoonup^{\text{E}} \mathbf{p}^{\text{D}}$

By the definition of $\vdash_{\text{CB}}$, our premise gives us that $\vdash_{\text{CB}} \mathbf{p}^{\text{D}}$.

Case 2:[**par-1exit**]

In this case we have $(\text{par } (\text{exit } \mathbf{n})\ \hat{\mathbf{p}}) \rightharpoonup^{\text{E}} (\text{exit } \mathbf{n})$

For any $\mathbf{n}$, $\vdash_{\text{CB}} (\text{exit } \mathbf{n})$ by the definition of $\vdash_{\text{CB}}$.

CASE 3:[**par-2exit**]

In this case we have $(\text{par } (\text{exit } \mathbf{n}_1)\ (\text{exit } \mathbf{n}_2)) \rightharpoonup^{E} (\text{exit } \mathit{max}(\mathbf{n}_1, \mathbf{n}_2))$

For any $\mathbf{n}$, $\vdash_{CB}\ (\text{exit } \mathbf{n})$ by the definition of $\vdash_{CB}$.

CASE 4:[**is-present**]

In this case we have $(\varrho\ \langle \theta^{r}, \mathbf{A} \rangle.\ \mathbf{E}[(\text{if } \mathbf{S}\ \mathbf{p}\ \mathbf{q})]) \rightharpoonup^{E} (\varrho\ \langle \theta^{r}, \mathbf{A} \rangle.\ \mathbf{E}[\mathbf{p}])$

where

$\theta^{r}(\mathbf{S}) = 1$

1. By the definition of BV and $\mathit{FV}\mathit{BV}(\text{if } \mathbf{S}\ \mathbf{p}\ \mathbf{q}) \subseteq \mathit{BV}(\mathbf{q})$ and $\mathit{FV}(\text{if } \mathbf{S}\ \mathbf{p}\ \mathbf{q}) \subseteq \mathit{FV}(\mathbf{q})$

2. By lemma 56 (SUBTERMS HAVE CORRECT BINDING), we know that $\vdash_{CB}\ \mathbf{q}$.

3. By (1), (2), and lemma 54 (CORRECT BINDING IN PRESERVE BY CONTEXT INSERTION) we know that $\vdash_{CB}\ \mathbf{E}[\mathbf{q}]$.

4. By (3), we can conclude that $\vdash_{CB}\ (\varrho\ \langle \theta^{r}, \mathbf{A} \rangle.\ \mathbf{E}[\mathbf{p}])$.

CASE 5:[**is-absent**]

In this case we have $(\varrho\ \langle \theta^{r}, \mathbf{A} \rangle.\ \mathbf{E}[(\text{if } \mathbf{S}\ \mathbf{p}\ \mathbf{q})]) \rightharpoonup^{E} (\varrho\ \langle \theta^{r}, \mathbf{A} \rangle.\ \mathbf{E}[\mathbf{q}])$

where

$\mathbf{S} \in \mathit{dom}(\theta^{r})$

$\theta^{r}(\mathbf{S}) = \bot$

$\mathbf{S} \notin \mathit{Can}^{S}_{\varrho}((\varrho\ \langle \theta^{r}, \mathbf{A} \rangle.\mathbf{E}[(\text{if } \mathbf{S}\mathbf{p}\mathbf{q})]), \{\})$

This case is analogous to the previous one

CASE 6:[**seq-done**]

In this case we have $(\text{seq nothing } \mathbf{q}) \rightharpoonup^{E} \mathbf{q}$

by the definition of $\vdash_{CB}\ (\text{seq nothing } \mathbf{q})$, we know that $\vdash_{CB}\ \mathbf{q}$

Case 7:[**seq-exit**]

In this case we have $(\text{seq}\ (\text{exit}\ \mathbf{n})\ \mathbf{q}) \rightharpoonup^{\text{E}} (\text{exit}\ \mathbf{n})$

For any $\mathbf{n}$, $\vdash_{\text{CB}}\ (\text{exit}\ \mathbf{n})$ by the definition of $\vdash_{\text{CB}}$.

Case 8:[**suspend**]

In this case we have $(\text{suspend}\ \mathbf{p}^{\text{S}}\ \mathbf{S}) \rightharpoonup^{\text{E}} \mathbf{p}^{\text{S}}$

By the definition of $\vdash_{\text{CB}}\ (\text{suspend}\ \mathbf{p}^{\text{S}}\ \mathbf{S})$, we know that $\vdash_{\text{CB}}\ \mathbf{p}^{\text{S}}$

Case 9:[**trap**]

In this case we have $(\text{trap}\ \mathbf{p}^{\text{S}}) \rightharpoonup^{\text{E}}\downarrow^{\text{p}} \mathbf{p}^{\text{S}}$

As $\mathbf{p}^{\text{S}}$ is either nothing or $(\text{exit}\ \mathbf{n})$, we know by the definition of $\vdash_{\text{CB}}$ that $\vdash_{\text{CB}}\ \mathbf{p}^{\text{S}}$.

Case 10:[**signal**]

In this case we have $(\text{signal}\ \mathbf{S}\ \mathbf{p}) \rightharpoonup^{\text{E}} (\varrho\ \langle \{\ \mathbf{S} \mapsto \bot\ \}, \text{WAIT} \rangle.\ \mathbf{p})$

As this proof does not change the bound or free variables, the term should remain $\vdash_{\text{CB}}$.

Case 11:[**merge**]

In this case we have $(\varrho\ \langle \theta^{\text{r}}{}_1, \mathbf{A}_1 \rangle.\ \mathbf{E}[(\varrho\ \langle \theta^{\text{r}}{}_2, \mathbf{A}_2 \rangle.\ \mathbf{p})]) \rightharpoonup^{\text{E}} (\varrho\ \langle (\theta^{\text{r}}{}_1 \leftarrow \theta^{\text{r}}{}_2), \mathbf{A}_1 \rangle.\ \mathbf{E}[\mathbf{p}])$

where

$\mathbf{A}_1 \geq \mathbf{A}_2$

This case is given by `R-maintain-lift-0` in the Agda codebase.

Case 12:[**emit**]

In this case we have $(\varrho\ \langle \theta^{\text{r}}, \text{GO} \rangle.\ \mathbf{E}[(\text{emit}\ \mathbf{S})]) \rightharpoonup^{\text{E}} (\varrho\ \langle (\theta^{\text{r}} \leftarrow \{\ \mathbf{S} \mapsto 1\ \}), \text{GO} \rangle.\ \mathbf{E}[\text{nothing}])$

where

$\mathbf{S} \in \mathit{dom}(\theta^{\text{r}})$

This follows by an analogous argument to the case for [**is-present**].

Case 13:[**loop**]

In this case we have $(\text{loop}\ \mathbf{p}) \rightharpoonup^{\text{E}} (\text{l}\overline{\text{oop}}\ \mathbf{p}\ \mathbf{p})$

The premise of $\vdash_{\text{CB}}$ (loop **p**) gives us that the bound and free variables of **p** are distinct. This give us the premise of $\vdash_{\text{CB}}$ ($\overline{\text{loop}}$ **p p**).

Case 14:[**loop^stop-exit**]

In this case we have ($\overline{\text{loop}}$ (exit **n**) **q**)$\rightharpoonup^{\text{E}}$(exit **n**)

For any **n**, $\vdash_{\text{CB}}$ (exit **n**) by the definition of $\vdash_{\text{CB}}$.

Case 15:[**par-swap**]

In this case we have (par **p q**)$\rightharpoonup^{\text{E}}$(par **q p**)

As set intersection is associative, this follows directly.

Case 16:[**shared**]

In this case we have (ϱ $\langle \theta^{\text{r}}, \mathbf{A} \rangle$. **E**[(shared **s** := **e p**)])$\rightharpoonup^{\text{E}}$($\varrho$ $\langle \theta^{\text{r}}, \mathbf{A} \rangle$. **E**[($\varrho$ $\langle${ **s** $\mapsto$ $\langle$**n** , old$\rangle$ }, WAIT$\rangle$. **p**)])

where

$FV(\mathbf{e}) \subseteq dom(\theta^{\text{r}})$

$\forall\, \mathbf{s} \in FV(\mathbf{e}).\ \mathbf{s} \notin Can^{\text{sh}}_{\varrho}((\varrho \langle \theta^{\text{r}}, \mathbf{A} \rangle \mathbf{E}[(\text{shared}\mathbf{s}:=\mathbf{e}\mathbf{p})]), \cdot)$

$\mathbf{n} = eval^{\text{H}}(\mathbf{e}, \theta^{\text{r}})$

This follows by an analogous argument to the case for [**is-present**].

Case 17:[**var**]

In this case we have (ϱ $\langle \theta^{\text{r}}, \mathbf{A} \rangle$. **E**[(var **x** := **e p**)])$\rightharpoonup^{\text{E}}$($\varrho$ $\langle \theta^{\text{r}}, \mathbf{A} \rangle$. **E**[($\varrho$ $\langle${ **x** $\mapsto$ $eval^{\text{H}}(\mathbf{e}, \theta^{\text{r}})$ }, WAIT$\rangle$. **p**)])

where

$FV(\mathbf{e}) \subseteq dom(\theta^{\text{r}})$

$\forall\, \mathbf{s} \in FV(\mathbf{e}).\ \mathbf{s} \notin Can^{\text{sh}}_{\varrho}((\varrho \langle \theta^{\text{r}}, \mathbf{A} \rangle \mathbf{E}[(\text{var}\mathbf{x}:=\mathbf{e}\mathbf{p})]), \cdot)$

This follows by an analogous argument to the case for [**is-present**].

Case 18:[**set-var**]

In this case we have (ϱ $\langle \theta^{\text{r}}, \mathbf{A} \rangle$. **E**[(:= **x e**)])$\rightharpoonup^{\text{E}}$($\varrho$ $\langle$($\theta^{\text{r}} \leftarrow$ { **x** $\mapsto$ $eval^{\text{H}}(\mathbf{e}, \theta^{\text{r}})$ }), **A**$\rangle$. **E**[nothing])

where

$\mathbf{x} \in dom(\theta^{\text{r}})$

$FV(\mathbf{e}) \subseteq dom(\theta^{r})$

$\forall\ \mathbf{s} \in FV(\mathbf{e}).\ \mathbf{s} \notin Can^{sh}_{\varrho}((\varrho\ \langle\theta^{r}, \mathbf{A}\rangle\ \mathbf{E}[(:=\mathbf{x}\mathbf{e})]), \cdot)$

This follows by an analogous argument to the case for [**is-present**].

Case 19:[**if-false**]

In this case we have $(\varrho\ \langle\theta^{r}, \mathbf{A}\rangle.\ \mathbf{E}[(\text{if!0 } \mathbf{x}\ \mathbf{p}\ \mathbf{q})]) \rightharpoonup^{E} (\varrho\ \langle\theta^{r}, \mathbf{A}\rangle.\ \mathbf{E}[\mathbf{q}])$

where

$\theta^{r}(\mathbf{x}) = 0$

This case follows by an analogous argument to the case for [**is-absent**].

Case 20:[**if-true**]

In this case we have $(\varrho\ \langle\theta^{r}, \mathbf{A}\rangle.\ \mathbf{E}[(\text{if!0 } \mathbf{x}\ \mathbf{p}\ \mathbf{q})]) \rightharpoonup^{E} (\varrho\ \langle\theta^{r}, \mathbf{A}\rangle.\ \mathbf{E}[\mathbf{p}])$

where

$\mathbf{x} \in dom(\theta^{r})$

$\theta^{r}(\mathbf{x}) \neq 0$

This case follows by an analogous argument to the case for [**is-present**].

Case 21:[**set-old**]

In this case we have $(\varrho\ \langle\theta^{r}, \text{GO}\rangle.\ \mathbf{E}[(+= \mathbf{s}\ \mathbf{e})]) \rightharpoonup^{E} (\varrho\ \langle(\theta^{r} \leftarrow \{\ \mathbf{s} \mapsto \langle eval/^{H}(\mathbf{e}, \theta^{r}), \text{new}\rangle\ \}), \text{GO}\rangle.\ \mathbf{E}[\text{nothing}])$

where

$\theta^{r}(\mathbf{s}) = \langle \_, \text{old}\rangle$

$FV(\mathbf{e}) \subseteq dom(\theta^{r})$

$\forall\ \mathbf{s} \in FV(\mathbf{e}).\ \mathbf{s} \notin Can^{sh}_{\varrho}((\varrho\ \langle\theta^{r}, \mathbf{E}[(+= \mathbf{s}\mathbf{e})]\rangle)), \cdot)$

This case follows by an analogous argument to the case for [**is-present**].

Case 22:[**set-new**]

In this case we have $(\varrho\ \langle\theta^{r}, \text{GO}\rangle.\ \mathbf{E}[(+= \mathbf{s}\ \mathbf{e})]) \rightharpoonup^{E} (\varrho\ \langle(\theta^{r} \leftarrow \{\ \mathbf{s} \mapsto \langle \mathbf{n} + eval/^{H}(\mathbf{e}, \theta^{r}), \text{new}\rangle\ \}), \text{GO}\rangle.\ \mathbf{E}[\text{nothing}])$

where

$FV(\mathbf{e}) \subseteq dom(\theta^{r})$

$\theta^{r}(\mathbf{s}) = \langle \_ , \text{new} \rangle$

$\forall\ \mathbf{s} \in \mathit{FV}(\mathbf{e}).\ \mathbf{s} \notin \mathit{Can}^{sh}_{\varrho}((\varrho\ \langle \theta^{r}, \mathbf{E}[(+= \mathbf{se})] \rangle))$, $\cdot$)

$\mathbf{n} = \mathit{eval}^{H}(\mathbf{e}, \theta^{r})$

This case follows by an analogous argument to the case for [**is-present**].

□

LEMMA 54 (CORRECT BINDING IN PRESERVE BY CONTEXT INSERTION)**.**

For all $\mathbf{E}$, $\mathbf{p}$, $\mathbf{q}$, *if* $\mathit{FV}(\mathbf{q}) \subseteq \mathit{FV}(\mathbf{p})$, $\mathit{BV}(\mathbf{q}) \subseteq \mathit{BV}(\mathbf{p})$, $\vdash_{CB}\ \mathbf{E}[\mathbf{p}]$, $\vdash_{CB}\ \mathbf{q}$ *then* $\vdash_{CB}\ \mathbf{E}[\mathbf{q}]$

PROOF.

Induction on $\mathbf{E}$:

CASE 1:$\mathbf{E}$=◯

This follows trivially by the premise that $\vdash_{CB}\ \mathbf{q}$.

CASE 2:$\mathbf{E}$=(seq $\mathbf{E}_o$ r)

1. By the definition of $\vdash_{CB}$ and the premise that $\vdash_{CB}\ \mathbf{E}[\mathbf{p}]$, we know that $\vdash_{CB}\ \mathbf{E}_o[\mathbf{p}]$.
2. By (1) we may invoke our induction hypothesis to conclude that $\vdash_{CB}\ \mathbf{E}_o[\mathbf{q}]$
3. By lemma 55 (FV AND IN-HOLE MAINTAIN SUBSET), we know that $\mathit{FV}(\mathbf{E}_o[\mathbf{p}]) \subseteq \mathit{FV}(\mathbf{E}_o[\mathbf{q}])$ and $\mathit{BV}(\mathbf{E}_o[\mathbf{p}]) \subseteq \mathit{BV}(\mathbf{E}_o[\mathbf{q}])$.
4. By (3) and the definition of $\bigcap$, we can conclude that $\mathit{BV}(\mathbf{E}_o[\mathbf{q}]) \cap \mathit{FV}(\mathrm{r}) = \varnothing$
5. By (2) and (4), we can conclude that $\vdash_{CB}\ \mathbf{E}[\mathbf{q}]$.

CASE 3:$\mathbf{E}$=(l$\overline{\text{oop}}$ $\mathbf{E}_o$ r)

As l$\overline{\text{oop}}$ has the same conditions as seq, this case is analogous to the previous one.

CASE 4:$\mathbf{E}$=(par $\mathbf{E}_o$ r)

1. By the definition of $\vdash_{\text{CB}}$ and the premise that $\vdash_{\text{CB}}\ \mathbf{E}[\mathbf{p}]$, we know that $\vdash_{\text{CB}}\ \mathbf{E}_o[\mathbf{p}]$.

2. By (1) we may invoke our induction hypothesis to conclude that $\vdash_{\text{CB}}\ \mathbf{E}_o[\mathbf{q}]$

3. By lemma 55 (FV AND IN-HOLE MAINTAIN SUBSET), we know that $\mathcal{FV}(\mathbf{E}_o[\mathbf{p}]) \subseteq \mathcal{FV}(\mathbf{E}_o[\mathbf{q}])$ and $\mathcal{BV}(\mathbf{E}_o[\mathbf{p}]) \subseteq \mathcal{BV}(\mathbf{E}_o[\mathbf{q}])$.

4. By (3) and the definition of $\bigcap$, we can conclude that $\mathcal{BV}(\mathbf{E}_o[\mathbf{q}]) \cap \mathcal{FV}(\mathrm{r}) = \varnothing$ and that $\mathcal{FV}(\mathbf{E}_o[\mathbf{q}]) \cap \mathcal{BV}(\mathrm{r}) = \varnothing$

5. By (2) and (4), we can conclude that $\vdash_{\text{CB}}\ \mathbf{E}[\mathbf{q}]$.

CASE 5:$\mathbf{E}$=(par r $\mathbf{E}_o$)

This case is analogous to the previous one.

CASE 6:$\mathbf{E}$=(trap $\mathbf{E}_o$)

This case follows by a straightforward induction.

CASE 7:$\mathbf{E}$=(suspend $\mathbf{E}_o$ $\mathbf{S}$)

This case follows by a straightforward induction and lemma 55 (FV AND IN-HOLE MAINTAIN SUBSET).

□

LEMMA 55 (FV AND IN-HOLE MAINTAIN SUBSET)**.**

For all $\mathbf{E}$, $\mathbf{p}$, $\mathbf{q}$, *if* $\mathcal{FV}(\mathbf{q}) \subseteq \mathcal{FV}(\mathbf{p})$ *and* $\mathcal{BV}(\mathbf{q}) \subseteq \mathcal{BV}(\mathbf{p})$, *then* $\mathcal{FV}(\mathbf{E}[\mathbf{p}]) \subseteq \mathcal{FV}(\mathbf{E}[\mathbf{q}])$ *and* $\mathcal{BV}(\mathbf{E}[\mathbf{p}]) \subseteq \mathcal{BV}(\mathbf{E}[\mathbf{q}])$

PROOF.

This follows by a straightforward induction over $\mathbf{E}$. □

LEMMA 56 (SUBTERMS HAVE CORRECT BINDING)**.**

For all $\mathbf{C}$, $\mathbf{q}$, *if* $\vdash_{\text{CB}}\ \mathbf{C}[\mathbf{q}]$, *then* $\vdash_{\text{CB}}\ \mathbf{q}$

Proof.

This follows by a straightforward induction over $\mathbf{C}$. □

B.4. Adequacy

This section contains the various lemma's needed for proving Adequacy of $\mathit{eval}^{\mathrm{E}}$.

Lemma 57 (Strongly Canonicalizing)**.**

For all $\mathbf{p}^{\mathrm{p}}_{\mathrm{GO}}$, $\mathbf{q}^{\mathrm{p}}_{\mathrm{GO}}$, *if* $\mathbf{p}^{\mathrm{p}}_{\mathrm{GO}} \longrightarrow^{\mathrm{R}} \mathbf{q}^{\mathrm{p}}_{\mathrm{GO}}$, *then* $\mathscr{S}(\mathbf{p}^{\mathrm{p}}_{\mathrm{GO}}) > \mathscr{S}(\mathbf{q}^{\mathrm{p}}_{\mathrm{GO}})$.

Interpretation. As $\mathscr{S}$ only returns natural numbers, by this we can conclude that eventually all terms will reach a state where there can only reduce by $\longrightarrow^{\mathrm{S}}$.

Proof.

Induction on $\longrightarrow^{\mathrm{R}}$:

Case 1: $\mathbf{p}^{\mathrm{p}}_{\mathrm{GO}} \rightharpoonup^{\mathrm{R}} \mathbf{p}^{\mathrm{p}}_{\mathrm{GO}}$

This case is given by lemma 58 (Strongly Canonicalizing on Compatible Closure).

Case 2: $\mathbf{C}^{\mathrm{p}}_{\mathrm{GO}}[\mathbf{p}^{\mathrm{p}}_{\mathrm{GO}\,i}] \longrightarrow^{\mathrm{R}} \mathbf{C}^{\mathrm{p}}_{\mathrm{GO}}[\mathbf{q}^{\mathrm{p}}_{\mathrm{GO}\,i}]$

In this case we have $\mathbf{p}^{\mathrm{p}}_{\mathrm{GO}\,i} \longrightarrow^{\mathrm{R}} \mathbf{q}^{\mathrm{p}}_{\mathrm{GO}\,i}$. By induction $\mathscr{S}(\mathbf{p}^{\mathrm{p}}_{\mathrm{GO}\,i}) > \mathscr{S}(\mathbf{q}^{\mathrm{p}}_{\mathrm{GO}\,i})$. Thus by lemma 58 (Strongly Canonicalizing on Compatible Closure) we can conclude that $\mathscr{S}(\mathbf{C}^{\mathrm{p}}_{\mathrm{GO}}[\mathbf{p}^{\mathrm{p}}_{\mathrm{GO}\,i}]) > \mathscr{S}(\mathbf{C}^{\mathrm{p}}_{\mathrm{GO}}[\mathbf{q}^{\mathrm{p}}_{\mathrm{GO}\,i}])$.

□

Lemma 58 (Strongly Canonicalizing on Compatible Closure)**.**

For all $\mathbf{C}^{\mathrm{p}}_{\mathrm{GO}}$, $\mathbf{p}^{\mathrm{p}}_{\mathrm{GO}}$, $\mathbf{q}^{\mathrm{p}}_{\mathrm{GO}}$, *if* $\mathscr{S}(\mathbf{p}^{\mathrm{p}}_{\mathrm{GO}}) > \mathscr{S}(\mathbf{q}^{\mathrm{p}}_{\mathrm{GO}})$ *then* $\mathscr{S}(\mathbf{C}^{\mathrm{p}}_{\mathrm{GO}}[\mathbf{p}^{\mathrm{p}}_{\mathrm{GO}}]) > \mathscr{S}(\mathbf{C}^{\mathrm{p}}_{\mathrm{GO}}[\mathbf{q}^{\mathrm{p}}_{\mathrm{GO}}])$

Proof.

This follows by a trivial induction over $\mathbf{C}^{\mathrm{p}}_{\mathrm{GO}}$, as each case of $\mathscr{S}\llbracket \mathbf{p}^{\mathrm{p}}_{\mathrm{GO}} \rrbracket$ only addes constants to the $\mathscr{S}$ of the subterms.

□

Lemma 59 (Strongly Canonicalizing on single step)**.**

For all $\mathbf{p}^{\mathrm{p}}_{\mathrm{GO}}$, $\mathbf{q}^{\mathrm{p}}_{\mathrm{GO}}$, *if* $\mathbf{p}^{\mathrm{p}}_{\mathrm{GO}} \rightharpoonup^{\mathrm{R}} \mathbf{q}^{\mathrm{p}}_{\mathrm{GO}}$ *then* $\mathscr{S}\llbracket \mathbf{p}^{\mathrm{p}}_{\mathrm{GO}} \rrbracket > \mathscr{S}\llbracket \mathbf{q}^{\mathrm{p}}_{\mathrm{GO}} \rrbracket$.

Proof.

Cases of $\mathbf{p}^{\mathrm{p}}_{\mathrm{GO}} \rightharpoonup^{\mathrm{R}} \mathbf{q}^{\mathrm{p}}_{\mathrm{GO}}$:

Case 1:[**par-nothing**]

In this case we have $(\mathrm{par}\ \mathrm{nothing}\ \mathbf{p}^{\mathrm{D}}) \rightharpoonup^{\mathrm{E}} \mathbf{p}^{\mathrm{D}}$

This case follows immediately from the definition of $\mathscr{S}$.

Case 2:[**par-1exit**]

In this case we have $(\mathrm{par}\ (\mathrm{exit}\ \mathbf{n})\ \hat{\mathbf{p}}) \rightharpoonup^{\mathrm{E}} (\mathrm{exit}\ \mathbf{n})$

This case follows immediately from the definition of $\mathscr{S}$.

Case 3:[**par-2exit**]

In this case we have $(\mathrm{par}\ (\mathrm{exit}\ \mathbf{n}_1)\ (\mathrm{exit}\ \mathbf{n}_2)) \rightharpoonup^{\mathrm{E}} (\mathrm{exit}\ \mathit{max}(\mathbf{n}_1, \mathbf{n}_2))$

This case follows immediately from the definition of $\mathscr{S}$.

Case 4:[**is-present**]

In this case we have $(\varrho\ \langle \theta^{\mathrm{r}}, \mathbf{A} \rangle.\ \mathbf{E}[(\mathrm{if}\ \mathbf{S}\ \mathbf{p}^{\mathrm{p}}_{\mathrm{GO}}\ \mathbf{q}^{\mathrm{p}}_{\mathrm{GO}})]) \rightharpoonup^{\mathrm{E}} (\varrho\ \langle \theta^{\mathrm{r}}, \mathbf{A} \rangle.\ \mathbf{E}[\mathbf{p}^{\mathrm{p}}_{\mathrm{GO}}])$

where

$\theta^{\mathrm{r}}(\mathbf{S}) = 1$

By lemma 58 (Strongly Canonicalizing on Compatible Closure), we can establish our result if $\mathscr{S}\llbracket (\mathrm{if}\ \mathbf{S}\ \mathbf{p}^{\mathrm{p}}_{\mathrm{GO}}\ \mathbf{q}^{\mathrm{p}}_{\mathrm{GO}}) \rrbracket > \mathbf{p}^{\mathrm{p}}_{\mathrm{GO}}$.

This is trivially true.

CASE 5:[**is-absent**]

In this case we have $(\varrho\ \langle \theta^{r}, \mathbf{A}\rangle.\ \mathbf{E}[(\text{if } \mathbf{S}\ \mathbf{p}^{p}_{GO}\ \mathbf{q}^{p}_{GO})])\rightharpoonup^{E}(\varrho\ \langle \theta^{r}, \mathbf{A}\rangle.\ \mathbf{E}[\mathbf{q}^{p}_{GO}])$

where

$\mathbf{S} \in \mathit{dom}(\theta^{r})$

$\theta^{r}(\mathbf{S}) = \bot$

$\mathbf{S} \notin \mathit{Can}^{S}_{\varrho}((\varrho\ \langle \theta^{r}, \mathbf{A}\rangle.\mathbf{E}[(\text{if } \mathbf{S}\mathbf{p}^{p}_{GO}\mathbf{q}^{p}_{GO})]), \{\})$

This case follows by an analgous argument to the previous case.

CASE 6:[**seq-done**]

In this case we have $(\text{seq nothing}\ \mathbf{q}^{p}_{GO})\rightharpoonup^{E}\mathbf{q}^{p}_{GO}$

This case follows immediately from the definition of $\mathscr{S}$.

CASE 7:[**seq-exit**]

In this case we have $(\text{seq (exit } \mathbf{n})\ \mathbf{q}^{p}_{GO})\rightharpoonup^{E}(\text{exit } \mathbf{n})$

This case follows immediately from the definition of $\mathscr{S}$.

CASE 8:[**suspend**]

In this case we have $(\text{suspend}\ \mathbf{p}^{S}\ \mathbf{S})\rightharpoonup^{E}\mathbf{p}^{S}$

This case follows immediately from the definition of $\mathscr{S}$.

CASE 9:[**trap**]

In this case we have $(\text{trap}\ \mathbf{p}^{S})\rightharpoonup^{E}\downarrow^{p}\mathbf{p}^{S}$

This case follows immediately from the definition of $\mathscr{S}$.

CASE 10:[**signal**]

In this case we have $(\text{signal}\ \mathbf{S}\ \mathbf{p}^{p}_{GO})\rightharpoonup^{E}(\varrho\ \langle\{\ \mathbf{S} \mapsto \bot\ \}, \text{WAIT}\rangle.\ \mathbf{p}^{p}_{GO})$

This case follows immediately from the definition of $\mathscr{S}$.

CASE 11:[**merge**]

In this case we have $(\varrho\ \langle \theta^{r}{}_{1}, \mathbf{A}_{1}\rangle.\ \mathbf{E}^{p}_{GO}[(\varrho\ \langle \theta^{r}{}_{2}, \mathbf{A}_{2}\rangle.\ \mathbf{p}^{p}_{GO})])\rightharpoonup^{E}(\varrho\ \langle(\theta^{r}{}_{1} \leftarrow \theta^{r}{}_{2}), \mathbf{A}_{1}\rangle.\ \mathbf{E}^{p}_{GO}[\mathbf{p}^{p}_{GO}])$

where

$\mathbf{A}_1 \geq \mathbf{A}_2$

1. $\mathscr{S}(\!|(\varrho\ \langle \theta^{r}{}_2, \mathbf{A}_2\rangle.\ \mathbf{p}^{p}_{GO})|\!) > \mathscr{S}(\!|\mathbf{p}^{p}_{GO}|\!)$, by the definition of $\mathscr{S}$.

2. for any r. $\mathscr{S}(\!|(\varrho\ \langle \theta^{r}{}_1, \mathbf{A}_1\rangle.\ \mathbf{E}^{p}_{GO}[r])|\!) = \mathscr{S}(\!|(\varrho\ \langle (\theta^{r}{}_1 \leftarrow \theta^{r}{}_2), \mathbf{A}_1\rangle.\ \mathbf{E}^{p}_{GO}[\mathbf{r}^{p}_{GO}])|\!)$, by the definition of $\mathscr{S}$.

3. By (2) on $\mathbf{p}$, $\mathscr{S}(\!|(\varrho\ \langle \theta^{r}{}_1, \mathbf{A}_1\rangle.\ \mathbf{E}[\mathbf{p}^{p}_{GO}])|\!) = \mathscr{S}(\!|(\varrho\ \langle (\theta^{r}{}_1 \leftarrow \theta^{r}{}_2), \mathbf{A}_1\rangle.\ \mathbf{E}[\mathbf{p}^{p}_{GO}])|\!)$.

4. By (3), (1), and lemma 58 (Strongly Canonicalizing on Compatible Closure),
   $\mathscr{S}(\!|(\varrho\ \langle \theta^{r}{}_1, \mathbf{A}_1\rangle.\ \mathbf{E}^{p}_{GO}[(\varrho\ \langle \theta^{r}{}_2, \mathbf{A}_2\rangle.\ \mathbf{p}^{p}_{GO})])|\!) > (\varrho\ \langle (\theta^{r}{}_1 \leftarrow \theta^{r}{}_2), \mathbf{A}_1\rangle.\ \mathbf{E}^{p}_{GO}[\mathbf{p}^{p}_{GO}])$

Case 12:[**emit**]

In this case we have $(\varrho\ \langle \theta^{r}, GO\rangle.\ \mathbf{E}^{p}_{GO}[(\text{emit } \mathbf{S})]) \rightharpoonup^{E} (\varrho\ \langle (\theta^{r} \leftarrow \{\ \mathbf{S} \mapsto 1\ \}), GO\rangle.\ \mathbf{E}^{p}_{GO}[\text{nothing}])$

where

$\mathbf{S} \in dom(\!|\theta^{r}|\!)$

1. For all r, $\mathscr{S}(\!|(\varrho\ \langle \theta^{r}, GO\rangle.\ \mathbf{E}[\mathbf{r}^{p}_{GO}])|\!) = \mathscr{S}(\!|(\varrho\ \langle (\theta^{r} \leftarrow \{\ \mathbf{S} \mapsto 1\ \}), GO\rangle.\ \mathbf{E}[\mathbf{r}^{p}_{GO}])|\!)$, By the definition of $\mathscr{S}$.

2. $\mathscr{S}(\!|(\text{emit } \mathbf{S})|\!) > \mathscr{S}(\!|\text{nothing}|\!)$, by the definition of $\mathscr{S}$.

3. $\mathscr{S}(\!|(\varrho\ \langle \theta^{r}, GO\rangle.\ \mathbf{E}[(\text{emit } \mathbf{S})])|\!) > \mathscr{S}(\!|(\varrho\ \langle (\theta^{r} \leftarrow \{\ \mathbf{S} \mapsto 1\ \}), GO\rangle.\ \mathbf{E}[\text{nothing}])|\!)$ by (1), (2), and lemma 58 (Strongly Canonicalizing on Compatible Closure).

□

B.4.1. Positive

Lemma 60 (Esterel Value is Circuit Value)**.**

Forall $(\varrho\ \langle \theta^{r}, GO\rangle.\ \mathbf{p}^{D})$, *if* $complete\text{-}wrt(\!|\theta^{r}, \mathbf{p}^{D}|\!)$, $(\varrho\ \langle \theta^{r}, GO\rangle.\ \mathbf{p}^{D})$ *is closed, and*

$[\![(\varrho\ \langle \theta^{r}, \mathrm{GO}\rangle.\ \mathbf{p}^{D})]\!](\mathrm{RES}) = [\![(\varrho\ \langle \theta^{r}, \mathrm{GO}\rangle.\ \mathbf{p}^{D})]\!](\mathrm{SUSP}) = [\![(\varrho\ \langle \theta^{r}, \mathrm{GO}\rangle.\ \mathbf{p}^{D})]\!](\mathrm{KILL}) = 0$, *and* $[\![(\varrho\ \langle \theta^{r}, \mathrm{GO}\rangle.\ \mathbf{p}^{D})]\!](\mathrm{GO}) = 1$.

then $[\![(\varrho\ \langle \theta^{r}, \mathrm{GO}\rangle.\ \mathbf{p}^{D})]\!]$ *is constructive.*

Proof.

To do this we must show that all wires in by $(\varrho\ \langle \theta^{r}, \mathrm{GO}\rangle.\ \mathbf{p}^{D})$ settle to a given value. First, we turn to the inputs. By the hypothesis of this lemma SUSP, RES, KILL, and GO have all settled.

For all signal wires in θ^{r}, by our hypothesis they are set to 1 by the defintion compilation of θ^{r}, or they are 0 by lemma 77 (Selection Start) and lemma 71 (Can S is sound).

For the remaining wires, they all settle by lemma 61 (Done is Constructive). □

Lemma 61 (Done is Constructive)**.**

For all $\mathbf{p}^{D}$ *and* θ, *if forall* $\mathbf{w} \in \mathit{inputs}([\![\mathbf{p}^{D}]\!])$, $[\![\mathbf{p}^{D}]\!](\mathbf{w}) \neq \bot$,

$[\![\mathbf{p}^{D}]\!](\mathrm{GO}) = 1$,

$[\![\mathbf{p}^{D}]\!](\mathrm{RES}) = [\![\mathbf{p}^{D}]\!](\mathrm{SUSP}) = [\![\mathbf{p}^{D}]\!](\mathrm{KILL}) = 0$,

and $[\![\mathbf{p}^{D}]\!] \setminus \theta$ *then* $[\![\mathbf{p}^{D}]\!]$ *is constructive*

Proof.

Induction on $\mathbf{p}^{D}$:

Case 1:$\mathbf{p}^{D}$=nothing

Example:

```
> (assert-totally-constructive (compile-esterel (term nothing)))
```

Case 2:$\mathbf{p}^{D}$=(exit $\mathbf{n}$)

Example:

```
> (assert-totally-constructive (compile-esterel (term (exit 5))))
```

Case 3:$\mathbf{p}^{D}$=pause

Example:

```
> (assert-totally-constructive (compile-esterel (term pause)))
```

Case 4:$\mathbf{p}^{D}$=(seq $\hat{\mathbf{p}}$ $\mathbf{q}^{p}$)

1. The compilation of seq passes all of its inputs to $\hat{\mathbf{p}}$ unchanged, therefor we can invoke our induction hypothesis to show that all wires in pause are defined.

2. by lemma 75 (Can K on paused is 1) we know that, for any possible binding environment θ, $0 \notin \mathit{Can}^{K}(\hat{\mathbf{p}}, \theta)$.

3. By (2) and lemma 77 (Selection Start) and lemma 72 (Can K is sound) we know that either $K0 \notin \mathit{inputs}(\llbracket \mathbf{p}^{D} \rrbracket)$ or $\llbracket \mathbf{p}^{D} \rrbracket(K0) = 0$.

4. By (3) and lemma 78 (Constructive unless Activated), $\llbracket \mathbf{q}^{p} \rrbracket$ is constructive.

5. By (1) and (4), the entire circuit is constructive.

Case 5:$\mathbf{p}^{D}$=(trap $\hat{\mathbf{p}}$)

The compilation of trap passes all of its inputs to $\hat{\mathbf{p}}$ unchanged, therefor we can invoke our induction hypothesis to show that all wires in pause are defined. Therefore the entire circuit is constructive.

CASE 6:$\mathbf{p}^{D}$=(par $\hat{\mathbf{p}}_1$ $\hat{\mathbf{p}}_2$)

The compilation of par passes all of its inputs to $\hat{\mathbf{p}}_1$ and $\hat{\mathbf{p}}_2$ unchanged, therefore by induction both are constructive. Note that the synchronizer is acyclic, therefore as all of its inputs are defined it too is constructive. Therefore the entire circuit is constructive.

CASE 7:$\mathbf{p}^{D}$=(suspend $\hat{\mathbf{p}}$ **S**)

1. By the definition of $[\![\cdot]\!]$, $[\![\hat{\mathbf{p}}]\!]$(GO) = $[\![$(suspend $\hat{\mathbf{p}}$ **S**)$]\!]$(GO) = 1.

2. By the definition of $[\![\cdot]\!]$, $[\![\hat{\mathbf{p}}]\!]$(KILL) = $[\![$(suspend $\hat{\mathbf{p}}$ **S**)$]\!]$(KILL) = 0.

3. let $\mathfrak{c}$ = $[\![$(suspend $\hat{\mathbf{p}}$ **S**)$]\!]$,

   By the definition of $[\![\cdot]\!]$, $[\![\hat{\mathbf{p}}]\!]$(SUSP) = $\mathfrak{c}$(SUSP) ∨ ($\mathfrak{c}$(**S**) ∧ $\mathfrak{c}$(RES) ∧ $\mathfrak{c}$(SEL)). which by our premises is:
   $[\![\hat{\mathbf{p}}]\!]$(SUSP) = 0 ∨ ($\mathfrak{c}$(**S**) ∧ 0 ∧ $\mathfrak{c}$(SEL)) = 0.

4. let $\mathfrak{c}$ = $[\![$(suspend $\hat{\mathbf{p}}$ **S**)$]\!]$,

   By the definition of $[\![\cdot]\!]$, $[\![\hat{\mathbf{p}}]\!]$(RES) = $\mathfrak{c}$(RES) ∧ $\mathfrak{c}$(SEL) ∧ ¬$\mathfrak{c}$(**S**) .
   which by our premises is: $[\![\hat{\mathbf{p}}]\!]$(RES) = 0 ∧ 1 ∧ ¬$\mathfrak{c}$(**S**) = 0.

5. By the definition of $[\![\cdot]\!]$, the input environment is passed in unchanged, therefore $[\![\hat{\mathbf{p}}]\!] \setminus \theta$.

6. By (1), (2), (3), (4), and (5), we an invoke our inductive hypothesis to conclude that $[\![\hat{\mathbf{p}}]\!]$ is constructive.

7. let $\mathfrak{c}$ = $[\![$(suspend $\hat{\mathbf{p}}$ **S**)$]\!]$.

   By the definition of $[\![\cdot]\!]$, the outputs of $\mathfrak{c}$ are the same as the outputs of $\hat{\mathbf{p}}$, except for the
   $\mathfrak{c}$(K1) = $[\![\hat{\mathbf{p}}]\!]$(K1) ∨ ($\mathfrak{c}$(**S**) ∧ $\mathfrak{c}$(RES) ∧ $[\![\hat{\mathbf{p}}]\!]$(SEL)).

   By our premises, this is:
   $\mathfrak{c}$(K1) = $[\![\hat{\mathbf{p}}]\!]$(K1) ∨ ($\mathfrak{c}$(**S**) ∧ 0 ∧ $[\![\hat{\mathbf{p}}]\!]$(SEL)) = $[\![\hat{\mathbf{p}}]\!]$(K1) ∨ 0 = $[\![\hat{\mathbf{p}}]\!]$(K1).

8. By (6) and (7) we can conclude that a wires are defined, and therefore $[\![(\text{suspend } \hat{\mathbf{p}}\ \mathbf{S})]\!]$ is constructive.

□

LEMMA 62 (BLOCKED TERMS ARE NON-CONSTRUCTIVE)**.**

For all $\mathbf{r}^{\mathrm{p}}{}_{outer} = (\varrho\ \langle \theta^{\mathrm{r}}, \mathrm{GO} \rangle.\ \mathbf{r}^{\mathrm{p}})$, *if* $\theta^{\mathrm{r}}; \mathrm{GO}; \bigcirc \vdash_{\mathrm{B}} \mathbf{r}^{\mathrm{p}}$, *and* $[\![(\varrho\ \langle \theta^{\mathrm{r}}, \mathrm{GO} \rangle.\ \mathbf{r}^{\mathrm{p}})]\!](\mathrm{SEL}) \simeq 0$ *then* $[\![\mathbf{r}^{\mathrm{p}}{}_{outer}]\!]$ *is non-constructive.*

PROOF.

1. By lemma 64 (BLOCKED IMPLIES CAN-RHO), we know that there is some signal $\mathbf{S}$ such that $\mathbf{S} \in \mathit{Can}^{\mathrm{S}}_{\varrho}(\!|\ (\varrho\ \langle \theta_1, \mathbf{A} \rangle.\ \mathbf{p}^{\mathrm{p}}), \{\}\ |\!)$.

2. By lemma 63 (INITIAL CONFIGURATIONS ARE NC), we know that any initial configuration is nc.

3. By (2) and lemma 65 (REACHABLE STATES FROM BLOCKED TERMS NON-CONSTRUCTIVE), we know that all reachable states will be *nc*.

4. By (1) and (3) we can conclude that there will be some signal wire $\mathbf{S}^{\mathrm{o}}$ such all reachable states will have that signal wire set to $\perp$.

5. By (4), the circuit is non-constructive.

□

LEMMA 63 (INITIAL CONFIGURATIONS ARE NC)**.**

For all $(\varrho\ \langle \theta^{\mathrm{r}}, \mathrm{GO} \rangle.\ \mathbf{p}^{\mathrm{p}})$, *if* $\mathit{closed}(\!|\ (\varrho\ \langle \theta^{\mathrm{r}}, \mathrm{GO} \rangle.\ \mathbf{p}^{\mathrm{p}})\ |\!)$ *then all possible initial configurations* $\theta^{\mathfrak{C}}{}_0$ *are* $\mathit{nc}(\!|\ (\varrho\ \langle \theta^{\mathrm{r}}, \mathrm{WAIT} \rangle.\ \mathbf{p}^{\mathrm{p}}), \{\}, \theta^{\mathfrak{C}}{}_0\ |\!)$.

PROOF.

As all internal and output wires start as $\perp$, and all return code and signal wires start are internal or output, and *nc* only makes demands about internal and output wires being $\perp$ this follows trivially.

The change form GO to WAIT is only necessary to enforce the grammar of $\mathbf{p}^{\mathrm{p}}$, as *nc* does not observe $\mathbf{A}$ ever this does not have any meaningful effect on our statements. □

LEMMA 64 (BLOCKED IMPLIES CAN-RHO)**.**

For all $\mathbf{p}$, θ^{r}, $\mathbf{A}$, *if* (θ^{r}); GO; $\bigcirc \vdash_{\mathrm{B}} \mathbf{p}^{\mathrm{p}}$ *then there exits some* $\mathbf{S}$ *such that* $\mathbf{S} \in \mathit{Can}_{\varrho}^{S}(\!|(\varrho\, \langle \theta^{\mathrm{r}}, \mathrm{GO}\rangle.\ \mathbf{p}^{\mathrm{p}}), \{\}|\!)$

PROOF.

This follows directly by induction over $\vdash_{\mathrm{B}}$, as the only bases cases involve either $\mathbf{S} \in \mathit{Can}_{\varrho}^{S}(\!|(\varrho\, \langle \theta^{\mathrm{r}}, \mathbf{A}\rangle.\ \mathbf{p}^{\mathrm{p}}), \{\}|\!)$ or $\mathbf{A} = \mathrm{WAIT}$, and the second case is excluded by our premises. □

LEMMA 65 (REACHABLE STATES FROM BLOCKED TERMS NON-CONSTRUCTIVE)**.**

For all $\mathbf{r}^{\mathrm{p}}{}_{outer} = (\varrho\, \langle \theta^{\mathrm{r}}, \mathrm{GO}\rangle.\ \mathbf{r}^{\mathrm{p}})$,

$\theta^{\varsigma}{}_{0}$, $\theta^{\varsigma}{}_{2}$,

let $\varsigma = [\![\mathbf{r}^{\mathrm{p}}]\!]$. *Let* $\theta^{\varsigma}{}_{0}$ *be an initial state.*

if $(\longrightarrow\!\!*\ \theta^{\varsigma}{}_{0}\ \theta^{\varsigma}{}_{1})$, $[\![(\varrho\, \langle \theta^{\mathrm{r}}, \mathrm{GO}\rangle.\ \mathbf{r}^{\mathrm{p}})]\!](\mathrm{SEL}) \simeq 0$, θ^{r}; GO; $\bigcirc \vdash_{\mathrm{B}} \mathbf{r}^{\mathrm{p}}$, *and* $\mathit{nc}(\mathbf{r}^{\mathrm{p}}, \theta, \theta^{\varsigma}{}_{0})$ *then* $\mathit{nc}(\mathbf{r}^{\mathrm{p}}, \theta, \theta^{\varsigma}{}_{2})$

PROOF.

1. By the definition of *nc*, there must exist some θ, such that $\mathit{nc}(\mathbf{r}^{\mathrm{p}}, \theta, \theta^{\varsigma}{}_{0})$.
2. As *nc* follows the same structure as Can_{ϱ}, we can use lemma 73 (CAN RHO S IS SOUND) to conclude that $\mathbf{r}^{\mathrm{p}} \setminus \theta$.
3. By (1) and (2) we may use lemma 66 (BLOCKED TERMS REMAIN NON-CONSTRUCTIVE). Thus by induction on the length of the reduction sequence $(\longrightarrow\!\!*\ \theta^{\varsigma}{}_{0}\ \theta^{\varsigma}{}_{1})$, we may conclude that $\mathit{nc}(\mathbf{r}^{\mathrm{p}}, \theta, \theta^{\varsigma}{}_{2})$.

□

LEMMA 66 (BLOCKED TERMS REMAIN NON-CONSTRUCTIVE)**.**

For all $\mathbf{r}^{\mathrm{p}}_{outer} = (\varrho\ \langle \theta^{\mathrm{r}}, \mathrm{GO}\rangle.\ \mathbf{E}^{\mathrm{p}}[\mathbf{r}^{\mathrm{p}}])$, θ, θ^{ς}_{1}, *and* θ^{ς}_{2},

let $\varsigma = [\![\mathbf{r}^{\mathrm{p}}]\!]$.

if $\theta^{\varsigma}_{1} \rightharpoonup^{\mathrm{C}} \theta^{\varsigma}_{2}$, $[\![\mathbf{r}^{\mathrm{p}}]\!](\mathrm{SEL}) \simeq 0$, $[\![\mathbf{r}^{\mathrm{p}}]\!] \setminus \theta, \theta^{\mathrm{r}}; \mathrm{GO}; \mathbf{E}^{\mathrm{p}} \vdash_{\mathrm{B}} \mathbf{r}^{\mathrm{p}}$, *and* $nc(\mathbf{r}^{\mathrm{p}}, \theta^{\mathrm{r}}, \theta^{\varsigma}_{1})$ *then* $nc(\mathbf{r}^{\mathrm{p}}, \theta^{\mathrm{r}}, \theta^{\varsigma}_{2})$

PROOF.

Induction on $\theta^{\mathrm{r}}; \mathrm{GO}; \mathbf{E}^{\mathrm{p}} \vdash_{\mathrm{B}} \mathbf{r}^{\mathrm{p}}$:

CASE 1:**[if]**

Let θ^{ς}_{1p} and θ^{ς}_{2p} be the substates that correspond to $\mathbf{p}^{\mathrm{p}}$ in θ^{ς}_{1} and θ^{ς}_{2} respectively. Let θ^{ς}_{1q} and θ^{ς}_{2q} be defined similarly.

1. By the definition of $\vdash_{\mathrm{B}}$, we know that $\mathbf{S} \in \mathit{Can}^{\mathrm{S}}_{\varrho}(\varrho\ \langle \theta^{\mathrm{r}}, \mathbf{A}\rangle.\ \mathbf{E}[(\mathrm{if}\ \mathbf{S}\ \mathbf{p}\ \mathbf{q})], \{\})$.

2. By the definition of $\vdash_{\mathrm{B}}$, we know that $\theta^{\mathrm{r}}(\mathbf{S}) = \bot$.

3. By $\mathrm{Can}\theta_{\mathrm{S}} \subseteq \mathrm{Can}_{\mathrm{S}}$ and (1), we know that $\mathbf{S} \in \mathit{Can}^{\mathrm{S}}(\mathbf{E}[(\mathrm{if}\ \mathbf{S}\ \mathbf{p}\ \mathbf{q})], \theta^{\mathrm{r}})$.

4. By (3), lemma 79 (S IS MAINTAINED ACROSS E) and our premise that $nc(\mathbf{r}^{\mathrm{p}}, \theta^{\mathrm{r}}, \theta^{\varsigma}_{1})$, we know that $\theta^{\varsigma}_{1}(\mathbf{S}^{\mathrm{i}}) = \theta^{\varsigma}_{1}(\mathbf{S}^{\mathrm{o}}) = \bot$.

5. By lemma 80 (GO IS MAINTAINED ACROSS E), we know that $\theta^{\varsigma}_{1i}(\mathrm{GO}) = 1$.

6. By lemma 79 (S IS MAINTAINED ACROSS E) and (4), we know that $\theta^{\varsigma}_{1i}(\mathbf{S}^{\mathrm{i}}) = \bot$.

7. By the definition of $[\![\cdot]\!]$, (5), and (6), we know that $\theta^{\varsigma}_{1p}(\mathrm{GO}) = \bot$ and $\theta^{\varsigma}_{1q}(\mathrm{GO}) = \bot$.

8. By the definition of *nc*, *nc-r*, (2) and (3) we know that $nc(\mathbf{p}^{\mathrm{p}}, \theta^{\mathrm{r}}, \theta^{\varsigma}_{1p})$ and $nc(\mathbf{p}^{\mathrm{p}}, \theta^{\mathrm{r}}, \theta^{\varsigma}_{1q})$.

9. By our premise that $[\![\mathbf{r}^{\mathrm{p}}]\!](\mathrm{SEL}) \simeq 0$, $[\![\mathbf{r}^{\mathrm{p}}]\!] \setminus \theta^{\mathrm{r}}$, (7), and (8), we may use lemma 67 (ADEQUACY OF CAN) to conclude $nc(\mathbf{p}^{\mathrm{p}}, \theta^{\mathrm{r}}, \theta^{\varsigma}_{2p})$ and $nc(\mathbf{p}^{\mathrm{p}}, \theta^{\mathrm{r}}, \theta^{\varsigma}_{2q})$.

10. By the definition of $[\![\cdot]\!]$, the output signals are either $\vee$ed from both branches. For any signal that is in *Can* of the other branch it must either be in *Can* of the other branch, and therefore by (9) be $\bot$, or it is not in *Can*, and therefore by lemma 71 (CAN S IS SOUND), and therefore must be 0. Therefore is a signal is in *Can* of the overall term, it is $\bot$ in θ^{c}_{2}. The same argument holds for the return codes.

11. By (9) and (10) we may conclude that $nc(\text{(if } \mathbf{S}\ \mathbf{p}^{p}\ \mathbf{q}^{p}), \theta^{r}, \theta^{c}_{2})$.

CASE 2:[**emit-wait**]

This clause is not possible, as we specified our **A** to be GO.

CASE 3:[**suspend**]

This case follows by a relatively straight forward induction.

CASE 4:[**trap**]

This case follows by a relatively straight forward induction.

CASE 5:[**seq**]

Cases of $0 \in \mathit{Can}^{K}(\mathbf{p}^{p}_{i}, \theta^{r})$:

CASE 5.1:$0 \in \mathit{Can}^{K}(\mathbf{p}^{p}_{i}, \theta^{r}) = 0 \notin \mathit{Can}^{K}(\mathbf{p}^{p}_{i}, \theta^{r})$

Let θ^{c}_{1p} and θ^{c}_{2p} be the substates that correspond to $\mathbf{p}^{p}$ in θ^{c}_{1} and θ^{c}_{2} respectively. Let θ^{c}_{1q} and θ^{c}_{2q} be defined similarly.

1. By lemma 72 (CAN K IS SOUND), we know that the K0 wire of $[\![\mathbf{p}^{p}_{i}]\!]$ is either currently 0 or is $\bot$ and will eventually step to 0.

2. By (1) and lemma 76 (ACTIVATION CONDITION) We know that all wires in the second subcircuit will be $\bot$ and eventually step to 0, or are currently 0.

3. By induction we know that $(\text{nc } \mathbf{p}^{p}_{i}\ \theta^{r}\ \theta^{c}_{p2})$.

4. By (2) and (3) we can conclude that $(\text{nc } \mathbf{p}^{p}_{i}\ \theta^{r}\ \theta^{c}_{2})$

CASE 5.II:$0 \in \mathit{Can}^{K}(\mathbf{p}^{p}_{i}, \theta^{r}) = 0 \in \mathit{Can}^{K}(\mathbf{p}^{p}_{i}, \theta^{r})$

This case follows similarly to the previous subcase.

CASE 6:[**par-both**]

This case follows by a relatively straight forward induction.

CASE 7:[**parl**]

This case follows by a relatively straight forward induction.

CASE 8:[**parr**]

This case follows by a relatively straight forward induction.

□

LEMMA 67 (ADEQUACY OF CAN)**.**

For all $\mathbf{r}^{p}$, θ, $\theta^{\mathfrak{c}}{}_{1}$, $\theta^{\mathfrak{c}}{}_{2}$ *let* $\mathfrak{c} = [\![\mathbf{r}^{p}]\!]$, *if* $\theta^{\mathfrak{c}}{}_{1} \longrightarrow^{C} \theta^{\mathfrak{c}}{}_{2}$, $[\![\mathbf{r}^{p}]\!](\mathrm{SEL}) \simeq 0$, $[\![\mathbf{r}^{p}]\!] \setminus \theta$, $\theta^{\mathfrak{c}}{}_{1}(\mathrm{GO}) = \bot$, *and* $\mathit{nc}(\mathbf{r}^{p}, \theta, \theta^{\mathfrak{c}}{}_{1})$ *then* $\mathit{nc}(\mathbf{r}^{p}, \theta, \theta^{\mathfrak{c}}{}_{2})$

INTERPRETATION. The core idea of this proof is that not only is *Can* sound, but is adequite to tell us when wires are $\bot$. Now this is subtle, as wires can be set to 1 without *Can* knowing, as it only analizes what *Can* happen, not what *Must* happen. Therefore this proof requires that GO is $\bot$, which is essense says that *nothing* must happen, as GO reprenents *must* in the circuit.

PROOF.

Induction on $\mathbf{r}^{p}$:

CASE 1:$\mathbf{r}^{p}$=nothing

1. $\mathit{Can}^{S}(\mathbf{r}^{p}, \theta)$ is empty, therefore $\mathit{nc}\text{-}S(\text{nothing}, \theta, \theta^{c}_{2})$ vacuously holds.

2. $\mathit{Can}^{K}(\mathbf{r}^{p}, \theta) = \{ 0 \}$, therefore we must only show that K0 is $\perp$. This holds trivially by the definition of $[\![\cdot]\!]$ and the fact that $\varsigma_{s1}(\text{GO}) = \perp$. Therefore $\mathit{nc}\text{-}S(\text{nothing}, \theta, \theta^{c}_{2})$ also holds.

3. $\mathit{nc}\text{-}r(\text{nothing}, \theta, \theta^{c}_{2})$ trivially holds.

4. By (1), (2), (3), $\mathit{nc}(\text{nothing}, \theta, \theta^{c}_{2})$ holds.

Case 2:$\mathbf{r}^{p}$=pause

This is analagous to the previous clause.

Case 3:$\mathbf{r}^{p}$=(exit $\mathbf{n}$)

This is analagous to the previous clause.

Case 4:$\mathbf{r}^{p}$=(emit $\mathbf{S}_{e}$)

This clause is analgous to the previous clauses, except the argument for Kn is repeated for $\mathbf{S}_{e}$.

Case 5:$\mathbf{r}^{p}$=(trap $\mathbf{p}^{p}$)

let θ^{c}_{1i} and θ^{c}_{2i} be the substates of θ^{c}_{1} and θ^{c}_{2} which corrispond to $[\![\mathbf{p}^{p}]\!]$.

1. By the defintion of nc and $\mathit{nc}\text{-}r$, we know that $\mathit{nc}(\mathbf{p}^{p}, \theta, \theta^{c}_{1i})$.

2. By the defintion of $[\![\cdot]\!]$ we know that the inner GO wire is unchanged, therefore it is $\perp$.

3. By the defintion of $[\![\cdot]\!]$ the signals are passed in unchanged, therefore $[\![\mathbf{p}^{p}]\!] \setminus \theta$

4. By the defintion of $[\![\cdot]\!]$ SEL is unchanged, therefore $[\![\mathbf{p}^{p}]\!](\text{SEL}) \simeq 0$.

5. If the step from θ^{c}_{1} to θ^{c}_{2} changes the substate By (1) through (4) we can invoke our induction hypothesis to learn that $\mathit{nc}(\mathbf{r}^{p}, \theta, \theta^{c}_{2i})$ for the inner circuit. Otherwise the substate remains unchanged therefore we still have $\mathit{nc}(\mathbf{r}^{p}, \theta, \theta^{c}_{2i})$

6. By the definition of compile the output signals remain unchanged. Therefore (5) gives us that $\mathit{nc}\text{-}S(\mathbf{r}^{p}, \theta, \theta^{c}_{2})$

7. Now we inspect the return codes and their wires. To show $\mathit{nc}\text{-}\mathcal{S}(\mathbf{r}^{\mathsf{p}}, \theta, \theta^{\mathsf{c}}_{2i})$ from $\mathit{nc}\text{-}\mathcal{S}(\mathbf{p}^{\mathsf{p}}, \theta, \theta^{\mathsf{c}}_{2})$ (by (5)), we must case on each $\mathbf{n} \in \mathit{Can}^{\mathrm{K}}(\mathbf{p}^{\mathsf{p}}, \theta)$: Cases of $\mathbf{n}$:

Case 5.i: $\mathbf{n} = 0$

In this case we know that $\theta^{\mathsf{c}}_{2i}(\mathrm{K0}) = \bot$, and $0 \in \mathit{Can}^{\mathrm{K}}(\mathbf{r}^{\mathsf{p}}, \theta)$. We have two subcases, either, $2 \in \mathit{Can}^{\mathrm{K}}(\mathbf{p}^{\mathsf{p}}, \theta)$, in which case by (5) we know that $\theta^{\mathsf{c}}_{2i}(\mathrm{K2}) = \bot$, or $2 \notin \mathit{Can}^{\mathrm{K}}(\mathbf{p}^{\mathsf{p}}, \theta)$, in which case $[\![\mathbf{p}^{\mathsf{p}}]\!](\mathrm{K2}) \simeq 0$. In either case the only possible value for $\theta^{\mathsf{c}}_{2}(\mathrm{K0}) = \bot$ is $\bot$.

Case 5.ii: $\mathbf{n} = 1$

In this case K1 is pass out unchanged, thus it must still be $\bot$.

Case 5.iii: $\mathbf{n} = 2$

This follow by the same arugment as the case for 0.

Case 5.iv: $\mathbf{n} > 3$

In this case In this case Kn is pass out unchanged, but renamed to Kn-1. This matches exactly with the behavior of Can, therefore the wire will remain $\bot$

8. by (5), (6), and (7), we may conclude that $\mathit{nc}(\mathbf{r}^{\mathsf{p}}, \theta, \theta^{\mathsf{c}}_{2})$

Case 6: $\mathbf{r}^{\mathsf{p}}$=(suspend $\mathbf{p}^{\mathsf{p}}$ $\mathbf{S}_s$)

This case follows by a similar argument to trap, but relies on $[\![\mathbf{r}^{\mathsf{p}}]\!](\mathrm{SEL}) \simeq 0$ instead of reasoning about the $\vee$ of K0 and K2.

Case 7: $\mathbf{r}^{\mathsf{p}}$=(if $\mathbf{S}_f$ $\mathbf{p}^{\mathsf{p}}$ $\mathbf{q}^{\mathsf{p}}$)

Let θ^{c}_{1p} and θ^{c}_{2p} be the substates of θ^{c}_{1} and θ^{c}_{2} that corrispond to $[\![\mathbf{p}^{\mathsf{p}}]\!]$. Let θ^{c}_{1q} and θ^{c}_{2q} be defined similarly. Cases of $\theta(\mathbf{S}_f)$:

Case 7.i: $\theta(\mathbf{S}_f)$=1

1. By the definition of *nc* and *nc-r*, we know that $nc(\mathbf{p}^{\mathrm{p}}, \theta, \theta^{\mathfrak{c}}_{1p})$.

2. By $[\![\mathbf{r}^{\mathrm{p}}]\!] \setminus \theta$ we know that $[\![\mathbf{q}]\!](\mathrm{GO}) \simeq 0$.

3. By $[\![\mathbf{r}^{\mathrm{p}}]\!](\mathrm{SEL}) \simeq 0$ and the definition of $[\![\cdot]\!]$ we know that $[\![\mathbf{p}^{\mathrm{p}}]\!](\mathrm{SEL}) \simeq 0$ and $[\![\mathbf{q}^{\mathrm{p}}]\!](\mathrm{SEL}) \simeq 0$.

4. By (2), (3), and lemma 76 (Activation Condition) we know that all outputs of $[\![\mathbf{q}^{\mathrm{p}}]\!]$ are 0.

5. By the definition of $[\![\cdot]\!]$ we know that $\theta^{\mathfrak{c}}_{1p}(\mathrm{GO}) = \bot$.

6. As the signals are passed in unchanged we may conlcude that $[\![\mathbf{p}^{\mathrm{p}}]\!] \setminus \theta$.

7. By (1), (5) and (3), (6) and our induction hypothesis we know that $nc(\mathbf{p}^{\mathrm{p}}, \theta, \theta^{\mathfrak{c}}_{2p})$

8. By (4) we know that the outputs of the circuit are given by the output of $[\![\mathbf{p}^{\mathrm{p}}]\!]$.

9. By (8) and (7) we know that $nc(\mathbf{p}^{\mathrm{p}}, \theta, \theta^{\mathfrak{c}}_{2})$

Case 7.ii: $\theta(\mathbf{S}_f)=0$

This case follows by an analagous arugment to the previous one.

Case 7.iii: $\theta(\mathbf{S}_f)=\bot$

1. If $\mathbf{S}_f \in Can^{\mathrm{K}}(\mathbf{r}^{\mathrm{p}}, \theta)$, we may repeat the argument from the 0 case, by lemma 71 (Can S is sound). Therefore we must show this for $\mathbf{S}_f \notin Can^{\mathrm{K}}(\mathbf{r}^{\mathrm{p}}, \theta)$.

2. By the definition of *nc* and *nc-r*, we know that $nc(\mathbf{p}^{\mathrm{p}}, \theta, \theta^{\mathfrak{c}}_{1p})$ and $nc(\mathbf{q}^{\mathrm{p}}, \theta, \theta^{\mathfrak{c}}_{1q})$.

3. By (1), and $nc(\mathbf{p}^{\mathrm{p}}, \theta, \theta^{\mathfrak{c}}_{1})$, we know that $\theta^{\mathfrak{c}}_{1}(\mathbf{S}^{\mathrm{o}}_f) = \bot$.

4. By (3) and $\theta^{\mathfrak{c}}_{1}(\mathrm{GO}) = \bot$, and the definition of $[\![\cdot]\!]$ we can conclude that $\theta^{\mathfrak{c}}_{1p}(\mathrm{GO}) = \bot$ and $\theta^{\mathfrak{c}}_{1q}(\mathrm{GO}) = \bot$

5. By $[\![\mathbf{r}^{\mathrm{p}}]\!](\mathrm{SEL}) \simeq 0$ and the definition of $[\![\cdot]\!]$ we know that $[\![\mathbf{p}^{\mathrm{p}}]\!](\mathrm{SEL}) \simeq 0$ and $[\![\mathbf{q}^{\mathrm{p}}]\!](\mathrm{SEL}) \simeq 0$.

6. As the signals are passed in unchanged we may conlcude that $[\![\mathbf{p}^{\mathrm{p}}]\!] \setminus \theta$ and $[\![\mathbf{q}^{\mathrm{p}}]\!] \setminus \theta$.

7. By (5), (6), (4), and (2) we may conclude that $nc(\mathbf{p}^{\mathrm{p}}, \theta, \theta^{\mathrm{c}}_{2p})$ and $nc(\mathbf{q}^{\mathrm{p}}, \theta, \theta^{\mathrm{c}}_{2q})$

8. By the definition of $[\![\cdot]\!]$, the output signals are either $\vee$ed from both branches. For any signal that is in *Can* of the other branch it must either be in *Can* of the other branch, and therefore by (7) be $\bot$, or it is not in *Can*, and therefore by lemma 71 (CAN S IS SOUND), and therefore must be 0. Therefore is a signal is in *Can* of the overall term, it is $\bot$ in θ^{c}_2. The same argument holds for the return codes.

9. By (7) and (8) we may conclude that $nc(\mathbf{r}^{\mathrm{p}}, \theta, \theta^{\mathrm{c}}_2)$.

CASE 8:$\mathbf{r}^{\mathrm{p}}$=(seq $\mathbf{p}^{\mathrm{p}}$ $\mathbf{q}^{\mathrm{p}}$)

Let θ^{c}_{1p} and θ^{c}_{2p} be the substates of θ^{c}_1 and θ^{c}_2 that corrispond to $[\![\mathbf{p}^{\mathrm{p}}]\!]$. Let θ^{c}_{1q} and θ^{c}_{2q} be defined similarly.

1. By the definitions of *nc* and *nc-r*, we have two cases but both let us conclude that $nc(\mathbf{p}^{\mathrm{p}}, \theta, \theta^{\mathrm{c}}_{1p})$.

2. As GO is sent directly to $[\![\mathbf{p}^{\mathrm{p}}]\!]$, we may conclude that $\theta^{\mathrm{c}}_{1i}(\mathrm{GO}) = \bot$.

3. By the definition of $[\![\cdot]\!]$, all of the signals are broadcast to the both subcircuits, thus we may conclude that $[\![\mathbf{p}^{\mathrm{p}}]\!] \setminus \theta$ and $[\![\mathbf{q}^{\mathrm{p}}]\!] \setminus \theta$.

4. By the definition of $[\![\cdot]\!]$ and the premise that

   $[\![\mathbf{r}^{\mathrm{p}}]\!](\mathrm{SEL}) \simeq 0$ we may conclude that $[\![\mathbf{p}^{\mathrm{p}}]\!](\mathrm{SEL}) \simeq 0$ and $[\![\mathbf{q}^{\mathrm{p}}]\!](\mathrm{SEL}) \simeq 0$.

5. By (4), (1), (2), and (3) we may conclude that $nc(\mathbf{p}^{\mathrm{p}}, \theta, \theta^{\mathrm{c}}_{2p})$.

6. Following the structure of *Can* and *nc-r*, we have: Cases of $0 \in Can^{\mathrm{K}}(\mathbf{p}^{\mathrm{p}}, \theta)$:

CASE 8.1:$0 \notin Can^{\mathrm{K}}(\mathbf{p}^{\mathrm{p}}, \theta)$

   6.1. By lemma 72 (CAN K IS SOUND), we can conclude that $[\![\mathbf{p}^{\mathrm{p}}]\!](\mathrm{K0}) \simeq 0$.

6.2. By (6.1) and the definition of $[\![\cdot]\!]$, we can conclude that $[\![\mathbf{q}^{\mathrm{p}}]\!](\mathrm{GO}) \simeq 0$.

6.3. By (6.2), the premise that $[\![\mathbf{r}^{\mathrm{p}}]\!](\mathrm{SEL}) \simeq 0$, and lemma 76 (Activation Condition), we may conclude that all the outputs of $[\![\mathbf{q}^{\mathrm{p}}]\!]$ are 0.

6.4. By the definition of $[\![\cdot]\!]$ and (6.3), all signals and control wires will have their value defined by $[\![\mathbf{p}^{\mathrm{p}}]\!]$. Thus we may conclude that $\mathit{nc}\text{-}\mathcal{S}(\mathbf{r}^{\mathrm{p}}, \theta, \theta^{\mathfrak{C}}{}_2)$ and $\mathit{nc}\text{-}\kappa(\mathbf{r}^{\mathrm{p}}, \theta, \theta^{\mathfrak{C}}{}_2)$.

6.5. By (6.4) and (5) we may conclude that $\mathit{nc}(\mathbf{r}^{\mathrm{p}}, \theta, \theta^{\mathfrak{C}}{}_2)$.

Case 8.ii:$0 \in \mathit{Can}^{\mathrm{K}}(\mathbf{p}^{\mathrm{p}}, \theta)$

6.1. By the definitions of *nc* and *nc-r* we may conclude that $\mathit{nc}(\mathbf{q}^{\mathrm{p}}, \theta, \theta^{\mathfrak{C}}{}_{2p})$.

6.2. By (1) and the premise of this case we may conclude that $\theta^{\mathfrak{C}}{}_{1p}(\mathrm{K0}) = \bot$.

6.3. By (6.2) and the definition of $[\![\cdot]\!]$ we may conclude that $\theta^{\mathfrak{C}}{}_{1q}(\mathrm{GO}) = \bot$.

6.4. By (4), (3), (6.3), and (6.1) we may conclude that $\mathit{nc}(\mathbf{p}^{\mathrm{p}}, \theta, \theta^{\mathfrak{C}}{}_{2q})$.

6.5. By the same argument as in the 1 case, using (5) and (6.4), we may conclude that $\mathit{nc}\text{-}\mathcal{S}(\mathbf{r}^{\mathrm{p}}, \theta, \theta^{\mathfrak{C}}{}_2)$ and $\mathit{nc}\text{-}\kappa(\mathbf{r}^{\mathrm{p}}, \theta, \theta^{\mathfrak{C}}{}_2)$.

6.6. By (5), (6.4), and (6.5) we may conclude that $\mathit{nc}(\mathbf{r}^{\mathrm{p}}, \theta, \theta^{\mathfrak{C}}{}_2)$.

Case 9:$\mathbf{r}^{\mathrm{p}}$=(par $\mathbf{p}^{\mathrm{p}}$ $\mathbf{q}^{\mathrm{p}}$)

This case proceeds by a similar argument to the previous two cases, except that GO is broadcast to both subcircuits, therefore no argument is needed to show that the GO of the subcircuits are $\bot$.

Case 10:$\mathbf{r}^{\mathrm{p}}$=(signal $\mathbf{S}_s$ $\mathbf{p}^{\mathrm{p}}$)

Let $\theta^{\mathfrak{C}}{}_{1i}$ and $\theta^{\mathfrak{C}}{}_{2i}$ be the substates of $\theta^{\mathfrak{C}}{}_1$ and $\theta^{\mathfrak{C}}{}_2$ which corrispond to $[\![\mathbf{p}^{\mathrm{p}}]\!]$.

1. By the definition of $[\![\cdot]\!]$, the SEL wire is unchanged, thus we may conclude that $[\![\mathbf{p}^{\mathsf{p}}]\!](\text{SEL}) \simeq 0$.

2. By the definition of $[\![\cdot]\!]$, the GO wire is unchanged, thus we may conclude that $\theta^{\mathsf{c}}{}_{1i}(\text{GO}) = \bot$.

3. By the definitions of *Can*, and *nc-r*, we have the following cases: Cases of $\mathbf{S}_s \in \mathit{Can}^{S}(\mathbf{p}^{\mathsf{p}}, \theta \leftarrow \{ \mathbf{S}_s \mapsto \bot \})$:

CASE 10.I: $\mathbf{S}_s \in \mathit{Can}^{S}(\mathbf{p}^{\mathsf{p}}, \theta \leftarrow \{ \mathbf{S}_s \mapsto \bot \})$

   3.1. By the definitions of *nc* and *nc-r*, we know that $\mathit{nc}(\mathbf{p}^{\mathsf{p}}, \theta \leftarrow \{ \mathbf{S} \mapsto \bot \}, \theta^{\mathsf{c}}{}_{1i})$.

   3.2. As $\theta \leftarrow \{ \mathbf{S}_s \mapsto \bot \}$ puts no restrictions on the values for $\mathbf{S}_s$, and the remainder of signal wires are passed in unchanged, we may conclude that $\mathbf{p}^{\mathsf{p}} \setminus \theta \leftarrow \{ \mathbf{S}_s \mapsto \bot \}$.

   3.3. By (3.1), (3.2), (1), and (2), we may conclude that $\mathit{nc}(\mathbf{p}^{\mathsf{p}}, \theta \leftarrow \{ \mathbf{S} \mapsto \bot \}, \theta^{\mathsf{c}}{}_{2i})$.

   3.4. As all signal wires that are not $\mathbf{S}_s$ and all control wires are passed out unchanged, and as $\mathbf{S}_s$ is removed from the output of *Can*, we may use (3.3) to conclude that $\mathit{nc}\text{-}\mathcal{S}(\mathbf{r}^{\mathsf{p}}, \theta, \theta^{\mathsf{c}}{}_{2})$ and $\mathit{nc}\text{-}\kappa(\mathbf{r}^{\mathsf{p}}, \theta, \theta^{\mathsf{c}}{}_{2})$.

   3.5. By (3.3) and (3.4) we may conclude that $\mathit{nc}(\mathbf{r}^{\mathsf{p}}, \theta, \theta^{\mathsf{c}}{}_{2})$.

CASE 10.II: $\mathbf{S}_s \notin \mathit{Can}^{S}(\mathbf{p}^{\mathsf{p}}, \theta \leftarrow \{ \mathbf{S} \mapsto \bot \})$

   3.1. By the definitions of *nc* and *nc-r*, we know that $\mathit{nc}(\mathbf{p}^{\mathsf{p}}, \theta \leftarrow \{ \mathbf{S} \mapsto 0 \}, \theta^{\mathsf{c}}{}_{1i})$.

   3.2. By lemma 71 (CAN S IS SOUND), we may conclude that $[\![\mathbf{p}^{\mathsf{p}}]\!](\mathbf{S}^{\mathsf{o}}{}_{s}) \simeq 0$.

   3.3. By (3.2) and the defintion of complie (which links $\mathbf{S}^{\mathsf{o}}$ to $\mathbf{S}^{\mathsf{i}}{}_{s}$), we may conclude that $[\![\mathbf{p}^{\mathsf{p}}]\!](\mathbf{S}^{\mathsf{i}}{}_{s}) \simeq 0$.

   3.4. As the remainder of the signal wires are unchanged, we may conclude that $[\![\mathbf{p}^{\mathsf{p}}]\!] \setminus \theta \leftarrow \{ \mathbf{S} \mapsto 0 \}$

   3.5. By (3.1), (3.4), (1), and (2), we may conclude that $\mathit{nc}(\mathbf{p}^{\mathsf{p}}, \theta \leftarrow \{ \mathbf{S} \mapsto 0 \}, \theta^{\mathsf{c}}{}_{2i})$.

   3.6. By the same arguments in the previous subcase, we may conclude that $\mathit{nc}(\mathbf{r}^{\mathsf{p}}, \theta, \theta^{\mathsf{c}}{}_{2})$.

CASE 11:$\mathbf{r}^{\mathrm{p}}=(\varrho\ \langle\theta^{\mathrm{r}}, \mathrm{WAIT}\rangle.\ \mathbf{p}^{\mathrm{p}})$

As this case is essentially the same as many nested signals, it follows by a similar argument to the previous case.

□

Auxiliary Lemmas.

LEMMA 68 (BLOCKED IS SEPARABLE)**.**

for all $\mathbf{p}$, θ, $\mathbf{A}$, $\mathbf{E}_1$, *and* $\mathbf{E}_2\,\theta$; $\mathbf{A}$; $\mathbf{E}_1 \vdash_{\mathrm{B}} \mathbf{E}_2[\mathbf{p}]$ *implies* θ; $\mathbf{A}$; $\mathbf{E}_1[\mathbf{E}_2] \vdash_{\mathrm{B}} \mathbf{p}$.

PROOF.

Induction on $\mathbf{E}_2$:

CASE 1:$\mathbf{E}_2=\bigcirc$

Trivial, as $\bigcirc[\mathbf{p}] = \mathbf{p}$ and $\mathbf{E}_1[\bigcirc] = \mathbf{E}_1$.

CASE 2:$\mathbf{E}_2=(\mathrm{suspend}\ \mathbf{E}_3\ \mathbf{S})$

Try by induction on the premise of this clause of $\vdash_{\mathrm{B}}$.

CASE 3:$\mathbf{E}_2=(\mathrm{trap}\ \mathbf{E}_3)$

Same as above.

CASE 4:$\mathbf{E}_2=(\mathrm{seq}\ \mathbf{E}_3\ \mathbf{q})$

Same as above.

CASE 5:$\mathbf{E}_2=(\mathrm{par}\ \mathbf{E}_3\ \mathbf{q})$

Same as above.

CASE 6:$\mathbf{E}_2=(\mathrm{par}\ \mathbf{p}\ \mathbf{E}_3)$

Same as above.

□

B.4.2. Negative

Lemma 69 (Non-stepping terms are values)**.**

For all $\mathbf{q}^{\mathrm{p}} = (\varrho\ \langle \theta^{\mathrm{r}}, \mathbf{A}\rangle.\ \mathbf{p}^{\mathrm{p}})$,
If $\mathit{closed}(\mathbf{q}^{\mathrm{p}})$, *and there does not exists any* ${\theta^{\mathrm{r}}}_o$ *and* ${\mathbf{p}^{\mathrm{p}}}_o$ *such that (either* $\mathbf{q}^{\mathrm{p}} \longrightarrow^{\mathrm{R}} (\varrho\ \langle {\theta^{\mathrm{r}}}_o, \mathbf{A}\rangle.\ \mathbf{E}^{\mathrm{p}}[{\mathbf{p}^{\mathrm{p}}}_o])$ *or there exists some* r *such that* $\mathbf{q}^{\mathrm{p}} \longrightarrow^{\mathrm{S}} (\varrho\ \langle \theta^{\mathrm{r}}, \mathbf{A}\rangle.\ \mathbf{E}^{\mathrm{p}}[\mathbf{r}^{\mathrm{p}}]) \longrightarrow^{\mathrm{R}} (\varrho\ \langle \theta_o, \mathbf{A}\rangle.\ \mathbf{E}^{\mathrm{p}}[{\mathbf{p}^{\mathrm{p}}}_o])$*) then either* $\mathbf{p}^{\mathrm{p}} \in \mathbf{p}^{\mathrm{D}}$ *or* $\theta^{\mathrm{r}}; \mathbf{A}; \bigcirc \vdash_{\mathrm{B}} \mathbf{p}^{\mathrm{p}}$

Proof.

As $\vdash_{\mathrm{B}}$ and whether or not a term is $\mathbf{p}^{\mathrm{D}}$ are decidable properties, we may act as if we have the law of the excluded middle here.

Thus we may take the contrapositive of lemma 70 (Not values must step), gives us this exactly. □

Lemma 70 (Not values must step)**.**

For all $\mathbf{q}^{\mathrm{p}} = (\varrho\ \langle \theta^{\mathrm{r}}, \mathbf{A}\rangle.\ \mathbf{E}^{\mathrm{p}}[\mathbf{p}^{\mathrm{p}}])$, *If* $\mathit{closed}(\mathbf{q}^{\mathrm{p}})$, $\mathbf{p}^{\mathrm{p}} \notin \mathbf{p}^{\mathrm{D}}$, *and* $\theta^{\mathrm{r}}; \mathbf{A}; \mathbf{E}^{\mathrm{p}} \nvdash_{\mathrm{B}} \mathbf{p}^{\mathrm{p}}$ *then there exists some* ${\theta^{\mathrm{r}}}_o$ *and* ${\mathbf{p}^{\mathrm{p}}}_o$ *such that either* $\mathbf{q}^{\mathrm{p}} \longrightarrow^{\mathrm{R}} (\varrho\ \langle {\theta}_o, \mathbf{A}\rangle.\ \mathbf{E}^{\mathrm{p}}[{\mathbf{p}^{\mathrm{p}}}_o])$ *or there exists some* $\mathbf{r}^{\mathrm{p}}$ *such that* $\mathbf{q}^{\mathrm{p}} \longrightarrow^{\mathrm{S}} (\varrho\ \langle \theta^{\mathrm{r}}, \mathbf{A}\rangle.\ \mathbf{E}^{\mathrm{p}}[\mathbf{r}^{\mathrm{p}}]) \longrightarrow^{\mathrm{R}} (\varrho\ \langle {\theta^{\mathrm{r}}}_o, \mathbf{A}\rangle.\ \mathbf{E}^{\mathrm{p}}[{\mathbf{p}^{\mathrm{p}}}_o])$

Interpretation. This proof is the contrapositive of the notion that "terms that do not reduce, modulo [**par-swap**], are values". As we are working in a closed universe with decidable properties, the contrapositive will give us our

Proof.

Induction on $\mathbf{p}^{\mathrm{p}}$:

Case 1:$\mathbf{p}^{\mathrm{p}}$=nothing

This case violates our hypothesis that $\mathbf{p}^{\mathrm{p}} \notin \mathbf{p}^{\mathrm{D}}$.

Case 2:$\mathbf{p}^{\mathrm{p}}$=pause

This case violates our hypothesis that $\mathbf{p}^{\mathrm{p}} \notin \mathbf{p}^{\mathrm{D}}$.

Case 3:$\mathbf{p}^{\mathrm{p}}$=(exit $\mathbf{n}$)

This case violates our hypothesis that $\mathbf{p}^{\mathrm{p}} \notin \mathbf{p}^{\mathrm{D}}$.

Case 4:$\mathbf{p}^{\mathrm{p}}$=(emit $\mathbf{S}$)

1. By the definition of *closed*, it must be the case that $\mathbf{A} = \mathrm{GO}$
2. As $\mathit{closed}(\mathbf{q}^{\mathrm{p}})$ and as $\mathbf{E}$ contains no binders, It must be the case that $\mathbf{S} \in \mathit{dom}(\theta^{\mathrm{r}})$.
3. Thus let $\theta_O = \theta \leftarrow \{ \mathbf{S} \mapsto 1 \}$ and $\mathbf{p}^{\mathrm{p}}{}_O = \mathrm{nothing}$. Let the step we take be [**emit**].

Case 5:$\mathbf{p}^{\mathrm{p}}$=(seq $\mathbf{p}^{\mathrm{p}}{}_O$ $\mathbf{q}^{\mathrm{p}}{}_O$)

1. By the definition of $\mathbf{p}^{\mathrm{D}}$, we know that $\mathbf{p}^{\mathrm{p}}{}_O \notin \hat{\mathbf{p}}$.
2. By the definition of $\vdash_{\mathrm{B}}$, we know that $\theta; \mathbf{A}; \mathbf{E}^{\mathrm{p}}[(\mathrm{seq} \bigcirc \mathbf{q}^{\mathrm{p}}{}_O)] \nvdash_{\mathrm{B}} \mathbf{p}^{\mathrm{p}}{}_O$.
3. We by (1) know that $\mathbf{p}^{\mathrm{p}}{}_O$ is not $\hat{\mathbf{p}}$, but we must consider of $\mathbf{p}^{\mathrm{p}}{}_O$ is $\mathbf{p}^{\mathrm{D}}$. By the definition of $\mathbf{p}^{\mathrm{D}}$, this gives:

   Cases of $\mathbf{p}^{\mathrm{p}}{}_O \in \mathbf{p}^{\mathrm{S}}$:

Case 5.1:$\mathbf{p}^{\mathrm{p}}{}_O \in \mathbf{p}^{\mathrm{S}}$

   Cases of $\mathbf{p}^{\mathrm{p}}{}_O$:

Case 5.1.a:$\mathbf{p}^{\mathrm{p}}{}_O$=nothing

   In this case, $\theta^{\mathrm{r}}{}_O = \theta^{\mathrm{r}}$, the resulting $\mathbf{p}^{\mathrm{p}}{}_O = \mathbf{q}^{\mathrm{p}}{}_O$, and we step by [**seq-done**].

Case 5.1.b:$\mathbf{p}^{\mathrm{p}}{}_O$=(exit $\mathbf{n}$)

   In this case, $\theta^{\mathrm{r}}{}_O = \theta$, the resulting $\mathbf{p}^{\mathrm{p}}{}_O$ = (exit $\mathbf{n}$), and we step by [**seq-exit**].

CASE 5.II: $\mathbf{p}_O \notin \mathbf{p}^S$

3.1. As we know that $\mathbf{p}^p{}_O \notin \mathbf{p}^S$ and $\mathbf{p}^p{}_O \notin \hat{\mathbf{p}}$, we know that $\mathbf{p}^p{}_O \notin \mathbf{p}^D$.

3.2. By (3.1), (1), and (2), we can use our induction hypothesis on $\mathbf{p}^p{}_O$ and $\mathbf{E}^p[(\mathrm{seq} \bigcirc \mathbf{q}^p{}_O)]$. As the result we get back differs from what we need only in that $(\mathrm{seq} \bigcirc \mathbf{q}^p{}_O)$ was shifted from $\mathbf{p}^p{}_O$ to $\mathbf{E}$, we can shift it back an return the result unchanged.

CASE 6: $\mathbf{p}^p = (\mathrm{par}\ \mathbf{p}^p{}_O\ \mathbf{q}^p{}_O)$

There is only one way for a par to $\mathbf{p}^D$, thus we know that $\neg(\mathbf{p}^p{}_O \in \hat{\mathbf{p}} \wedge \mathbf{q}^p{}_O \in \hat{\mathbf{p}})$.

There are three ways for an par to be $\vdash_B$, thus we know that

$$\begin{array}{l}\neg((\theta^r;\ \mathbf{A};\ \mathbf{E}^p[(\mathrm{par} \bigcirc \mathbf{q}^p{}_O)] \vdash_B \mathbf{p}^p{}_O \wedge \theta^r;\ \mathbf{A};\ \mathbf{E}^p[(\mathrm{par}\ \mathbf{p}^p{}_O \bigcirc)] \vdash_B \mathbf{q}^p{}_O) \\ \quad \vee \\ \quad (\theta^r;\ \mathbf{A};\ \mathbf{E}^p[(\mathrm{par} \bigcirc \mathbf{q}^p{}_O)] \vdash_B \mathbf{p}^p{}_O \wedge \mathbf{q}^p{}_O \in \mathbf{p}^D) \\ \quad \vee \\ \quad (\mathbf{p}^p{}_O \in \mathbf{p}^D \wedge \theta;\ \mathbf{A};\ \mathbf{E}[(\mathrm{par}\ \mathbf{p}^p{}_O \bigcirc)] \vdash_B \mathbf{q}^p{}_O))\end{array}$$

On the whole this gives us the following expression:

$$\begin{array}{l}\neg(\mathbf{p}^p{}_O \in \hat{\mathbf{p}} \wedge \mathbf{q}^p{}_O \in \hat{\mathbf{p}}) \\ \wedge \\ \neg((\theta;\ \mathbf{A};\ \mathbf{E}^p[(\mathrm{par} \bigcirc \mathbf{q}^p{}_O)] \vdash_B \mathbf{p}^p{}_O \wedge \theta^r;\ \mathbf{A};\ \mathbf{E}[(\mathrm{par}\ \mathbf{p}^p{}_O \bigcirc)] \vdash_B \mathbf{q}^p{}_O) \\ \quad \vee \\ \quad (\theta;\ \mathbf{A};\ \mathbf{E}^p[(\mathrm{par} \bigcirc \mathbf{q}^p{}_O)] \vdash_B \mathbf{p}^p{}_O \wedge \mathbf{q}^p{}_O \in \mathbf{p}^D) \\ \quad \vee \\ \quad (\mathbf{p}^p{}_O \in \mathbf{p}^D \wedge \theta^r;\ \mathbf{A};\ \mathbf{E}[(\mathrm{par}\ \mathbf{p}^p{}_O \bigcirc)] \vdash_B \mathbf{q}^p{}_O))\end{array}$$

Note that a term which is $\hat{\mathbf{p}}$ is also $\mathbf{p}^D$. Given this, we can find the the disjuctive normal form of the above expression, giving us four cases:

$$\theta^r;\ \mathbf{A};\ \mathbf{E}^p[(\mathrm{par} \bigcirc \mathbf{q}^p{}_O)] \nvdash_B \mathbf{p}^p{}_O \wedge \theta^r;\ \mathbf{A};\ \mathbf{E}[(\mathrm{par}\ \mathbf{p}^p{}_O \bigcirc)] \nvdash_B \mathbf{q}^p{}_O \wedge \mathbf{p}^p{}_O \notin \hat{\mathbf{p}}$$

$$\theta^r;\ \mathbf{A};\ \mathbf{E}^p[(\mathrm{par} \bigcirc \mathbf{q}^p{}_O)] \nvdash_B \mathbf{p}^p{}_O \wedge \theta^r;\ \mathbf{A};\ \mathbf{E}^p[(\mathrm{par}\ \mathbf{p}^p{}_O \bigcirc)] \nvdash_B \mathbf{q}^p{}_O \wedge \mathbf{q}^p{}_O \notin \hat{\mathbf{p}}$$

$\theta^{r}; \mathbf{A}; \mathbf{E}^{p}[(\mathrm{par} \bigcirc \mathbf{q}^{p}{}_{o})] \nrightarrow_{B} \mathbf{p}^{p}{}_{o} \wedge \mathbf{p}^{p}{}_{o} \notin \mathbf{p}^{D}$

$\theta^{r}; \mathbf{A}; \mathbf{E}^{p}[(\mathrm{par}\ \mathbf{p}^{p}{}_{o} \bigcirc)] \nrightarrow_{B} \mathbf{q}^{p}{}_{o} \wedge \mathbf{q}^{p}{}_{o} \notin \mathbf{p}^{D}$

Cases of the above:

Case 6.i: $\theta^{r}; \mathbf{A}; \mathbf{E}[(\mathrm{par} \bigcirc \mathbf{q}^{p}{}_{o})] \nrightarrow_{B} \mathbf{p}^{p}{}_{o} \wedge \theta^{r}; \mathbf{A}; \mathbf{E}^{p}[(\mathrm{par}\ \mathbf{p}^{p}{}_{o} \bigcirc)] \nrightarrow_{B} \mathbf{q}^{p}{}_{o} \wedge \mathbf{q}^{p}{}_{o} \notin \hat{\mathbf{p}}$

In this case we have

1. We know that $\theta^{r}; \mathbf{A}; \mathbf{E}[(\mathrm{par} \bigcirc \mathbf{p}^{p}{}_{o})] \nrightarrow_{B} \mathbf{q}^{p}{}_{o}$

2. We know that $\mathbf{q}^{p}{}_{o} \notin \hat{\mathbf{p}}$

3. Cases of $\mathbf{q}^{p}{}_{o} \in \mathbf{p}^{S}$:

Case 6.i.a: $\mathbf{q}^{p}{}_{o} \in \mathbf{p}^{S}$

Cases of $\mathbf{p}^{p}{}_{o} \in \mathbf{p}^{D}$:

Case 6.i.a.1: $\mathbf{p}^{p}{}_{o} \in \mathbf{p}^{D}$

Cases of $\langle \mathbf{p}^{p}{}_{o}, \mathbf{q}^{p}{}_{o} \rangle$:

Case 6.i.a.1.i: $\langle \mathbf{p}^{p}{}_{o}, \mathbf{q}^{p}{}_{o} \rangle = \langle \mathrm{nothing}, (\mathrm{exit}\ \mathbf{n}_{2}) \rangle$

In this case let $\theta^{r}{}_{o} = \theta^{r}$ and $\mathbf{p}^{p}{}_{o} = (\mathrm{exit}\ \mathbf{n}_{2})$, and our reduction be a single use of **[par-nothing]**.

Case 6.i.a.1.ii: $\langle \mathbf{p}^{p}{}_{o}, \mathbf{q}^{p}{}_{o} \rangle = \langle (\mathrm{exit}\ \mathbf{n}_{1}), (\mathrm{exit}\ \mathbf{n}_{2}) \rangle$

In this case let $\theta^{r}{}_{o} = \theta^{r}$ and $\mathbf{p}^{p}{}_{o} = (\mathrm{exit}\ (\mathrm{max}\ \mathbf{n}_{1}\ \mathbf{n}_{2}))$, and our reduction be a single use of **[par-2exit]**.

Case 6.i.a.1.iii: $\langle \mathbf{p}^{p}{}_{o}, \mathbf{q}^{p}{}_{o} \rangle = \langle \hat{\mathbf{p}}, (\mathrm{exit}\ \mathbf{n}_{2}) \rangle$

In this case let $\theta^{r}{}_{o} = \theta^{r}$, $\mathbf{p}^{p}{}_{o} = (\mathrm{exit}\ \mathbf{n}_{2})$, and $\mathbf{r}^{p} = (\mathrm{par}\ (\mathrm{exit}\ \mathbf{n}_{2})\ \hat{\mathbf{p}})$. Our reductions are one use of **[par-swap]** and one use of **[par-1exit]**.

Case 6.i.a.1.iv: $\langle \mathbf{p}^{p}{}_{o}, \mathbf{q}^{p}{}_{o} \rangle = \langle \mathbf{p}^{D}, \mathrm{nothing} \rangle$

This is analogues to the previous case, but using **[par-nothing]** instead of **[par-1exit]**.

Case 6.i.a.2:$\mathbf{p}^{\mathrm{p}}{}_{o} \notin \mathbf{p}^{\mathrm{D}}$

In this case we may invoke our induction hypothesis on $\mathbf{p}^{\mathrm{p}}{}_{o}$, as we know $\theta^{\mathrm{r}}; \mathbf{A}; \mathbf{E}^{\mathrm{p}}[(\mathrm{par} \bigcirc \mathbf{q}^{\mathrm{p}}{}_{o})] \nvdash_{\mathrm{B}} \mathbf{p}^{\mathrm{p}}{}_{o}$ and $\mathbf{p}^{\mathrm{p}}{}_{o} \notin \mathbf{p}^{\mathrm{D}}$. As usual we may reassemble the result of the induction by shifting one frame of the $\mathbf{E}$ back over and using the same reductions.

Case 6.i.b:$\mathbf{q}^{\mathrm{p}}{}_{o} \notin \mathbf{p}^{\mathrm{S}}$

3.1.As we know $\mathbf{q}^{\mathrm{p}}{}_{o} \notin \mathbf{p}^{\mathrm{S}}$ and $\mathbf{q}^{\mathrm{p}}{}_{o} \notin \hat{\mathbf{p}}$, we know $\mathbf{q}^{\mathrm{p}}{}_{o} \notin \mathbf{p}^{\mathrm{D}}$.

3.2.We may invoke our induction hypothesis on $\mathbf{E}^{\mathrm{p}}[(\mathrm{par}\ \mathbf{p}^{\mathrm{p}}{}_{o} \bigcirc)]$ and $\mathbf{q}^{\mathrm{p}}{}_{o}$. This gives us that exists some $\theta^{\mathrm{r}}{}_{o1}$ and $\mathbf{p}^{\mathrm{p}}{}_{o1}$ using (1) and (3.1) such that $\mathbf{q}^{\mathrm{p}} \longrightarrow^{\mathrm{R}} (\varrho\ \langle \theta^{\mathrm{r}}{}_{o1}, \mathbf{A} \rangle.\ \mathbf{E}^{\mathrm{p}}[\mathbf{p}^{\mathrm{p}}{}_{o1}])$ or there exists some r such that $\mathbf{q}^{\mathrm{p}} \longrightarrow^{\mathrm{S}} \mathbf{r}^{\mathrm{p}} \longrightarrow^{\mathrm{R}} (\varrho\ \langle \theta^{\mathrm{r}}{}_{o1}, \mathbf{A} \rangle.\ \mathbf{E}^{\mathrm{p}}[(\mathrm{par}\ \mathbf{p}^{\mathrm{p}}{}_{o} \bigcirc)][\mathbf{p}^{\mathrm{p}}{}_{o1}])$.

3.3.In this case we can take the result from (3.2) and shift The $(\mathrm{par}\ \mathbf{p}^{\mathrm{p}}{}_{o} \bigcirc)$ over to $\mathbf{p}^{\mathrm{p}}{}_{o1}$. Giving us a resulting $\mathbf{p}^{\mathrm{p}}{}_{o2} = (\mathrm{par}\ \mathbf{p}^{\mathrm{p}}{}_{o}\ \mathbf{p}^{\mathrm{p}}{}_{o1})$, and an unchanged $\theta^{\mathrm{r}}{}_{o1}$. As the overall terms have not changed, the reductions form (3.2) are unchanged. Thus we return those reductions.

Case 6.ii:$\theta^{\mathrm{r}}; \mathbf{A}; \mathbf{E}^{\mathrm{p}}[(\mathrm{par} \bigcirc \mathbf{q}^{\mathrm{p}}{}_{o})] \nvdash_{\mathrm{B}} \mathbf{p}^{\mathrm{p}}{}_{o} \wedge \theta^{\mathrm{r}}; \mathbf{A}; \mathbf{E}^{\mathrm{p}}[(\mathrm{par}\ \mathbf{p}^{\mathrm{p}}{}_{o} \bigcirc)] \nvdash_{\mathrm{B}} \mathbf{q}^{\mathrm{p}}{}_{o} \wedge \mathbf{p}^{\mathrm{p}}{}_{o} \notin \hat{\mathbf{p}}$

This case is the same as the previous, but finding a reduction in the other branch. As in this case in the subcase where we concider one branch begin $\mathbf{p}^{\mathrm{D}}$ and the other being $\mathbf{p}^{\mathrm{S}}$, $\mathbf{p}^{\mathrm{p}}{}_{o}$ is the branch that is $\mathbf{p}^{\mathrm{S}}$ we do not need to use $\longrightarrow^{\mathrm{S}}$.

Case 6.iii:$\theta^{\mathrm{r}}; \mathbf{A}; \mathbf{E}^{\mathrm{p}}[(\mathrm{par} \bigcirc \mathbf{q}^{\mathrm{p}}{}_{o})] \nvdash_{\mathrm{B}} \mathbf{p}^{\mathrm{p}}{}_{o} \wedge \mathbf{p}^{\mathrm{p}}{}_{o} \notin \mathbf{p}^{\mathrm{D}}$

As we know that $\mathbf{p}^{\mathrm{p}}{}_{o}$ is not $\vdash_{\mathrm{B}}$ or $\mathbf{p}^{\mathrm{D}}$ we can induct on $\mathbf{p}^{\mathrm{p}}{}_{o}$ and $\mathbf{E}^{\mathrm{p}}[(\mathrm{par} \bigcirc \mathbf{q}^{\mathrm{p}}{}_{o})]$. As usual we may reassemble the result of the induction by shifting one frame of the $\mathbf{E}^{\mathrm{p}}$ back over and using the same reductions.

Case 6.iv:$\theta^{\mathrm{r}}; \mathbf{A}; \mathbf{E}^{\mathrm{p}}[(\mathrm{par}\ \mathbf{p}^{\mathrm{p}}{}_{o} \bigcirc)] \nvdash_{\mathrm{B}} \mathbf{q}^{\mathrm{p}}{}_{o} \wedge \mathbf{q}^{\mathrm{p}}{}_{o} \notin \mathbf{p}^{\mathrm{D}}$

This case is the same as the previous, but finding a reduction in the other branch.

CASE 7:$\mathbf{p}^{\mathrm{p}}=(\mathrm{trap}\ \mathbf{p}^{\mathrm{p}}{}_{o})$

1. By the definition of $\mathbf{p}^{\mathrm{D}}$ we know that $\mathbf{p}_{o} \notin \hat{\mathbf{p}}$.

2. By the definition of $\vdash_{\mathrm{B}}$ we know that $\theta^{\mathrm{r}}; \mathbf{A}; \mathbf{E}^{\mathrm{p}}[(\mathrm{trap}\ \bigcirc)] \nvdash_{\mathrm{B}} \mathbf{p}^{\mathrm{p}}{}_{o}$.

3. Cases of $\mathbf{p}^{\mathrm{p}}{}_{o} \in \mathbf{p}^{\mathrm{S}}$:

CASE 7.I:$\mathbf{p}^{\mathrm{p}}{}_{o} \in \mathbf{p}^{\mathrm{S}}$

In this case we may reduced by $[\mathbf{trap}]$.

CASE 7.II:$\mathbf{p}^{\mathrm{p}}{}_{o} \notin \mathbf{p}^{\mathrm{S}}$

3.1. Given $\mathbf{p}^{\mathrm{p}}{}_{o} \notin \mathbf{p}^{\mathrm{S}}$ and (1), we know that $\mathbf{p}^{\mathrm{p}}{}_{o} \notin \mathbf{p}^{\mathrm{D}}$

3.2. Given (3.1) and (2) we may use our induction hypothesis on $\mathbf{p}_{o}$ and $\mathbf{E}^{\mathrm{p}}[(\mathrm{trap}\ \bigcirc)]$. As usual we may re-assemble the result of the induction by shifting one frame of the $\mathbf{E}^{\mathrm{p}}$ back over and using the same reductions.

CASE 8:$\mathbf{p}^{\mathrm{p}}=(\mathrm{suspend}\ \mathbf{p}^{\mathrm{p}}{}_{o}\ \mathbf{S})$

This case is analogous to the previous case but reducing by $[\mathbf{suspend}]$ rather than $[\mathbf{trap}]$.

CASE 9:$\mathbf{p}^{\mathrm{p}}=(\mathrm{signal}\ \mathbf{S}\ \mathbf{p}^{\mathrm{p}}{}_{o})$

In this case we may reduce by $[\mathbf{signal}]$.

CASE 10:$\mathbf{p}^{\mathrm{p}}=(\varrho\ \langle\theta^{\mathrm{r}}{}_{o}, \mathbf{A}_{o}\rangle.\ \mathbf{p}^{\mathrm{p}}{}_{o})$

In this case we may reduce by $[\mathbf{merge}]$, as by $\mathit{closed}(\mathbf{q})$ we know that $\mathbf{A} = \mathrm{GO}$, which must be $\geq \mathbf{A}_{o}$.

CASE 11:$\mathbf{p}^{\mathrm{p}}=(\mathrm{if}\ \mathbf{S}\ \mathbf{p}^{\mathrm{p}}{}_{o}\ \mathbf{q}^{\mathrm{p}}{}_{o})$

By $\mathit{closed}(\mathbf{q}^{\mathrm{p}})$ and $\mathbf{E}$ not containing any binders we know that $\mathbf{S} \in \mathit{dom}(\theta^{\mathrm{r}})$.

Cases of $\theta^{\mathrm{r}}(\mathbf{S})$:

CASE 11.I: $\theta^{r}(\mathbf{S})=0$

Signals bound in a ϱ must not be 0, therefore this case is not possible.

CASE 11.II: $\theta^{r}(\mathbf{S})=1$

In this case we may reduce by [**is-present**].

CASE 11.III: $\theta^{r}(\mathbf{S})=\bot$

In this case, by $\theta^{r}; \mathbf{A}; \mathbf{E} \kappa_{B} \mathbf{p}^{p}{}_{o}$, It must be the case that $\mathbf{S} \notin \mathit{Can}^{S}_{\varrho}((\varrho\ \langle\theta, \mathbf{A}\rangle.\ \mathbf{E}[(\text{if } \mathbf{S}\ \mathbf{p}^{p}{}_{o}\ \mathbf{q}^{p}{}_{o})]), \{\})$. Therefore we may reduce by [**is-absent**].

□

B.5. Can Properties

This section contains lemmas and proofs about *Can* and its relation to the circuit translation. The core theorem here is lemma 71 (CAN S IS SOUND).

LEMMA 71 (CAN S IS SOUND)**.**

For any term and environment $\mathbf{p}^{p}$ *and* θ *and any signal* $\mathbf{S}$, *if* $[\![\mathbf{p}^{p}]\!] \setminus \theta$, $\mathbf{S} \notin \mathit{Can}^{S}(\mathbf{p}^{p}, \theta)$, *and* $[\![\mathbf{p}^{p}]\!](\text{SEL}) \simeq 0$, *then* $[\![\mathbf{p}^{p}]\!](\mathbf{S}^{o}) \simeq 0$

INTERPRETATION. This theorem states that *Can* accurately predicts when signal output wires will be set to 0.

PROOF.

Induction on $\mathbf{p}^{p}$:

CASE 1: $\mathbf{p}^{p}$=nothing

There is no $\mathbf{S}^{o}$ wire, thus is it by definition 0.

CASE 2:$\mathbf{p}^{\mathrm{p}}$=pause

Same as previous case.

CASE 3:$\mathbf{p}^{\mathrm{p}}$=(exit $\mathbf{n}$)

Same as previous case.

CASE 4:$\mathbf{p}^{\mathrm{p}}$=(emit $\mathbf{S}_2$)

Cases of $\mathbf{S} = \mathbf{S}_2$:

CASE 4.I:$\mathbf{S} = \mathbf{S}_2$

In this case $\mathbf{S} \in \mathit{Can}^{\mathrm{S}}(\mathbf{p}^{\mathrm{p}}, \theta)$, which violates our hypothesis.

CASE 4.II:$\mathbf{S} \neq \mathbf{S}_2$

In this case $[\![(\text{emit } \mathbf{S}_2)]\!]$ does define an $\mathbf{S}_2$ wire, therefore $\mathbf{S} \notin \mathit{outputs}([\![(\text{emit } \mathbf{S}_2)]\!])$, Therefore, by definition, $[\![(\text{emit } \mathbf{S}_2)]\!](\mathbf{S})$ is 0.

CASE 5:$\mathbf{p}^{\mathrm{p}}$=(signal $\mathbf{S}_2\, \mathbf{p}^{\mathrm{p}}{}_i$)

Cases of $\mathbf{S} = \mathbf{S}_2$:

CASE 5.I:$\mathbf{S} = \mathbf{S}_2$

In this case the compilation of signal removes $\mathbf{S}$ from the set of output signals, which means

$\mathbf{S}^{\mathrm{o}} \notin \mathit{outputs}([\![\mathbf{p}^{\mathrm{p}}]\!])$, and therefore once again it must be 0.

CASE 5.II:$\mathbf{S} \neq \mathbf{S}_2$

In this case we can see that there are two cases for Can:

Cases of $\mathbf{S}_2 \in \mathit{Can}^{\mathrm{S}}(\mathbf{p}^{\mathrm{p}}{}_i, \theta \leftarrow \{\mathbf{S}_2 \mapsto \bot\})$:

CASE 5.II.A:$\mathbf{S}_2 \in \mathit{Can}^{\mathrm{S}}(\mathbf{p}^{\mathrm{p}}{}_i, \theta \leftarrow \{\mathbf{S}_2 \mapsto \bot\})$

1. By the definition of Can, $\mathit{Can}^S(\mathsf{(signal\ S_2\ p^p_i)}, \theta) = \mathit{Can}^S(\mathsf{p}^p_i, \theta \leftarrow \{ \mathsf{S}_2 \mapsto \bot \}) \setminus \{ \mathsf{S} \}$.

2. By $\mathsf{S} \neq \mathsf{S}_2$, (1), and the premise that $\mathsf{S} \notin \mathit{Can}^S(\mathsf{p}^p, \theta)$, we may conclude that $\mathsf{S} \notin \mathit{Can}^S(\mathsf{p}^p_i, \theta \leftarrow \{ \mathsf{S}_2 \mapsto \bot \})$.

3. By the definition of $[\![\cdot]\!]$, the compilation of signal will link S^o_2 to S^i_2. Therefore we can conclude that $[\![\mathsf{p}^p_i]\!] \setminus \theta \leftarrow \{ \mathsf{S}_2 \mapsto \bot \}$, as the binding of the other signals does not change, and $\mathsf{p}^p \setminus \{ \mathsf{S}_2 \mapsto \bot \}$ will always hold.

4. As the compilation of signal does not change SEL, we may use (2) and (3) to invoke our induction hypothesis to conclude that $[\![\mathsf{p}^p_i]\!](\mathsf{S}^o) = 0$

5. By $\mathsf{S} \neq \mathsf{S}_2$ and definition of the compilation of signal, we know that the S^o wire will remain unchanged. Therefore we can conclude that $[\![\mathsf{p}^p_i]\!](\mathsf{S}_o) = [\![\mathsf{(signal\ S_2\ p^p_i)}]\!](\mathsf{S}_o) = 0$.

Case 5.ii.b:$\mathsf{S}_2 \notin \mathit{Can}^S(\mathsf{p}^p_i, \theta \leftarrow \{ \mathsf{S}_2 \mapsto \bot \})$

In this case $\mathit{Can}^S(\mathsf{(signal\ S_2\ p_i)}, \theta) = \mathit{Can}^S(\mathsf{p}^p_i, \theta \leftarrow \{ \mathsf{S}_2 \mapsto 0 \}) \setminus \{ \mathsf{S} \}$. The argument for this case follows exactly along the same lines as the previous case, but we must instead show that $[\![\mathsf{p}^p_i]\!] \setminus \theta \leftarrow \{ \mathsf{S}_2 \mapsto 0 \}$ rather than $[\![\mathsf{p}^p_i]\!] \setminus \theta \leftarrow \{ \mathsf{S}_2 \mapsto \bot \}$. To show this we first argue that $[\![\mathsf{p}^p_i]\!] \setminus \theta \leftarrow \{ \mathsf{S}_2 \mapsto \bot \}$ still holds, as it replaces no restrictions on the value of S_2. From this we can apply induction using $\mathsf{S}_2 \notin \mathit{Can}^S(\mathsf{p}^p_i, \theta \leftarrow \{ \mathsf{S}_2 \mapsto \bot \})$ to argue that $[\![\mathsf{p}^p_i]\!](\mathsf{S}_2) = 0$. Since this is the interpretation of $\{ \mathsf{S}_2 \mapsto 0 \}$, we can safely conclude that $[\![\mathsf{p}^p_i]\!] \setminus \theta \leftarrow \{ \mathsf{S}_2 \mapsto 0 \}$. Thus we can apply the reasoning from the previous case to conclude that $[\![\mathsf{(signal\ S_2\ p^p_i)}]\!](\mathsf{S}_o) = 0$.

Case 6:p^p=$\mathsf{(par\ p^p_i\ q^p_i)}$

In this case $\mathit{Can}^S(\mathsf{(par\ p^p_i\ q^p_i)}, \theta) = \mathit{Can}^S(\mathsf{p}^p_i, \theta) \cup \mathit{Can}^S(\mathsf{q}^p_i, \theta)$. This we can conclude that $\mathsf{S} \notin \mathit{Can}^S(\mathsf{p}^p_i, \theta)$ and $\mathsf{S} \notin \mathit{Can}^S(\mathsf{q}^p_i, \theta)$. We also know that $[\![\mathsf{(par\ p^p_i\ q^p_i)}]\!](\mathrm{SEL}) = [\![\mathsf{p}^p_i]\!](\mathrm{SEL}) \vee [\![\mathsf{q}^p_i]\!](\mathrm{SEL})$, which implies that $[\![\mathsf{p}^p_i]\!](\mathrm{SEL}) \simeq 0$ and $[\![\mathsf{q}^p_i]\!](\mathrm{SEL}) \simeq 0$.

Therefore, by induction, we can conclude that $[\![\mathsf{p}^p_i]\!](\mathsf{S}_o) \simeq 0$ and $[\![\mathsf{q}^p_i]\!](\mathsf{S}_o) \simeq 0$.

As both the overall output of S_o of both subcircuits is 0, and by definition $[\![\mathsf{(par\ p^p_i\ q^p_o)}]\!](\mathsf{S}_o) = [\![\mathsf{p}_i]\!](\mathsf{S}_o) \vee [\![\mathsf{q}_i]\!](\mathsf{S}_o)$, it must be that $[\![\mathsf{(par\ p^p_i\ q^p_o)}]\!](\mathsf{S}_o) \simeq 0$.

CASE 7:$\mathbf{p}^p$=(if $\mathbf{S}_2\ \mathbf{p}^p{}_i\ \mathbf{q}^p{}_i$)

We know that $[\![(\text{if } \mathbf{S}_2\ \mathbf{p}^p{}_i\ \mathbf{q}^p{}_i)]\!](\text{SEL}) = [\![\mathbf{p}^p{}_i]\!](\text{SEL}) \vee [\![\mathbf{q}^p{}_i]\!](\text{SEL}) = 0$. Therefore

$[\![\mathbf{p}^p{}_i]\!](\text{SEL}) \simeq 0$ and $[\![\mathbf{q}^p{}_i]\!](\text{SEL}) \simeq 0$. We also know that $[\![(\text{if } \mathbf{S}_2\ \mathbf{p}^p{}_i\ \mathbf{q}^p{}_i)]\!](\mathbf{S}_o) = [\![\mathbf{p}^p{}_i]\!](\mathbf{S}_o) \vee [\![\mathbf{q}^p{}_i]\!](\mathbf{S}_o)$.

Cases of $\theta(\mathbf{S}_2)$:

CASE 7.I: $\theta(\mathbf{S}_2)$=1

In this case we know that $Can^S(\text{(if } \mathbf{S}_2\ \mathbf{p}^p{}_i\ \mathbf{q}^p{}_i), \theta) = Can^S(\mathbf{p}^p{}_i, \theta)$. In addition not that $\theta(\mathbf{S}_2) = 1$ means that $[\![(\text{if } \mathbf{S}_2\ \mathbf{p}^p{}_i\ \mathbf{q}^p{}_i)]\!](\mathbf{S}^i{}_2) \simeq 1$, therefore

$[\![\mathbf{q}^p{}_i]\!](\text{GO}) \simeq [\![(\text{if } \mathbf{S}_2\ \mathbf{p}^p{}_i\ \mathbf{q}^p{}_i)]\!](\text{GO}) \wedge \neg[\![(\text{if } \mathbf{S}_2\ \mathbf{p}^p{}_i\ \mathbf{q}^p{}_i)]\!](\mathbf{S}^i{}_2)$, which must be 0. Therefore by lemma 76 (ACTIVATION CONDITION) we know that $[\![\mathbf{q}^p{}_i]\!](\mathbf{S}^o) \simeq 0$.

Then we may invoke our induction hypothesis to show that $[\![\mathbf{p}^p{}_i]\!](\mathbf{S}^o) \simeq 0$. Thus we can use a similar chain of reasoning to the previous case to argue that $[\![(\text{if } \mathbf{S}_2\ \mathbf{p}^p{}_i\ \mathbf{q}^p{}_i)]\!](\mathbf{S}^o) \simeq 0$.

CASE 7.II: $\theta(\mathbf{S}_2)$=0

This case is analogous to the previous one, except that the branches switch roles.

CASE 7.III: $\theta(\mathbf{S}_2)$=$\perp$

In this case we know that $Can^S(\text{(if } \mathbf{S}_2\ \mathbf{p}^p{}_i\ \mathbf{q}^p{}_i), \theta) = Can^S(\mathbf{p}^p{}_i, \theta) \cup Can^S(\mathbf{q}^p{}_i, \theta)$. Thus we can conclude that $\mathbf{S} \notin Can^S(\mathbf{p}^p{}_i, \theta)$ and $\mathbf{S} \notin Can^S(\mathbf{q}^p{}_i, \theta)$. As previously we know that $\mathbf{S}^o$ must be in the outputs of $[\![\mathbf{p}^p{}_i]\!]$ or $[\![\mathbf{q}^p{}_i]\!]$. Thus, by induction, $[\![\mathbf{p}_i]\!](\mathbf{S}^o) \simeq 0$ and $[\![\mathbf{q}_i]\!](\mathbf{S}^o) \simeq 0$.

From this we may conclude that $[\![(\text{if } \mathbf{S}_2\ \mathbf{p}^p{}_i\ \mathbf{q}^p{}_i)]\!](\mathbf{S}^o) \simeq 0$.

CASE 8:$\mathbf{p}^p$=(suspend $\mathbf{p}^p{}_i\ \mathbf{S}_2$)

In this case we know that $Can^S(\text{(suspend } \mathbf{p}^p{}_i\ \mathbf{S}_2), \theta) = Can^S(\mathbf{p}^p{}_i, \theta)$. We also know that

$[\![(\text{suspend } \mathbf{p}^p{}_i\ \mathbf{S}_2)]\!](\text{SEL}) = [\![\mathbf{p}^p{}_i]\!](\text{SEL})$. Therefor by induction $[\![\mathbf{p}_i]\!](\mathbf{S}^o) \simeq 0$. Finally the compilation of suspend does not change output signals so we can conclude that $[\![(\text{suspend } \mathbf{p}_i\ \mathbf{S}_2)]\!](\mathbf{S}^o) \simeq 0$.

CASE 9:$\mathbf{p}^{\mathrm{p}}=(\mathrm{trap}\ \mathbf{p}^{\mathrm{p}}_{i})$

This case follows identically to the previous one, as the compilation of trap neither modifies SEL nor signals form its inner term.

CASE 10:$\mathbf{p}^{\mathrm{p}}=(\mathrm{seq}\ \mathbf{p}^{\mathrm{p}}_{i}\ \mathbf{q}^{\mathrm{p}}_{i})$

Note that $[\![(\mathrm{seq}\ \mathbf{p}^{\mathrm{p}}_{i}\ \mathbf{q}^{\mathrm{p}}_{i})]\!](\mathrm{SEL}) = [\![\mathbf{p}^{\mathrm{p}}_{i}]\!](\mathrm{SEL}) \vee [\![\mathbf{q}^{\mathrm{p}}_{i}]\!](\mathrm{SEL})$. Therefore $[\![\mathbf{p}^{\mathrm{p}}_{i}]\!](\mathrm{SEL}) = [\![\mathbf{q}^{\mathrm{p}}_{i}]\!](\mathrm{SEL}) = 0$. From the definition of *Can* we have two cases: Cases of $0 \in \mathit{Can}^{\mathrm{K}}(\mathbf{p}^{\mathrm{p}}_{i}, \theta)$:

CASE 10.I:$0 \in \mathit{Can}^{\mathrm{K}}(\mathbf{p}^{\mathrm{p}}_{i}, \theta) = 0 \notin \mathit{Can}^{\mathrm{K}}(\mathbf{p}^{\mathrm{p}}_{i}, \theta)$

In this case we have $\mathit{Can}^{\mathrm{S}}((\mathrm{seq}\ \mathbf{p}^{\mathrm{p}}_{i}\ \mathbf{q}^{\mathrm{p}}_{i}), \theta) = \mathit{Can}^{\mathrm{S}}(\mathbf{p}^{\mathrm{p}}_{i}, \theta)$. By induction we can conclude $[\![\mathbf{p}^{\mathrm{p}}_{i}]\!](\mathbf{S}^{\mathrm{o}}) \simeq 0$. In addition by lemma 72 (CAN K IS SOUND) we can conclude that $[\![\mathbf{p}^{\mathrm{p}}_{i}]\!](\mathrm{K0}) \simeq 0$. This tells us that $[\![\mathbf{q}^{\mathrm{p}}_{i}]\!](\mathrm{GO}) \simeq 0$. Thus, by lemma 76 (ACTIVATION CONDITION), we can conclude that $[\![\mathbf{q}^{\mathrm{p}}_{i}]\!](\mathbf{S}^{\mathrm{o}}) \simeq 0$. As $\mathbf{S}_{O}$ is either no in the outputs or 0 in either subcircuit, we can conclude that $[\![(\mathrm{seq}\ \mathbf{p}^{\mathrm{p}}_{i}\ \mathbf{q}^{\mathrm{p}}_{i})]\!](\mathbf{S}^{\mathrm{o}}) \simeq 0$.

CASE 10.II:$0 \in \mathit{Can}^{\mathrm{K}}(\mathbf{p}^{\mathrm{p}}_{i}, \theta) = 0 \in \mathit{Can}^{\mathrm{K}}(\mathbf{p}^{\mathrm{p}}_{i}, \theta)$

In this case we have $\mathit{Can}^{\mathrm{S}}((\mathrm{seq}\ \mathbf{p}^{\mathrm{p}}_{i}\ \mathbf{q}^{\mathrm{p}}_{i}), \theta) = \mathit{Can}^{\mathrm{S}}(\mathbf{p}^{\mathrm{p}}_{i}, \theta) \cup \mathit{Can}^{\mathrm{S}}(\mathbf{q}^{\mathrm{p}}_{i}, \theta)$. From this we know that

$\mathbf{S} \notin \mathit{Can}^{\mathrm{S}}(\mathbf{p}^{\mathrm{p}}_{i}, \theta)$ and $\mathbf{S} \notin \mathit{Can}^{\mathrm{S}}(\mathbf{q}^{\mathrm{p}}_{i}, \theta)$. Thus, by induction we have that $[\![\mathbf{p}^{\mathrm{p}}_{i}]\!](\mathbf{S}^{\mathrm{o}}) \simeq 0$ and that $[\![\mathbf{q}^{\mathrm{p}}_{i}]\!](\mathbf{S}_{O}) \simeq 0$. As $\mathbf{S}^{\mathrm{o}}$ is 0 in both subcircuits, we can conclude that $[\![(\mathrm{seq}\ \mathbf{p}^{\mathrm{p}}_{i}\ \mathbf{q}^{\mathrm{p}}_{i})]\!](\mathbf{S}^{\mathrm{o}}) \simeq 0$.

CASE 11:$\mathbf{p}^{\mathrm{p}}=(\varrho\ \langle\theta^{\mathrm{r}}, \mathbf{A}\rangle.\ \mathbf{p}_{i})$

This case is shown by lemma 73 (CAN RHO S IS SOUND).

□

LEMMA 72 (CAN K IS SOUND)**.**

For any term and environment $\mathbf{p}^{\mathrm{p}}$ *and* θ *and any return code* $\boldsymbol{\kappa}$, *if* $[\![\mathbf{p}^{\mathrm{p}}]\!] \setminus \theta$, $\boldsymbol{\kappa} \notin \mathit{Can}^{\mathrm{K}}(\mathbf{p}^{\mathrm{p}}, \theta)$, *and* $[\![\mathbf{p}^{\mathrm{p}}]\!](\mathrm{SEL}) \simeq 0$, *then* $[\![\mathbf{p}^{\mathrm{p}}]\!](\mathrm{K}\boldsymbol{\kappa}) \simeq 0$

INTERPRETATION. This theorem states that *Can* accurately predicts when control wires will be set to 0.

PROOF.

Induction on $\mathbf{p}^{\mathrm{p}}$:

CASE 1:$\mathbf{p}^{\mathrm{p}}$=nothing

Note that in this case $\mathit{Can}^{\mathrm{K}}(\mathrm{nothing}, \theta) = \{ 0 \}$. Cases of (es $\kappa = 0$):

CASE 1.I:$\kappa = 0$

In this case $\kappa \in \mathit{Can}^{\mathrm{K}}(\mathbf{p}, \theta)$ which violates our hypothesis.

CASE 1.II:$\kappa \neq 0$

There is no Kκ wire in this case, thus it is by definition 0.

CASE 2:$\mathbf{p}^{\mathrm{p}}$=(emit **S**)

This is the same as the previous case.

CASE 3:$\mathbf{p}^{\mathrm{p}}$=(exit **n**)

This is the same as the previous two cases, but with **n** substituted for 0.

CASE 4:$\mathbf{p}^{\mathrm{p}}$=pause

Note that $\mathit{Can}^{\mathrm{K}}(\mathrm{pause}, \theta) = \{ 1 \}$. In the only K other wire in $[\![\mathrm{pause}]\!]$ is K0, so we need only concern ourselves with that. $[\![\mathrm{pause}]\!](\mathrm{K0}) = [\![\mathrm{pause}]\!](\mathrm{SEL}) \wedge [\![\mathrm{pause}]\!](\mathrm{RES})$, so as $[\![\mathbf{p}]\!](\mathrm{SEL}) \simeq 0$, $[\![\mathrm{pause}]\!](\mathrm{K0}) \simeq 0$.

CASE 5:$\mathbf{p}^{\mathrm{p}}$=(signal **S** $\mathbf{p}^{\mathrm{p}}{}_i$)

1. By the definition of Can, $\mathit{Can}^{\mathrm{K}}(\text{(signal } \mathbf{S}\ \mathbf{p}^{\mathrm{p}}{}_i), \theta) = \mathit{Can}^{\mathrm{K}}(\mathbf{p}^{\mathrm{p}}{}_i, \theta)$.

2. By the definition of $[\![\cdot]\!]$, $[\![(\text{signal } \mathbf{S}\ \mathbf{p}^{\mathrm{p}}{}_i)]\!](\mathrm{SEL}) = [\![\mathbf{p}^{\mathrm{p}}{}_i]\!](\mathrm{SEL})$.

3. By (1) and (2), this case follows by induction.

CASE 6:$\mathbf{p}^{\mathrm{P}}=(\mathrm{seq}\ \mathbf{p}^{\mathrm{P}}_{i}\ \mathbf{q}^{\mathrm{P}}_{i})$

1. By the definition of $[\![\cdot]\!]$ and the premise that $[\![\mathbf{p}^{\mathrm{P}}]\!](\mathrm{SEL}) \simeq 0$, we can conclude that $[\![\mathbf{p}^{\mathrm{P}}_{i}]\!](\mathrm{SEL}) \simeq 0$ and $[\![\mathbf{q}^{\mathrm{P}}_{i}]\!](\mathrm{SEL}) \simeq 0$.

2. By the definition of $[\![\cdot]\!]$, $[\![\mathbf{p}^{\mathrm{P}}]\!](\mathrm{K0}) \simeq [\![\mathbf{q}^{\mathrm{P}}_{i}]\!](\mathrm{K0})$

3. By the definition of $[\![\cdot]\!]$, for all $\boldsymbol{\kappa} > 0$ $[\![\mathbf{p}^{\mathrm{P}}]\!](\mathrm{K}\kappa) \simeq [\![\mathbf{p}^{\mathrm{P}}_{i}]\!](\mathrm{K}\kappa) \vee [\![\mathbf{q}^{\mathrm{P}}_{i}]\!](\mathrm{K}\kappa)$

4. Cases of $0 \in \mathit{Can}^{\mathrm{K}}(\mathbf{p}^{\mathrm{P}}_{i}, \theta)$:

CASE 6.I:$0 \notin \mathit{Can}^{\mathrm{K}}(\mathbf{p}^{\mathrm{P}}_{i}, \theta)$

4.1. By the definition of Can, we know that $\mathit{Can}(\mathbf{p}^{\mathrm{P}}, \theta) = \mathit{Can}(\mathbf{p}^{\mathrm{P}}_{i}, \theta)$

4.2. By (4.1) we know that $(\mathrm{L} \notin \mathbf{n}\ \mathit{Can}^{\mathrm{K}}(\mathbf{p}^{\mathrm{P}}_{i}, \theta))$

4.3. By (4.2), (1), and induction, we know that $[\![\mathbf{p}^{\mathrm{P}}_{i}]\!](\mathrm{K0}) \simeq 0$

4.4. By (4.3) and the definition of compile we know that $[\![\mathbf{q}^{\mathrm{P}}_{i}]\!](\mathrm{GO}) \simeq 0$

4.5. by (4.4) and (1), and lemma 76 (ACTIVATION CONDITION), we know that all outputs of $\mathbf{q}^{\mathrm{P}}_{i}$ are 0.

4.6. By (4.1) and (1), and induction, we know that $[\![\mathbf{p}^{\mathrm{P}}_{i}]\!](\mathrm{Kn}) \simeq 0$

4.7. By (4.5), (4.6), and both (2) or (3), may conclude that $[\![\mathbf{p}^{\mathrm{P}}]\!](\mathrm{Kn}) \simeq 0$.

CASE 6.II:$0 \in \mathit{Can}^{\mathrm{K}}(\mathbf{p}^{\mathrm{P}}_{i}, \theta)$

4.1. By the definition of Can, we know that $\mathit{Can}^{\mathrm{K}}(\mathbf{p}^{\mathrm{P}}, \theta) = \mathit{Can}^{\mathrm{K}}(\mathbf{p}^{\mathrm{P}}_{i}, \theta) \cup \mathit{Can}^{\mathrm{K}}(\mathbf{q}^{\mathrm{P}}_{i}, \theta)$

4.2. By (4.1) and our premise, we know that $(\mathrm{L} \notin \mathbf{n}\ \mathit{Can}^{\mathrm{K}}(\mathbf{p}^{\mathrm{P}}_{i}, \theta))$ and $(\mathrm{L} \notin \mathbf{n}\ \mathit{Can}^{\mathrm{K}}(\mathbf{q}^{\mathrm{P}}_{i}, \theta))$

4.3. By (4.2) and (1), we may induct on $\mathbf{p}^{\mathrm{p}}{}_i$ and $\mathbf{q}^{\mathrm{p}}{}_i$ to conclude that $[\![\mathbf{p}^{\mathrm{p}}{}_i]\!](\mathrm{Kn}) \simeq 0$ and $[\![\mathbf{q}^{\mathrm{p}}{}_i]\!](\mathrm{Kn}) \simeq 0$

4.4. By (4.3) and (3), we may conclude that $[\![\mathbf{p}^{\mathrm{p}}]\!](\mathrm{Kn}) \simeq 0$.

CASE 7:$\mathbf{p}^{\mathrm{p}}$=(if $\mathbf{S}$ $\mathbf{p}^{\mathrm{p}}{}_i$ $\mathbf{q}^{\mathrm{p}}{}_i$)

1. By the definition of $[\![\cdot]\!]$ and the premise that $[\![\mathbf{p}^{\mathrm{p}}]\!](\mathrm{SEL}) \simeq 0$, we can conclude that $[\![\mathbf{p}^{\mathrm{p}}{}_i]\!](\mathrm{SEL}) \simeq 0$ and $[\![\mathbf{q}^{\mathrm{p}}{}_i]\!](\mathrm{SEL}) \simeq 0$.
2. By the definition of $[\![\cdot]\!]$, for all $\boldsymbol{\kappa}$ $[\![\mathbf{p}^{\mathrm{p}}]\!](\mathrm{K}\kappa) \simeq [\![\mathbf{p}^{\mathrm{p}}{}_i]\!](\mathrm{K}\kappa) \vee [\![\mathbf{q}^{\mathrm{p}}{}_i]\!](\mathrm{K}\kappa)$
3. Cases of $\theta(\mathbf{S})$:

CASE 7.I:$\theta(\mathbf{S})$=1

3.1. By the definition of *Can*, we know that ($\mathrm{L}\not\in$ $\mathbf{n}$ $\mathit{Can}^{\mathrm{K}}(\mathbf{p}^{\mathrm{p}}{}_i, \theta)$).

3.2. By (3.1), (1), and induction we can conclude that $[\![\mathbf{q}^{\mathrm{p}}{}_i]\!](\mathrm{Kn}) \simeq 0$.

3.3. by $[\![\mathbf{p}^{\mathrm{p}}]\!] \setminus \theta$, and the definition of $[\![\cdot]\!]$, we can conclude that $[\![\mathbf{q}^{\mathrm{p}}{}_i]\!](\mathrm{GO}) \simeq 0$.

3.4. By (3.2), (3.3), and (2) we can conclude that $[\![\mathbf{p}^{\mathrm{p}}]\!](\mathrm{Kn}) \simeq 0$.

CASE 7.II:$\theta(\mathbf{S})$=0

This case is analogous to the previous case.

CASE 7.III:$\theta(\mathbf{S})$=$\perp$

This follows directly by induction on both branches.

CASE 8:$\mathbf{p}^{\mathrm{p}}$=(par $\mathbf{p}^{\mathrm{p}}{}_i$ $\mathbf{q}^{\mathrm{p}}{}_i$)

1. By the definition of Can, $Can^{K}(\mathrm{(par}\ \mathbf{p}^{p}_{i}\ \mathbf{q}^{p}_{i}\mathrm{)}, \theta) = \{\ max(\kappa_1, \kappa_2) \mid \kappa_1 \in Can^{K}(\mathbf{p}^{p}_{i}, \theta)\ , \kappa_2 \in Can^{K}(\mathbf{q}^{p}_{i}, \theta)\ \}$

2. By (1) and the premise that $\kappa \notin Can^{K}(\mathbf{p}^{p}, \theta)$, we have three cases:

   - $\kappa \notin Can^{K}(\mathbf{p}^{p}_{i}, \theta)$ and $\kappa \notin Can^{K}(\mathbf{q}^{p}_{i}, \theta)$
   - $\kappa \in Can^{K}(\mathbf{p}^{p}_{i}, \theta)$ but for all $\kappa_2 \in Can^{K}(\mathbf{q}^{p}_{i}, \theta)$, $k_2 > \kappa$
   - $\kappa \in Can^{K}(\mathbf{q}^{p}_{i}, \theta)$ but for all $\kappa_2 \in Can^{K}(\mathbf{p}^{p}_{i}, \theta)$, $k_2 > \kappa$

3. Cases of (2):

CASE 8.I: $\langle \kappa \notin Can^{K}(\mathbf{p}^{p}_{i}, \theta), \kappa \notin Can^{K}(\mathbf{q}^{p}_{i}, \theta) \rangle$

   3.1. It is clear from the definition of the synchronizer that an output wire Kn can only be 1 if at least one of the subcircuits has its Kn equal to 1

   3.2. By induction we may conclude that $[\![\mathbf{p}^{p}_{i}]\!](\mathrm{K}\kappa) \simeq 0$ and $[\![\mathbf{q}^{p}_{i}]\!](\mathrm{K}\kappa) \simeq 0$.

   3.3. By (3.1) and (3.2), we can conclude that $[\![\mathrm{(par}\ \mathbf{p}^{p}_{i}\ \mathbf{q}^{p}_{i}\mathrm{)}]\!](\mathrm{K}\kappa) \simeq 0$

CASE 8.II: $\kappa \in Can^{K}(\mathbf{p}^{p}_{i}, \theta)$

   3.1. As $[\![\mathbf{p}^{p}]\!](\mathrm{SEL}) \simeq 0$, and by the definition of $[\![\cdot]\!]$, we know that both the LEM and REM wires are 0.

   3.2. By the definition of the parallel synchronizer, (3.1) means that a Kn wire can be 1 only if there is a Ln_1 and a Rn_2 wire which are 1, where $\mathbf{n}_1 \leq \mathbf{n}$ and $\mathbf{n}_2 \leq \mathbf{n}$

   3.3. By induction on each $\mathrm{K}\kappa_3$ wire less κ in $\mathbf{q}^{p}_{i}$, which in this clause must not be in the result of Can, all Rn_2 wires less than Kκ must be 0.

   3.4. By (3.3) and (3.2), $[\![\mathrm{(par}\ \mathbf{p}^{p}_{i}\ \mathbf{q}^{p}_{i}\mathrm{)}]\!](\mathrm{K}\kappa) \simeq 0$.

CASE 8.III:$\kappa \in \mathit{Can}^{K}(\mathbf{q}^{p}_{i}, \theta)$

This case is analogous to the previous one.

CASE 9:$\mathbf{p}^{p}$=(suspend $\mathbf{p}^{p}_{i}$ **S**)

This case follows fairly directly by induction.

CASE 10:$\mathbf{p}^{p}$=(trap $\mathbf{p}^{p}_{i}$)

This case follows fairly directly by induction.

CASE 11:$\mathbf{p}^{p}$=(ϱ $\langle \theta^{r}, \mathbf{A} \rangle$. $\mathbf{p}_{i}$)

This case is given by lemma 74 (CAN RHO K IS SOUND).

□

LEMMA 73 (CAN RHO S IS SOUND)**.**

For all $\mathbf{p}^{p}$, θ, **A**, **S**, *if* $\mathbf{S} \notin \mathit{Can}^{S}_{\varrho}(\varrho\ \langle \theta^{r}, \mathbf{A} \rangle.\ \mathbf{p}^{p}), \{\})$ *and* $[\![(\varrho\ \langle \theta^{r}, \mathbf{A} \rangle.\ \mathbf{p}^{p})]\!](\mathrm{SEL}) \simeq 0$ *then* $[\![(\varrho\ \langle \theta^{r}, \mathbf{A} \rangle.\ \mathbf{p}^{p})]\!](\mathbf{S}^{o}) \simeq 0$

INTERPRETATION. This theorem states that Can_{ϱ} accurately predicts when signal output wires will be set to 0.

PROOF.

Can_{ϱ} is essentially repeated applications of the signal case in Can. This holds by the same line of argument there used in that case of lemma 71 (CAN S IS SOUND). □

LEMMA 74 (CAN RHO K IS SOUND)**.**

For any term and environment $\mathbf{p}^{p}$ *and* θ *and* **A**, *and return code* κ *if* $\kappa \notin \mathit{Can}^{K}_{\varrho}(\varrho\ \langle \theta^{r}, \mathbf{A} \rangle.\ \mathbf{p}^{p}), \{\})$,
and $[\![(\varrho\ \langle \theta^{r}, \mathbf{A} \rangle.\ \mathbf{p}^{p})]\!](\mathrm{SEL}) \simeq 0$, *then* $[\![(\varrho\ \langle \theta^{r}, \mathbf{A} \rangle.\ \mathbf{p}^{p})]\!](\mathrm{K}\kappa) \simeq 0$

INTERPRETATION. This theorem states that Can_{ϱ} accurately predicts when control wires will be set to 0.

PROOF.

Can_0 is essentially repeated applications of the signal case in Can. This holds by the same line of argument there used in that case of lemma 72 (CAN K IS SOUND). □

LEMMA 75 (CAN K ON PAUSED IS 1)**.**

For all $\hat{\mathbf{p}}$, θ, $Can^K(\hat{\mathbf{p}}, \theta) = \{1\}$

PROOF.

Induction on $\hat{\mathbf{p}}_O$:

CASE 1:$\hat{\mathbf{p}}_O$=pause

Follows by the definition of Can.

CASE 2:$\hat{\mathbf{p}}_O$=(suspend $\hat{\mathbf{p}}$ **S**)

By the definition of Can, $Can^K((\text{suspend } \mathbf{p}\ \mathbf{S}), \theta) = Can^K(\mathbf{p}, \theta)$, thus this follows by induction.

CASE 3:$\hat{\mathbf{p}}_O$=(seq $\hat{\mathbf{p}}$ **q**)

By induction we know that $Can^K(\hat{\mathbf{p}}, \theta) = \{1\}$. By the definition of Can, this means $Can^K((\text{seq } \hat{\mathbf{p}}\ \mathbf{q}), \theta) = Can^K(\hat{\mathbf{p}}, \theta)$. Therefore, $Can^K((\text{seq } \hat{\mathbf{p}}\ \mathbf{q}), \theta) = \{1\}$.

CASE 4:$\hat{\mathbf{p}}_O$=(par $\hat{\mathbf{p}}_1$ $\hat{\mathbf{p}}_2$)

By induction $Can^K(\hat{\mathbf{p}}_1, \theta) = Can^K(\hat{\mathbf{p}}_2, \theta) = \{1\}$. Thus, by the definition of Can, $Can^K((\text{par } \hat{\mathbf{p}}_1\ \hat{\mathbf{p}}_2), \theta) = \{1\}$.

CASE 5:$\hat{\mathbf{p}}_O$=(trap $\hat{\mathbf{p}}$)

By induction $Can^K(\hat{\mathbf{p}}, \theta) = \{1\}$. By the definition of $\downarrow^\kappa$ and Can,
$Can^K((\text{trap } \hat{\mathbf{p}}), \theta) = \{\downarrow^\kappa x \mid x \in Can^K(\hat{\mathbf{p}}, \theta)\} = \{\downarrow^\kappa x \mid x \in \{1\}\} = \{1\}$.

□

B.6. Circuit Compilation Properties

This section contains proofs and properties about the circuit compilation and how it relates to concepts in the term writing system like term decomposition and free variables.

Lemma 76 (Activation Condition)**.**

for any term $\mathbf{p}^{\mathrm{p}}$*, if* $[\![\mathbf{p}^{\mathrm{p}}]\!](\mathrm{GO}) \vee ([\![\mathbf{p}^{\mathrm{p}}]\!](\mathrm{SEL}) \wedge [\![\mathbf{p}^{\mathrm{p}}]\!](\mathrm{RES})) = 0$ *then all output* Kn *and all signals in the output environment are* 0*.*

Proof.

Induction on $\mathbf{p}^{\mathrm{p}}$:

Case 1:$\mathbf{p}^{\mathrm{p}}$=nothing

The only output is $[\![\mathrm{nothing}]\!](\mathrm{K0}) = [\![\mathrm{nothing}]\!](\mathrm{GO})$, which is 0 by our premises

Case 2:$\mathbf{p}^{\mathrm{p}}$=(exit **n**)

Analogous to the last case.

Case 3:$\mathbf{p}^{\mathrm{p}}$=(emit **S**)

Analogous to the last case.

Case 4:$\mathbf{p}^{\mathrm{p}}$=pause

Example:

```
> (assert-same
   #:constraints '(not (or GO (and --SEL RES)))
   (compile-esterel (term pause))
   (compile-esterel (term nothing)))
```

Thus this follows by the same argument as the nothing case.

Case 5:$\mathbf{p}^{\mathrm{p}}$=(signal **S** $\mathbf{p}^{\mathrm{p}}{}_{i}$)

The compilation here only removes one signal from the interface, therefor this holds by induction.

Case 6:$\mathbf{p}^{\mathrm{p}}$=(par $\mathbf{p}^{\mathrm{p}}{}_{i}$ $\mathbf{q}^{\mathrm{p}}{}_{i}$)

GO and RES are pass in unchanged, and $[\![(\mathrm{par}\ \mathbf{p}^{\mathrm{p}}{}_{i}\ \mathbf{q}^{\mathrm{p}}{}_{i})]\!](\mathrm{SEL}) = [\![\mathbf{p}^{\mathrm{p}}{}_{i}]\!](\mathrm{SEL}) \vee [\![\mathbf{q}^{\mathrm{p}}{}_{i}]\!](\mathrm{SEL})$. Therefore our premises must hold on $\mathbf{p}^{\mathrm{p}}{}_{i}$ and $\mathbf{q}^{\mathrm{p}}{}_{i}$. Thus, induction, the outputs of $[\![\mathbf{p}^{\mathrm{p}}]\!]$ and $[\![\mathbf{q}^{\mathrm{p}}]\!]$ are 0.

The output signals of $[\![(\mathrm{par}\ \mathbf{p}^{\mathrm{p}}{}_{i}\ \mathbf{q}^{\mathrm{p}}{}_{i})]\!]$ are the $\vee$ of the inner branches, thus they must be 0.

The control outputs of the synchronizer requires at least some of its inputs be 1 to give an output. However by our premises and induction they are all 0, thus all Kns are 0.

Case 7:$\mathbf{p}^{\mathrm{p}}$=(seq $\mathbf{p}^{\mathrm{p}}{}_{i}$ $\mathbf{q}^{\mathrm{p}}{}_{i}$)

All inputs are passed to $[\![\mathbf{p}^{\mathrm{p}}{}_{i}]\!]$ unchanged, thus by induction all of its outputs are 0.

The GO of $[\![\mathbf{q}^{\mathrm{p}}{}_{i}]\!]$ is given by $[\![\mathbf{p}^{\mathrm{p}}{}_{i}]\!](\mathrm{K0})$, and the rest of the inputs are broadcast, thus by induction all outputs of $[\![\mathbf{q}^{\mathrm{p}}{}_{i}]\!]$ are 0. Therefore all outputs of the overall circuit are 0.

Case 8:$\mathbf{p}^{\mathrm{p}}$=(trap $\mathbf{p}^{\mathrm{p}}{}_{i}$)

GO and RES are passed to $[\![\mathbf{p}^{\mathrm{p}}{}_{i}]\!]$ unchanged, thus by induction the outputs of $[\![\mathbf{p}^{\mathrm{p}}{}_{i}]\!]$ are 0. Therefore the outputs of the overall circuit are 0.

Case 9:$\mathbf{p}^{\mathrm{p}}$=(suspend $\mathbf{p}^{\mathrm{p}}{}_{i}$ **S**)

GO is passed to $[\![\mathbf{p}^{\mathrm{p}}{}_{i}]\!]$ unchanged. The RES of $[\![\mathbf{p}^{\mathrm{p}}{}_{i}]\!]$ is $\wedge$ed with RES, thus it too must be 0. Thus by induction the outputs of $[\![\mathbf{p}^{\mathrm{p}}{}_{i}]\!]$ are 0. Therefore the outputs of the overall circuit are 0.

Case 10:$\mathbf{p}^{\mathrm{p}}$=(ϱ $\langle \boldsymbol{\theta}^{\mathrm{r}}, \mathrm{WAIT}\rangle$. $\mathbf{p}^{\mathrm{p}}{}_{i}$)

In this case GO and RES are passed unchanged, therefore this follows directly by induction.

CASE 11:$\mathbf{p}^{\mathsf{p}}$=(if $\mathbf{S}$ $\mathbf{p}^{\mathsf{p}}_{\mathit{i}}$ $\mathbf{q}^{\mathsf{p}}_{\mathit{i}}$)

The GO wire of $[\![\mathbf{p}^{\mathsf{p}}_{\mathit{i}}]\!]$ and $[\![\mathbf{q}^{\mathsf{p}}_{\mathit{i}}]\!]$ are $\wedge$ed with the overall GO wire, thus their GO wires must be 0. The RES wire is broadcast, thus it too is 0. Thus by induction the outputs of both branches are 0. The outputs are $\vee$ed, therefore the overall outputs are all 0.

□

LEMMA 77 (SELECTION START)**.**

for any term $\mathbf{p}^{\mathsf{p}}$*, during the first instant* $[\![\mathbf{p}]\!](\mathrm{SEL}) = 0$.

PROOF.

This is easy to see as all registers are initialized to 0, and SEL is the $\vee$ of all registers. □

LEMMA 78 (CONSTRUCTIVE UNLESS ACTIVATED)**.**

for any term $\mathbf{p}^{\mathsf{p}}$*, if* $[\![\mathbf{p}^{\mathsf{p}}]\!](\mathrm{GO}) \vee ([\![\mathbf{p}^{\mathsf{p}}]\!](\mathrm{SEL}) \wedge [\![\mathbf{p}^{\mathsf{p}}]\!](\mathrm{RES})) = 0$ *then* $[\![\mathbf{p}]\!]$ *is constructive for any assignments to its inputs.*

INTERPRETATION. The point of this proof is to show that a circuit from the compilation of a term can only exhibit non-constructive behavior when they are activated, justifying that dead code can be erased without effecting the constructivity of the overall circuit.

PROOF.

The inductive arguments of this proof are the same as for lemma 76 (ACTIVATION CONDITION), thus some details have been elided. Induction on $\mathbf{p}^{\mathsf{p}}$:

CASE 1:$\mathbf{p}^{\mathsf{p}}$=nothing

Example:

```
> (assert-totally-constructive
   #:constraints '(not GO)
   (compile-esterel (term nothing)))
```

Case 2:$\mathbf{p}^{\mathrm{p}}$=(emit **S**)

Example:

```
> (assert-totally-constructive
   #:constraints '(not GO)
   (compile-esterel (term (emit S))))
```

Case 3:$\mathbf{p}^{\mathrm{p}}$=(exit **n**)

Example:

```
> (assert-totally-constructive
   #:constraints '(not GO)
   (compile-esterel (term (exit 0))))
```

Note that the actual number on the exit just a label on the wire, therefore this holds for all **n**.

Case 4:$\mathbf{p}^{\mathrm{p}}$=pause

Example:

```
> (assert-totally-constructive
   #:constraints '(not (or GO (and --SEL RES)))
   (compile-esterel (term pause)))
```

Case 5:$\mathbf{p}^{\mathrm{p}}$=(trap $\mathbf{p}^{\mathrm{p}}{}_{i}$)

This case follows by simple induction.

Case 6:$\mathbf{p}^{\mathrm{p}}$=(suspend $\mathbf{p}^{\mathrm{p}}{}_{i}$ $\mathbf{S}$)

This case follows by simple induction.

Case 7:$\mathbf{p}^{\mathrm{p}}$=(signal $\mathbf{S}$ $\mathbf{p}^{\mathrm{p}}{}_{i}$)

By lemma 76 (Activation Condition), the signal outputs of $[\![\mathbf{p}^{\mathrm{p}}{}_{i}]\!]$ must be 0. Thus the $\mathbf{S}$ wire is 0. By induction By induction $[\![\mathbf{p}^{\mathrm{p}}{}_{i}]\!]$ must be constructive. Thus all wires are not $\bot$.

Case 8:$\mathbf{p}^{\mathrm{p}}$=(par $\mathbf{p}^{\mathrm{p}}{}_{i}$ $\mathbf{q}^{\mathrm{p}}{}_{i}$)

By lemma 76 (Activation Condition), the control outputs of $[\![\mathbf{p}^{\mathrm{p}}{}_{i}]\!]$ and $[\![\mathbf{p}^{\mathrm{p}}{}_{i}]\!]$ must be 0.

By induction $[\![\mathbf{p}^{\mathrm{p}}{}_{i}]\!]$ and $[\![\mathbf{q}^{\mathrm{p}}{}_{i}]\!]$ must be constructive.

As all inputs to the synchronizer are 0, one can trace the execution forward to show that it too must be constructive.

Thus all wires are not $\bot$.

Case 9:$\mathbf{p}^{\mathrm{p}}$=(seq $\mathbf{p}^{\mathrm{p}}{}_{i}$ $\mathbf{q}^{\mathrm{p}}{}_{i}$)

By induction $[\![\mathbf{p}^{\mathrm{p}}{}_{i}]\!]$ must be constructive. By lemma 76 (Activation Condition), all the control outputs $[\![\mathbf{p}^{\mathrm{p}}{}_{i}]\!]$ are 0. Thus we can perform induction to show that $[\![\mathbf{q}^{\mathrm{p}}{}_{i}]\!]$ is constructive. Thus all wires must be constructive.

Case 10:$\mathbf{p}^{\mathrm{p}}$=(if $\mathbf{S}$ $\mathbf{p}^{\mathrm{p}}{}_{i}$ $\mathbf{q}^{\mathrm{p}}{}_{i}$)

This case follows by induction akin to the same clause in lemma 76 (Activation Condition).

Case 11:$\mathbf{p}^{\mathrm{p}}$=(ϱ $\langle \theta^{\mathrm{r}}, \mathrm{WAIT} \rangle$. $\mathbf{p}^{\mathrm{p}}{}_{i}$)

This case follows by induction akin to the same clause in lemma 76 (Activation Condition).

□

LEMMA 79 (S IS MAINTAINED ACROSS E)**.**

For all $\mathbf{p}^{\mathrm{p}}{}_i = \mathbf{E}^{\mathrm{p}}[\mathbf{q}^{\mathrm{p}}{}_i]$, *and* $\mathbf{S}$, *if* $\mathbf{S}^{\mathrm{i}} \in \mathit{inputs}(\llbracket \mathbf{p}^{\mathrm{p}}{}_i \rrbracket)$ *then* $\llbracket \mathbf{q}^{\mathrm{p}}{}_i \rrbracket(\mathbf{S}^{\mathrm{i}}) \simeq \llbracket \mathbf{p}^{\mathrm{p}}{}_i \rrbracket(\mathbf{S}^{\mathrm{i}})$

PROOF.

Induction on $\mathbf{E}^{\mathrm{p}}$:

CASE 1:$\mathbf{E}^{\mathrm{p}}=\bigcirc$

Trivial as $\mathbf{p}^{\mathrm{p}}{}_i = \mathbf{q}^{\mathrm{p}}{}_i$

CASE 2:$\mathbf{E}^{\mathrm{p}}=(\mathrm{trap}\ \mathbf{E}^{\mathrm{p}})$

As trap does not change touch the signal wires, this follows by induction.

CASE 3:$\mathbf{E}^{\mathrm{p}}=(\mathrm{suspend}\ \mathbf{E}^{\mathrm{p}}\ \mathbf{S})$

As suspend does not change touch the signal wires, this follows by induction.

CASE 4:$\mathbf{E}^{\mathrm{p}}=(\mathrm{par}\ \mathbf{E}^{\mathrm{p}}\ \mathbf{q}^{\mathrm{p}}{}_i)$

As par does not change touch the signal wires, this follows by induction.

CASE 5:$\mathbf{E}^{\mathrm{p}}=(\mathrm{par}\ \mathbf{p}^{\mathrm{p}}{}_i\ \mathbf{E}^{\mathrm{p}})$

As par does not change touch the signal wires, this follows by induction.

CASE 6:$\mathbf{E}^{\mathrm{p}}=(\mathrm{seq}\ \mathbf{E}^{\mathrm{p}}\ \mathbf{q}^{\mathrm{p}}{}_i)$

As par does not change touch the signal wires, this follows by induction.

□

LEMMA 80 (GO IS MAINTAINED ACROSS E)**.**

For all $\mathbf{p}^{\mathrm{p}} = \mathbf{E}^{\mathrm{p}}[\mathbf{q}^{\mathrm{p}}]$, $\llbracket \mathbf{q}^{\mathrm{p}} \rrbracket(\mathrm{GO}) \simeq \llbracket \mathbf{p}^{\mathrm{p}} \rrbracket(\mathrm{GO})$

PROOF.

This proof follows the exact same argument as lemma 79 (S IS MAINTAINED ACROSS E). □

LEMMA 81 (SELECTION DEFINITION)**.**

For any term $\mathbf{p}^{\mathrm{p}} = \mathbf{E}[\mathbf{q}^{\mathrm{p}}]$, *There exist some wires such that* $[\![\mathbf{p}^{\mathrm{p}}]\!](\mathrm{SEL}) = [\![\mathbf{q}^{\mathrm{p}}]\!](\mathrm{SEL}) \vee \mathbf{w}_{others} \ldots$

PROOF.

This follows trivially from the definition of $[\![\cdot]\!]$, as SEL is always the $\vee$ of the SEL wires of the inner terms. □

LEMMA 82 (S OUTPUT IRRELEVANT)**.**

For any term $\mathbf{p}^{\mathrm{p}} = \mathbf{E}[\mathbf{q}^{\mathrm{p}}]$, *for any output wire* $\mathbf{S}^{\mathrm{o}}$ *in* $[\![\mathbf{q}^{\mathrm{p}}]\!]$ *there exists no wire* $\mathbf{w}$ *that is not itself an instance of* $\mathbf{S}^{\mathrm{o}}$ *in* $[\![\mathbf{p}^{\mathrm{p}}]\!]$ *which depends on* $\mathbf{S}^{\mathrm{o}}$

INTERPRETATION. The point of this theorem is to show that an $(\mathrm{emit}\ \mathbf{S})$ cannot be ”read” by its context until that emit is closed by a signal or ϱ form.

PROOF.

This follows from the same argument as lemma 81 (SELECTION DEFINITION). □

LEMMA 83 (FREE VARIABLES ARE INPUT/OUTPUTS)**.**

For any $\mathbf{p}^{\mathrm{p}}$ *and* $\mathbf{S}$, $\mathbf{S} \in \mathcal{FV}(\mathbf{p}^{\mathrm{p}})$ *if any only if* $\mathbf{S}_i \in inputs([\![\mathbf{p}^{\mathrm{p}}]\!])$ *or* $\mathbf{S}_o \in outputs([\![\mathbf{p}^{\mathrm{p}}]\!])$

INTERPRETATION. This states that the free variables of a term capture exactly the input and output signal wires. That is then notion “free variable” exactly corresponds to the non-control part of the circuit interface

PROOF.

Note that a signal is free only if it occurs in a (if $\mathbf{S}\ \mathbf{p}^{\mathsf{p}}\ \mathbf{q}^{\mathsf{p}}$) or (emit $\mathbf{S}$) that does not have an outer binder. $[\![(\text{if}\ \mathbf{S}\ \mathbf{p}^{\mathsf{p}}\ \mathbf{q}^{\mathsf{p}})]\!]$ will generate an $\mathbf{S}_i$ wires and (emit $\mathbf{S}$) will generate an $\mathbf{S}_o$ wire. The compilation of all non-binding terms does not change the set of input or output signals. The compilation of (signal $\mathbf{S}\ \mathbf{p}^{\mathsf{p}}$) and ($\varrho\ \langle \theta, \mathbf{A} \rangle.\ \mathbf{p}^{\mathsf{p}}$) remove the $\mathbf{S}_i$ and $\mathbf{S}_o$ wires from the input/output sets for the signals they bind. Thus the input/output sets for signals exactly match the notion of free variables. □

APPENDIX C

The circuit sovler, Circuitous

This appendix is meant to serve as an explanation of the core of the circuit solving library Circuitous. Specifically it describes the interpreter implementation at commit `a3ba4cc` of https://github.com/florence/circuitous/, which is the version used while building this document. Note that the explanation of this section assumes familiarity with Malik (1994), Shiple et al. (1996), and Racket (Flatt and PLT 2010), Rosette (Torlak and Bodik 2013). I would strongly recommend that any reader familiarize themselves with the the above papers, the Rosette guide[1]. the (current incomplete) documentation for Circuitious[2], and the Circuitious test cases[3], as these are not covered in enough detail in this dissertation for this section to make sense.

The purpose of this appendix is help make this work more reproducible and to help increase the readers confidence that the circuit solver is actually correct. As such it focus on the small (approximately 600 lines) Rosette kernel which directly interprets and solves circuits and its tests. The library has contains more: a Racket front-end which compiles to the Rosette model, and helper procedure for manipulating circuits. However those are not described here.

C.1. Internal representation of circuits

Internally, a circuit is represented as an association list, mapping variable names to Boolean expressions, also represented as a list. However this definition is a little misleading, as there are actually two possible ways the circuit library does this. First, there is a representation that directly implements the representation in Malik (1994), where variable names are either pairs of either the symbol `'+` or `'-`, and an arbitrary symbol. Every wire in the circuit has a `+` and a `-` form, corresponding to Malik's `.1` and `.0` forms, respectively. I will refer to these as the positive and negative variants of the wire. Correspondingly, values for each of these variables take on the form of `#t` or `#f`, matching 1 and

[1] https://docs.racket-lang.org/rosette-guide/index.html?q=rosette
[2] https://github.com/florence/circuitous/tree/master/circuitous-doc
[3] https://github.com/florence/circuitous/tree/master/circuitous-test/circuitous/tests

0 in Malik's formulation. I will refer to this as the `pos-neg` representation. Circuits are also represented by using the values #t, #f, and $\perp$ directly. In this case variables are just symbols, and the interpretations of $\wedge$ and $\vee$ are lifted to operate on $\perp$. I will refer to this as the `three-valued` representation.

To handle this the implementation of the interpreter is parameterized by the two different representations. This is accomplished using Racket's `unit` system (Flatt and Felleisen 1998), which is a first-class module system that supports recursive linking. It is akin to ML's Functors, in that modules are parameterized by other modules. As such, a unit which implements one of these representations follows the following signature in figure 47. In the comments which describe the contracts on what the modules provide, `symbolic-boolean` refers to a Rosette symbolic value which evaluates to a boolean, `state` refers to the library's representation of $\theta^{\mathfrak{c}}$, `value` refers to whichever set of values that representation chooses, and `circuit` refers to whichever for of association lists the representation uses.

The functions behave as follows: `symbolic-variable` creates a rosette symbolic variable for a variable in the circuit; `initial-value` gives the value that all variables in the initial `state` should be (`#f` for `pos-neg`, and $\perp$ for `three-valued`); `f-or`, `f-and` `f-not`, and `f-implies` are used by the interpreter to lift expression in the circuit into racket functions that operate over the current state, so that each variable is associated with a single function that computes it's current value; `constraints` generates global constraints for the representation (such as the `+` and `-` variables being mutually exclusive in the `pos-neg` representation); `contructive?` gives a symbolic expression which will be true if and only if the circuit's state is constructive (that is does not contain the representation of $\perp$); `initialize-to-false` and `initialize-to-true` create a state for a (sub)circuit where all values are the representations of the 0 or 1, respectively; `get-maximal-statespace` computes the totally number of instants that may be needed to explore all possible register states given the number of registers in the circuit, a la Shiple et al. (1996); `interp-bound` computes the maximum number of iterations that may be needed to evaluate a circuit in a single instant; `outputs=?` is an equality predicate over states; `constructive-constraints` computes an expression which will evaluate to true if and only if the circuit is constructive on the given inputs.

```
#lang racket/signature
#;(-> variable symbolic-value)
symbolic-boolean

#;Value
initial-value

#;(-> (-> state value) (-> state value) (-> state value))
f-or

#;(-> (-> state value) (-> state value) (-> state value))
f-and

#;(-> (-> state value) (-> state value))
f-not

#;(-> (-> state value) (-> state value) (-> state value))
f-implies

#;(-> state symbolic-boolean)
constraints

#;(-> circuit (-> state boolean))
constructive?

#;(-> circuit state)
initialize-to-false

#;(-> circuit state)
initialize-to-true

#;(-> natural natural)
get-maximal-statespace

#;(-> circuit natural)
interp-bound

#;(-> state state boolean)
outputs=?

#;(-> circuit expression)
constructive-constraints
```

Figure 47: sem-sig.rkt

C.2. The circuit solver

The actual implementations of the representations are recursively linked with the interpreter, therefore I will describe the interpreter interface and implementation before describing the representations' implementations. The signature for the interpreter unit is given in figure 48.

The primary function of interest is `verify-same`, which is the core implementation of the solver. It takes in two circuits, a list of which wires in them correspond to the inputs and outputs of registers, and constraints the caller wishes to impose, and a list of outputs to observe. If the list of outputs is `#f`, all wires are considered outputs. If the registers are set to `#f`, no registers are taken and a slightly different code path is taken that bypasses the multi-instant solver and jumps straight to the single instant solver. This is only used for testing and debugging, therefore I will not describe that code path. This function assumes the circuits have been consistently renamed such that any two wires in the two circuits that have the same name are part of that circuits interface, and therefore may be compared when comparing the circuit's for equality. This renaming is handled at a higher level in the library which is not discussed here. The full implementation of `verify-same` is given in figure 49. The implementation logs debug information then decides if it should take the debug code path or not. As we are ignoring the debug path, the next function of interest is `verify-same/multi`.

The function `verify-same/multi`, shown in figure 50, constructs all the inputs the solver needs, then gives those to the solver via the `do-verify` macro. The inputs to `do-verify` are as follows: `#:=?` is the equality procedure used to compare the outputs for equality. In this case it is `result=?/multi` which loops over the result from each input and uses the `outputs=?` and `constructive?` from the representation unit to make sure each instant behaved the same (see figure 54 and figure 55). The `#:expr1` and `#:expr2` arguments are the two symbolic expressions to execute. The `#:given-constraints` arguments are any constraints given to the solver by the caller (i.e. that GO implies ¬SEL). The `#:gened-constraints` are constraints necessitated by the representation (as specified by `constraints`). And finally `#:outputs` specifies which output wires to observe.

The two expressions are generated by the function `eval/multi*`, which builds the racket representation of a circuit and evaluates it. The first argument is a list of the inputs for each instant, which in this case is a list who's length is long

```
#lang racket/signature

#;(-> circuit circuit
      #:register-pairs1 [(or #f (listof variable variable))]
      #:register-pairs2 [(or #f (listof variable variable))]
      #:constraints [expression]
      #:outputs [(or #f (listof symbol))]
      (or unsat? (list model? state)))
verify-same

#;(-> circuit #:exclude [(listof variable)] (listof (list variable symbolic-variable)))
symbolic-inputs

#;(-> circuit (listof (list variable symbolic-variable)) state)
build-state

#;(-> circuit (listof (list variable (-> state booolean))))
build-formula

#;(-> expression (-> state booolean))
build-expression

#;(-> state circuit-as-functions state)
eval

#;(-> (listof state) circuit-as-functions (listof (list variable variable)) (listof state))
eval/multi

#;(-> (listof state) circuit (listof (list variable variable)) (listof state))
eval/multi*

#;(-> state state #:outputs [(listof variable)] boolean?)
result=?

#;(-> (listof state) (listof state) #:outputs [(listof variable)] boolean?)
result=?/multi

totally-constructive?
```

Figure 48: interp-sig.rkt

enough to explore all register states (computed by `get-maximal-statespace`), and who's values are all symbolic (generated by `symbolic-inputs`). The two sets of constraints are build for every instant by either translating the given constraints into racket function via `build-expression` and evaluating it on all symbolic instants, or by calling

```
(define (verify-same P1 P2
                     #:register-pairs1 [register-pairs1 #f]
                     #:register-pairs2 [register-pairs2 #f]
                     #:constraints [extra-constraints 'true]
                     #:outputs [outputs #f])
    (log-circuit-solver-debug
     "P1: ~a" (pretty-format P1))
    (log-circuit-solver-debug
     "P2: ~a" (pretty-format P2))
    (cond
      [(and register-pairs1 register-pairs2)
       (verify-same/multi P1 P2
                          #:register-pairs1 register-pairs1
                          #:register-pairs2 register-pairs2
                          #:constraints extra-constraints
                          #:outputs outputs)]
      [(not (or register-pairs1 register-pairs2))
       (verify-same/single P1 P2
                           #:constraints extra-constraints
                           #:outputs outputs)]
      [else (error "missing register pair set")]))
```

Figure 49: `verify-same`

the representations `constraints` function on the output of each instant of execution (which will exclude any result states that violate the constraints).

Before describing the symbolic interpreter, I will first discuss describe the solver. The solver macro `do-verify`, found in figure 51, just surrounds the entire computation in a `with-asserts*`, which captures any rosette asserts and resets the assertion store after the solver completes. The real implementation is in `verify/f`, found in figure 52.

The function `verify/f` begins with a large chunk of debug logging statements. It then constructs the symbolic expression `eq`, which is true if and only if the equality predicate `=?` returns `#t`.[4] This symbolic expression is then `verify`ed by Rosette under the assumptions that all of our constraints return true. If a `sat`isfying result is returned it means the verification failed.[5] In this case more debugging information is logged, and a result is returned containing

[4]The equality check for `#t` is needed because racket treats any non-`#f` value as `#t`. If any of these procedures returns a non-Boolean value, such as `'⊥` in the case of the `three-valued` representation, we want to treat that as not true.

[5]SMT solvers, and therefore Rosette, represent verification problems by making sure that the it's negation, when expressed as an existential, does not have a satisfying solution. That is the verification of $\forall$ `x, P(x)` is proven by showing that $\exists$ `x,` $\neg$`P(x)` is unsatisfiable.

```
(define (verify-same/multi P1 P2
                           #:register-pairs1 [register-pairs1 (list)]
                           #:register-pairs2 [register-pairs2 (list)]
                           #:constraints [extra-constraints 'true]
                           #:outputs [outputs #f])
    (do-verify
     #:=? result=?/multi
     #:expr1 e1 #:expr2 e2
     #:given-constraints extra
     #:gened-constraints const
     #:outputs outputs
     (log-circuit-solver-debug "starting multi run for\n~a\nand\n~a"
                               (pretty-format P1)
                               (pretty-format P2))
     (define register-ins1 (map first register-pairs1))
     (define register-outs1 (map second register-pairs1))
     (define register-ins2 (map first register-pairs2))
     (define register-outs2 (map second register-pairs2))
     (define maximal-statespace
       (max (get-maximal-statespace (length register-pairs1))
            (get-maximal-statespace (length register-pairs2))))
     (log-circuit-solver-debug "maximal-statespace: ~a" maximal-statespace)
     (define inputs
       (let loop ([x maximal-statespace])
         (cond [(zero? x) (list)]
               [else
                (cons
                 (symbolic-inputs (append P1 P2)
                                  #:exclude (append register-outs1 register-outs2))
                 (loop (sub1 x)))])))
     (log-circuit-solver-debug "inputs: ~a" (pretty-format inputs))
     (define e1 (eval/multi* inputs P1 register-pairs1))
     (define e2 (eval/multi* inputs P2 register-pairs2))
     (define (make-extra e)
       (andmap (lambda (v) (equal? #t ((build-expression extra-constraints) v))) e))
     (define (make-c e)
       (andmap (lambda (v) (equal? #t (constraints v))) e))
     (define (build-extra ea eb)
       (map (lambda (x)
              (append x
                      (initialize-to-false
                       (map first (first eb)))))
            ea))
     (define extra
       (and (make-extra (build-extra e1 e2))
            (make-extra (build-extra e2 e1))))
     (define const (and (make-c e1) (make-c e2)))))
```

Figure 50: verify-same/multi

```
(define-syntax do-verify
    (syntax-parser
      [(_ #:=? =?:id
          #:expr1 e1:id
          #:expr2 e2:id
          #:given-constraints given-constraints:id
          #:gened-constraints gened-constraints:id
          #:outputs outputs:id
          body:expr ...)
       #'(with-asserts*
          body ...
          (verify/f =? e1 e2 given-constraints gened-constraints outputs))]))
```

Figure 51: do-verify

the satisfying core, and the result of evaluating the two symbolic expressions under that core. Otherwise the unsat core is returned, representing the success of the verification.

Before moving on to the interpreter, the last three solver functions to explain is symbolic-inputs, result=?/multi, and result=?. The first, in figure 53, generates a symbolic variable for every input the in the circuit, giving an association list from the variables to their symbolic representations. This will form the core of the construction of the initial state. The implementation is in figure 53 which does not bare much further discussion.

The two result functions use the outputs=? and constructive? functions from the representation to check if two states, or list of states in the case of result=?/multi, are the same. The result=?/multi function (figure 54) demands that the two lists be the same length, and that every state is result=?. The result=? (figure 55) demands that two states have the same outputs (as determined by outputs=? and outputs) and have the same constructivity, as determined by constructive?.

The last part of the solver totally-constructive? (figure 56). This function is a little bit of a hack, as it just checks if the given circuit is equal to the empty circuit while checking no inputs, which devolves to checking that both circuits have the same constructivity when their inputs are constructive. Therefore the constraints for the call to verify-same/multi contain the constraints which force all free variables in the circuit (the inputs) to be constructive.

```
(define (verify/f =? e1 e2 given-constraints gened-constraints outputs)
    (log-circuit-solver-debug "e1: ~a" (pretty-format e1))
    (log-circuit-solver-debug "e1 vars: ~a" (pretty-format (symbolics e1)))
    (log-circuit-solver-debug "e2: ~a" (pretty-format e2))
    (log-circuit-solver-debug "e2 vars: ~a" (pretty-format (symbolics e2)))
    (log-circuit-solver-debug "constraints: ~a"
                              (pretty-format (equal? #t given-constraints)))
    (log-circuit-solver-debug "generated constraints: ~a"
                              (pretty-format (equal? #t gened-constraints)))
    (log-circuit-solver-debug "asserts: ~a" (pretty-format (asserts)))
    (log-circuit-solver-debug "outputs: ~a" (pretty-format outputs))

    (define eq (equal? #t (=? e1 e2 #:outputs outputs)))

    (log-circuit-solver-debug "eq: ~a" (pretty-format eq))
    (log-circuit-solver-debug "eq symbolics: ~a" (pretty-format (symbolics eq)))
    (define r
      (verify
       #:assume (assert (and (equal? #t given-constraints)
                             (equal? #t gened-constraints)))
       #:guarantee (assert eq)))
    (when (sat? r)
      (log-circuit-solver-debug
       "symbolics in result: ~a"
       (pretty-format
        (map
         (lambda (x) (list x (r x)))
         (symbolics eq)))))
    (if (unsat? r)
        r
        (let ([r (complete-solution r (symbolics eq))])
          (list r (evaluate e1 r) (evaluate e2 r)))))
```

Figure 52: verify/f

```
(define (symbolic-inputs P #:exclude [exclude (list)])
    (filter-map
     (lambda (x)
       (and (not (member x exclude))
            (list x (symbolic-boolean x))))
     (FV P)))
```

Figure 53: symbolic-inputs

```
(define (result=?/multi a b #:outputs [outputs #f])
    (and
     (equal? (length a) (length b))
     (let andmap ([a a]
                  [b b])
       (if (empty? a)
           #t
           (and (result=? (first a) (first b) #:outputs outputs)
                (andmap (rest a) (rest b)))))))
```

Figure 54: `result=?/multi`

```
(define (result=? a b #:outputs [outputs #f])
    (and
     (outputs=? a b #:outputs outputs)
     (equal? (constructive? a)
             (constructive? b))))
```

Figure 55: `result=?`

```
(define (totally-constructive? p
                               #:register-pairs [rp (list)]
                               #:constraints [c 'true])
    (define r
      (verify-same/multi p (list)
                         #:register-pairs1 rp
                         #:register-pairs2 (list)
                         #:outputs (list)
                         #:constraints
                         `(and ,(constructive-constraints
                                 (initialize-to-false (FV p)))
                               ,c)))
    (if (unsat? r)
        r
        (take r 2)))
```

Figure 56: `totally-constructive?`

C.3. The circuit interpreter

The top level function of the circuit interpreter is `eval/multi*` (figure 57), which is responsible for constructing the interpreters representation of the circuit. It's first argument is the list of inputs for each instant, it's second is the circuit

```
(define (eval/multi* IVS eqs register-pairs)
    (define mid (build-state eqs (list)))
    (eval/multi (map (lambda (x) (append x mid)) IVS)
                (build-formula eqs)
                (map first register-pairs)
                (initialize-to-false
                 (map second register-pairs))))
```

Figure 57: `eval/multi*`

equations, and the last are the input/output pairs of wire names for the registers.[6] The `eval/multi*` function delegates to the `eval/multi` function, which operates on the internal circuit representation. It's arguments are the starting state for each instant, which is the inputs appended to an initial wire state computed by `build-state`; the internal circuit representation list which maps each wire name to a function of the current state to a value) is constructed by `build-formula`; the list of wires which are inputs to registers, and a substate that corresponds to the initial state of the outputs of the registers, which must be in the same order as the inputs. This initial state has all registers set to false.

The `eval/multi` function (figure 58) uses the single instant evaluator `eval` to evaluate each instant, threading the inputs of each register to the outputs in subsequent instants. It will short-circuit if any instant is not constructive, as the future behavior of such a circuit is unspecified (that is, non-constructive is an error state). This function is a recursive loop which keeps track of the current register states `out-registers`, a backwards list of the outputs of each instant `seen`, and the remaining list of input states to execute `states`. If the states are empty is return the `seen` states, which will be reversed. Otherwise it adds the current register state to the next input state and `eval`uates that instant. If the result is not constructive the result is added to the seen list and the loop is aborted. Otherwise the input values of the registers are copied to the outputs, and the loop restarts.

The `eval` (figure 59) fully evaluates a single instant. Following the procedure laid out by Malik (1994), it uses the `step` function to evaluate all the gates it can. This process must settle in `interp-bound` steps, as the values of each gate are monotonic. Note that this loop does not exit early if a fixed pointed is reached. While this is an optimization in the concrete case, it is actually a pessimization when solving. This is because, as the state is symbolic, the check that a fixed point is reached adds extra, recursive, formula to the solver.

[6]Note that the circuit interpreter works on both concrete and symbolic inputs. This is the joy of using Rosette.

```
(define (eval/multi states formulas in-registers out-registers)
    (log-circuit-eval-debug "starting eval/multi")
    (reverse
     (let loop ([registers out-registers]
                [seen (list)]
                [states states])
       (log-circuit-eval-debug "states: ~a" (pretty-format states))
       (log-circuit-eval-debug "seen: ~a" (pretty-format seen))
       (log-circuit-eval-debug "registers: ~a" out-registers)
       (cond
         [(empty? states) seen]
         [else
          (define next (eval (append (first states) registers)
                             formulas))
          (log-circuit-eval-debug "next: ~a" (pretty-format next))
          (if (not (constructive? next))
              (cons next seen)
              (loop (map (lambda (in outpair)
                           (list (first outpair)
                                 (deref next in)))
                         in-registers
                         out-registers)
                    (cons next seen)
                    (rest states)))])))))
```

Figure 58: `eval/multi`

```
(define (eval state formulas)
    (define (eval* state formulas bound)
      (if (zero? bound)
          state
          (let ([x (step state formulas)])
            (eval* x formulas (sub1 bound)))))
    (eval* state formulas (interp-bound formulas)))
```

Figure 59: `eval`

Next, the `step` function (figure 60) takes in the current state and list of formula and evaluates each formula exactly once on the current state to generate a new state. It does this by iterating over the current state and, for each value, if it is defined by a wire (rather that being an input), the function for that wire is evaluated on the current state. Note that,

```
(define (step state formulas)
    (map
     (lambda (w)
       (define name (first w))
       (define f
         (and (contains? formulas name)
              (deref formulas name)))
       (if f
           (list name (f state))
           w))
     state))
```

Figure 60: `step`

```
(define (build-state P inputs)
    (append
     (map
      (lambda (w) (list (first w) initial-value))
      P)
     inputs))
```

Figure 61: `build-state`

again, wires which are don't represent `'⊥` don't need to be evaluated, but this would be a pessimization when solving as it added more formula.[7]

The next interesting function in the interpreter is `build-state` (figure 61). It builds a state from the circuit and it's inputs, which in the symbolic case will have been generated by `symbolic-inputs`. It simply sets all wires to their initial value, and adds in the inputs.[8]

The final interesting functions in the interpreter are `build-formula` (figure 62) and `build-expression` (figure 63), which convert a the circuit AST into Racket functions. The recursively walk the AST and invoke the `f-and`, `f-or`, and `f-not` functions into "compile" the expressions. Constants and variables are handled manually, rather than by the representations.

[7]Note that evaluating the wire again also will add more formula, but these formula will be added in either case as rosette must explore both branches of the conditional.

[8]The inputs argument is unused in the multi-instant case, as `eval/multi` handles that itself.

```
(define (build-formula P)
    (map
     (lambda (x)
       (match-define (list n '= e) x)
       (list n (build-expression e)))
     P))
```

Figure 62: build-formula

```
(define (build-expression e)
    (match e
      [`(and ,e1 ,e2)
       (f-and (build-expression e1)
              (build-expression e2))]
      [`(or ,e1 ,e2)
       (f-or (build-expression e1)
             (build-expression e2))]
      [`(not ,e1)
       (f-not (build-expression e1))]
      [`(implies ,e1 ,e2)
       (f-implies (build-expression e1)
                  (build-expression e2))]
      [(or #t `true) (lambda (_) #t)]
      [(or #f `false) (lambda (_) #f)]
      [`⊥ (lambda (_) '⊥)]
      [x
       (lambda (w) (deref w x))]))
```

Figure 63: build-expression

C.4. Implementing the representations

Finally on too the representation. First up is the easier of the two to understand, the `three-valued` representation.

C.4.1. The three-value representation

As the interface to the representations have already been explained, this explanation will be brief. The interp-bound (figure 64) is the number of wires in the circuit, as in each cycle either a fixed point has been reached or one gate's

```
(define (interp-bound formula)
    (length formula))
```

Figure 64: `interp-bound` in the `three-valued` representation

```
(define initial-value '⊥)
```

Figure 65: `initial-value` in the `three-valued` representation

```
(define (get-maximal-statespace x)
    (expt 2 x))
```

Figure 66: `get-maximal-statespace` in the `three-valued` representation

```
(define (f-and x y)
    (lambda (w)
      (define a (x w))
      (define b (y w))
      (case a
        [(#f) #f]
        [(#t) b]
        [(⊥)
         (case b
           [(#f) #f]
           [(#t ⊥) '⊥]
           [else (error 'and "second argument is not an extended boolean: ~a" b)])]
        [else
         (error 'and "first argument is not an extended boolean: ~a" a)])))
```

Figure 67: `f-and` in the `three-valued` representation

value will change. The `initial-value` (figure 65) is `'⊥`. The number of instants needed for `x` registers (figure 66) is 2^x, as each register can only take on the values `#t` or `#f`.[9]

The "compiling" functions `f-and` (figure 67), `f-or` (figure 68), and `f-not` (figure 69) directly implement the truth-tables found in section 2.3.1. They contain extra error cases, which should never be triggerable.

The representation gets more interesting with `symbolic-boolean` (figure 70). Rosette implements a union of three values like `#t`, `#f`, and `'⊥` as a symbolic computation that returns one of these values. Therefore two symbolic

[9]The value `'⊥` is forbidden, because that would mean the previous instant was non-constructive.

```
(define (f-or x y)
    (lambda (w)
      (define a (x w))
      (define b (y w))
      (case a
        [(#t) #t]
        [(#f) b]
        [(⊥)
         (case b
           [(#t) #t]
           [(#f ⊥) '⊥]
           [else (error 'or "second argument is not an extended boolean: ~a" b)])]
        [else
         (error 'or "first argument is not an extended boolean: ~a" a)])))
```

Figure 68: `f-or` in the `three-valued` representation

```
(define (f-not a)
    (lambda (w)
      (case (a w)
        [(#t) #f]
        [(#f) #t]
        [(⊥) '⊥]
        [else (error 'not "argument is not an extended boolean: ~a" (a w))])))
```

Figure 69: `f-not` in the `three-valued` representation

Booleans—`pos` and `neg`—are created[10]. If `pos` is `#t`, then the overall symbolic Boolean is `#t`. Otherwise if `neg` is `#t`, the overall symbolic Boolean is `#f`. If both are `#f`, the overall symbolic Boolean is `'⊥`. The case where both are `#t` is excluded by assertion. Not that this means that, at the lowest level, the `three-valued` and `pos-neg` representation actually use the same representation of values. The only difference is in the representation of wires: In the `three-valued` case the two Booleans are bundled into one equation, whereas in the `pos-neg` representation they will have separate equations.

This representation has no external `constraints` (figure 71), and is `constructive?` (figure 72 and figure 73) if all wires are `#t` or `#f`.

[10]In order to create them dynamically I use Rosette's reflective API, and ensure the variables are unique using a small amount of state in `next-unique!`.

```
(define (symbolic-boolean name)
    (define pos
      (constant (~a "pos-" name "$" (next-unique! name)) boolean?))
    (define neg
      (constant (~a "neg-" name "$" (next-unique! name)) boolean?))
    (assert (not (and pos neg)))
    (if pos #t (if neg #f '⊥)))
```

Figure 70: `symbolic-boolean` in the `three-valued` representation

```
(define (constraints _)
    #t)
```

Figure 71: `constraints` in the `three-valued` representation

```
(define (constructive? a)
    (equal?
     ((build-expression (constructive-constraints a)) a)
     #t))
```

Figure 72: `constructive?` in the `three-valued` representation

```
(define (constructive-constraints inputs)
    (if (empty? inputs)
        'true
        `(and (or ,(first (first inputs)) (not ,(first (first inputs))))
              ,(constructive-constraints (rest inputs)))))
```

Figure 73: `constructive-constraints` in the `three-valued` representation

Two outputs are `outputs=?` (figure 74) when either all of the specified output wires are the same, or—if no output wire set is given—every wire which is in both circuits has the same value. Note when an output set is given, if a circuit does not have that wire we treat it's value as `#f`.

C.4.2. The pos-neg representation

The `pos-neg` representation is a little more subtle. There are two formula for each wire, but the two formula are mutually exclusive, therefore the `interp-bound` (figure 75) is half of the size of overall number of formula. The

```
(define (outputs=? a b #:outputs [outputs #f])
    (if outputs
        (andmap
         (lambda (w)
           (equal?
            (and (contains? a w) (deref a w))
            (and (contains? b w) (deref b w))))
         outputs)
        (andmap
         (lambda (w)
           (implies
            (contains? b (first w))
            (equal? (second w) (deref b (first w)))))
         a)))
```

Figure 74: `outputs=?` in the `three-valued` representation

```
(define (interp-bound formula)
    (/ (length formula) 2))
```

Figure 75: `interp-bound` in the `pos-neg` representation

```
(define (get-maximal-statespace x)
    (expt 2 (inexact->exact (ceiling (/ x 2)))))
```

Figure 76: `get-maximal-statespace` in the `pos-neg` representation

```
(define (constructive? P)
    ((build-expression (constructive-constraints P)) P))
```

Figure 77: `constructive?` in the `pos-neg` representation

`get-maximal-statespace` (figure 76) function also halves it's inputs to account for the exclusivity of the positive an negative representations, giving $2^{x/2}$.

Constructivity checking(figure 77 and figure 78) also changes. When the positive variant of each wire is found , an expression is added to the constraints insisting that either the positive or negative variant be true. To avoid adding redundant expressions, the negative variant is skipped.

```
(define (constructive-constraints inputs)
    (let fold ([current inputs])
      (if
       (empty? current)
       'true
       (let ([n (first (first current))])
         (cond
           [(eq? '+ (first n))
            `(and
              (or ,n (- ,(second n)))
              ,(fold (rest current)))]
           [else (fold (rest current))])))))
```

Figure 78: `constructive-constraints` in the `pos-neg` representation

```
(define initial-value #f)
```

Figure 79: `initial-value` in the `pos-neg` representation

```
(define (symbolic-boolean name)
    (constant (string-replace
               (~a name "$" (next-unique! name))
               " "
               "_")
              boolean?))
```

Figure 80: `symbolic-boolean` in the `pos-neg` representation

The `initial-value` is `#f` (figure 79) as setting both parts of the representation of the wire to `#f` sets the overall representation to `#f`. Symbolic Booleans just represented by a single symbolic variable (figure 80).

However this leads to a non-trival implementation of the global `constraints` (figure 81). It ensures that for every wire with a positive and negative part, only one of them may be true in the given state. Therefore for each positive variant we ensure that at most one of the positive and negative variants are true.

The equality predicate `outputs=?` (figure 82) has a more subtle accounting of the positive and negative variants. Like in the `three-valued` case we treat a missing wire as false. However this means that we treat the positive and negative variants asymmetrically: if we expect the overall wire to be false we must treat the negative variant as true. Therefore

```
(define (constraints I)
    (andmap
     (λ (x)
       (implies
        (and (list? x)
             (list? (first x))
             (eq? (first (first x)) '+)
             (contains? I `(- ,(second (first x)))))
        (not (and (second x)
                  (deref I `(- ,(second (first x))))))))
     I))
```

Figure 81: `constraints` in the `pos-neg` representation

```
(define (outputs=? a b #:outputs [outputs #f])
    (if outputs
        (andmap
         (lambda (w)
           (cond
             [(and (list? w) (equal? (first w) '-))
              (equal?
               (or (not (contains? a w)) (deref a w))
               (or (not (contains? b w)) (deref b w)))]
             [else
              (equal?
               (and (contains? a w) (deref a w))
               (and (contains? b w) (deref b w)))]))
         outputs)
        (andmap
         (lambda (w)
           (implies
            (contains? b (first w))
            (equal? (second w) (deref b (first w)))))
         a)))
```

Figure 82: `outputs=?` in the `pos-neg` representation

`outputs=?` checks which variant it is currently handling and adjusts accordingly. This is only needed in the case where outputs are given, as in the other case we only check wires that we know are in both circuits.

C.5. Trusting the solver

Of course explaining the code should not be enough to convince anyone that the code is enough. Thus I offer the following evidence: test cases and code coverage. The test cases consist of approximate 116 manual test cases targeting specific behavior. In addition there are 1000 random tests which run against compare behavior of the implementation to a redex model of the circuit semantics presented in section 2.3.4.

In addition code coverage of the implementation shows 96% code coverage, where the 4% correspond to error cases that should be unreachable, or are in areas of the codebase used for circuit manipulation and not used in my proofs.

APPENDIX D

Proving equalities through the calculus

The proofs in this section are written in a DSL which checks them against the equations of the calculus, then generates the prose in that section.

THEOREM 84 (CAN SWAP ADJACENT SIGNALS)**.**

For all $\mathbf{p}$, $\mathbf{S}_1$, $\mathbf{S}_2$, $(\text{signal } \mathbf{S}_1\ (\text{signal } \mathbf{S}_2\ \mathbf{p})) \simeq^{E} (\text{signal } \mathbf{S}_2\ (\text{signal } \mathbf{S}_1\ \mathbf{p}))$

PROOF.

1. By [**signal**], $(\text{signal } \mathbf{S}_1\ (\text{signal } \mathbf{S}_2\ \mathbf{p})) \equiv^{\text{E}} (\varrho\ \langle\{\ \mathbf{S}_1 \mapsto \bot\ \}, \text{WAIT}\rangle.\ (\text{signal } \mathbf{S}_2\ \mathbf{p}))$

2.

2.1. By [**signal**], $(\text{signal } \mathbf{S}_2\ \mathbf{p}) \equiv^{\text{E}} (\varrho\ \langle\{\ \mathbf{S}_2 \mapsto \bot\ \}, \text{WAIT}\rangle.\ \mathbf{p})$

2.2. By [**ctx**] and (2.1), $(\varrho\ \langle\{\ \mathbf{S}_1 \mapsto \bot\ \}, \text{WAIT}\rangle.\ (\text{signal } \mathbf{S}_2\ \mathbf{p})) \equiv^{\text{E}} (\varrho\ \langle\{\ \mathbf{S}_1 \mapsto \bot\ \}, \text{WAIT}\rangle.\ (\text{signal } \mathbf{S}_2\ \mathbf{p}))$

3. By [**merge**], $(\varrho\ \langle\{\ \mathbf{S}_1 \mapsto \bot\ \}, \text{WAIT}\rangle.\ (\varrho\ \langle\{\ \mathbf{S}_2 \mapsto \bot\ \}, \text{WAIT}\rangle.\ \mathbf{p})) \equiv^{\text{E}} (\varrho\ \langle\{\ \mathbf{S}_1 \mapsto \bot\ \} \leftarrow \{\ \mathbf{S}_2 \mapsto \bot\ \}, \text{WAIT}\rangle.\ \mathbf{p})$

4.

4.1. By [**merge**], $(\varrho\,\langle\{\,\mathbf{S}_2 \mapsto \bot\,\}, \mathrm{WAIT}\rangle.\ (\varrho\,\langle\{\,\mathbf{S}_1 \mapsto \bot\,\}, \mathrm{WAIT}\rangle.\ \mathbf{p})) \equiv^{\mathrm{E}} (\varrho\,\langle\{\,\mathbf{S}_1 \mapsto \bot\,\} \leftarrow \{\,\mathbf{S}_2 \mapsto \bot\,\}, \mathrm{WAIT}\rangle.\ \mathbf{p})$

4.2. By [**sym**] and (4.1), $(\varrho\,\langle\{\,\mathbf{S}_1 \mapsto \bot\,\} \leftarrow \{\,\mathbf{S}_2 \mapsto \bot\,\}, \mathrm{WAIT}\rangle.\ \mathbf{p}) \equiv^{\mathrm{E}} (\varrho\,\langle\{\,\mathbf{S}_2 \mapsto \bot\,\}, \mathrm{WAIT}\rangle.\ (\varrho\,\langle\{\,\mathbf{S}_1 \mapsto \bot\,\}, \mathrm{WAIT}\rangle.\ \mathbf{p}))$

5.

5.1.

5.1.1. By [**signal**], $(\mathrm{signal}\ \mathbf{S}_2\ (\mathrm{signal}\ \mathbf{S}_1\ \mathbf{p})) \equiv^{\mathrm{E}} (\varrho\,\langle\{\,\mathbf{S}_2 \mapsto \bot\,\}, \mathrm{WAIT}\rangle.\ (\mathrm{signal}\ \mathbf{S}_1\ \mathbf{p}))$

5.1.2.

5.1.2.1. By [**signal**], $(\mathrm{signal}\ \mathbf{S}_1\ \mathbf{p}) \equiv^{\mathrm{E}} (\varrho\,\langle\{\,\mathbf{S}_1 \mapsto \bot\,\}, \mathrm{WAIT}\rangle.\ \mathbf{p})$

5.1.2.2. By [**ctx**] and (5.1.2.1), $(\varrho\,\langle\{\,\mathbf{S}_2 \mapsto \bot\,\}, \mathrm{WAIT}\rangle.\ (\mathrm{signal}\ \mathbf{S}_1\ \mathbf{p})) \equiv^{\mathrm{E}} (\varrho\,\langle\{\,\mathbf{S}_2 \mapsto \bot\,\}, \mathrm{WAIT}\rangle.\ (\mathrm{signal}\ \mathbf{S}_1\ \mathbf{p}))$

5.1.3. By [**trans**], and (5.1.1) through (5.1.2), $(\mathrm{signal}\ \mathbf{S}_2\ (\mathrm{signal}\ \mathbf{S}_1\ \mathbf{p})) \equiv^{\mathrm{E}} (\varrho\,\langle\{\,\mathbf{S}_2 \mapsto \bot\,\}, \mathrm{WAIT}\rangle.\ (\varrho\,\langle\{\,\mathbf{S}_1 \mapsto \bot\,\}, \mathrm{WAIT}\rangle.\ \mathbf{p}))$

5.2. By [**sym**] and (5.1), $(\varrho\,\langle\{\,\mathbf{S}_2 \mapsto \bot\,\}, \mathrm{WAIT}\rangle.\ (\varrho\,\langle\{\,\mathbf{S}_1 \mapsto \bot\,\}, \mathrm{WAIT}\rangle.\ \mathbf{p})) \equiv^{\mathrm{E}} (\mathrm{signal}\ \mathbf{S}_2\ (\mathrm{signal}\ \mathbf{S}_1\ \mathbf{p}))$

6. By [**trans**], and (1) through (5), $(\text{signal } \mathbf{S}_1\ (\text{signal } \mathbf{S}_2\ \mathbf{p})) \equiv^{\text{E}} (\text{signal } \mathbf{S}_2\ (\text{signal } \mathbf{S}_1\ \mathbf{p}))$

By the above and theorem 29 (Soundness), we may conclude that $(\text{signal } \mathbf{S}_1\ (\text{signal } \mathbf{S}_2\ \mathbf{p})) \simeq^{E} (\text{signal } \mathbf{S}_2\ (\text{signal } \mathbf{S}_1\ \mathbf{p}))$.

□

Theorem 85 (Can take the else branch for adjacent signals)**.**

For all $\mathbf{S}$, $\mathbf{p}$, $\mathbf{q}$, *If* $\mathbf{S} \notin \mathit{Can}^{S}(\mathbf{p}, \{\ \mathbf{S} \mapsto \bot\ \})$ *and,* $\mathbf{S} \notin \mathit{Can}^{S}(\mathbf{q}, \{\ \mathbf{S} \mapsto \bot\ \})$, *then*

$(\text{signal } \mathbf{S}\ (\text{if } \mathbf{S}\ \mathbf{p}\ \mathbf{q})) \simeq^{E} (\text{signal } \mathbf{S}\ \mathbf{q})$

Proof.

1. By [**signal**], $\begin{matrix}(\text{signal } \mathbf{S} \\ (\text{if } \mathbf{S}\ \mathbf{p}\ \mathbf{q}))\end{matrix} \equiv^{\text{E}} \begin{matrix}(\varrho\ \langle\{\ \mathbf{S} \mapsto \bot\ \}, \text{WAIT}\rangle. \\ (\text{if } \mathbf{S}\ \mathbf{p}\ \mathbf{q}))\end{matrix}$

2. By [**is-absent**], $\begin{matrix}(\varrho\ \langle\{\ \mathbf{S} \mapsto \bot\ \}, \text{WAIT}\rangle. \\ (\text{if } \mathbf{S}\ \mathbf{p}\ \mathbf{q}))\end{matrix} \equiv^{\text{E}} \begin{matrix}(\varrho\ \langle\{\ \mathbf{S} \mapsto \bot\ \}, \text{WAIT}\rangle. \\ \mathbf{q})\end{matrix}$

3.

   3.1. By [**signal**], $\begin{matrix}(\text{signal } \mathbf{S} \\ \mathbf{q})\end{matrix} \equiv^{\text{E}} \begin{matrix}(\varrho\ \langle\{\ \mathbf{S} \mapsto \bot\ \}, \text{WAIT}\rangle. \\ \mathbf{q})\end{matrix}$

   3.2. By [**sym**] and (3.1), $\begin{matrix}(\varrho\ \langle\{\ \mathbf{S} \mapsto \bot\ \}, \text{WAIT}\rangle. \\ \mathbf{q})\end{matrix} \equiv^{\text{E}} \begin{matrix}(\text{signal } \mathbf{S} \\ \mathbf{q})\end{matrix}$

4. By [**trans**], and (1) through (3), $(\text{signal } \mathbf{S}\ (\text{if } \mathbf{S}\ \mathbf{p}\ \mathbf{q})) \equiv^{\text{E}} (\text{signal } \mathbf{S}\ \mathbf{q})$

By the above and theorem 29 (Soundness), we may conclude that $(\text{signal } \mathbf{S}\ (\text{if } \mathbf{S}\ \mathbf{p}\ \mathbf{q})) \simeq^{E} (\text{signal } \mathbf{S}\ \mathbf{q})$. □

Theorem 86 (Lifting signals)**.**

For all $\mathbf{S}$, $\mathbf{p}$, $\mathbf{E}$, $\mathbf{A}$, $(\varrho\ \langle\{\}, \mathbf{A}\rangle.\ \mathbf{E}[(\text{signal } \mathbf{S}\ \mathbf{p})]) \simeq^{E} (\varrho\ \langle\{\}, \mathbf{A}\rangle.\ (\text{signal } \mathbf{S}\ \mathbf{E}[\mathbf{p}]))$

PROOF.

1.

1.1. By [**signal**], $(\text{signal } \mathbf{S}\ \mathbf{p}) \equiv^{\mathrm{E}} (\varrho\ \langle\{\ \mathbf{S} \mapsto \bot\ \}, \mathrm{WAIT}\rangle.\ \mathbf{p})$

1.2. By [**ctx**] and (1.1), $(\varrho\ \langle\{\}, \mathbf{A}\rangle.\ \mathbf{E}[(\text{signal } \mathbf{S}\ \mathbf{p})]) \equiv^{\mathrm{E}} (\varrho\ \langle\{\}, \mathbf{A}\rangle.\ \mathbf{E}[(\text{signal } \mathbf{S}\ \mathbf{p})])$

2. By [**merge**], $(\varrho\ \langle\{\}, \mathbf{A}\rangle.\ \mathbf{E}[(\varrho\ \langle\{\ \mathbf{S} \mapsto \bot\ \}, \mathrm{WAIT}\rangle.\ \mathbf{p})]) \equiv^{\mathrm{E}} (\varrho\ \langle\{\ \mathbf{S} \mapsto \bot\ \}, \mathbf{A}\rangle.\ \mathbf{E}[\mathbf{p}])$

3.

3.1. By [**merge**], $(\varrho\ \langle\{\}, \mathbf{A}\rangle.\ (\varrho\ \langle\{\ \mathbf{S} \mapsto \bot\ \}, \mathrm{WAIT}\rangle.\ \mathbf{E}[\mathbf{p}])) \equiv^{\mathrm{E}} (\varrho\ \langle\{\ \mathbf{S} \mapsto \bot\ \}, \mathbf{A}\rangle.\ \mathbf{E}[\mathbf{p}])$

3.2. By [**sym**] and (3.1), $(\varrho\ \langle\{\ \mathbf{S} \mapsto \bot\ \}, \mathbf{A}\rangle.\ \mathbf{E}[\mathbf{p}]) \equiv^{\mathrm{E}} (\varrho\ \langle\{\}, \mathbf{A}\rangle.\ (\varrho\ \langle\{\ \mathbf{S} \mapsto \bot\ \}, \mathrm{WAIT}\rangle.\ \mathbf{E}[\mathbf{p}]))$

4.

4.1.

4.1.1. By [**signal**], $(\text{signal } \mathbf{S}\ \mathbf{E}[\mathbf{p}]) \equiv^{\mathrm{E}} (\varrho\ \langle\{\ \mathbf{S} \mapsto \bot\ \}, \mathrm{WAIT}\rangle.\ \mathbf{E}[\mathbf{p}])$

4.1.2. By [**sym**] and (4.1.1), $(\varrho\ \langle\{\ \mathbf{S} \mapsto \bot\ \}, \mathrm{WAIT}\rangle.\ \mathbf{E}[\mathbf{p}]) \equiv^{\mathrm{E}} (\text{signal } \mathbf{S}\ \mathbf{E}[\mathbf{p}])$

4.2. By **[ctx]** and (4.1), $(\varrho\ \langle\{\}, \mathbf{A}\rangle.\ (\varrho\ \langle\{\ \mathbf{S} \mapsto \bot\ \}, \mathrm{WAIT}\rangle.\ \mathbf{E}[\mathbf{p}])) \equiv^{E} (\varrho\ \langle\{\}, \mathbf{A}\rangle.\ (\varrho\ \langle\{\ \mathbf{S} \mapsto \bot\ \}, \mathrm{WAIT}\rangle.\ \mathbf{E}[\mathbf{p}]))$

5. By [**trans**], and (1) through (4), $(\varrho\ \langle\{\}, \mathbf{A}\rangle.\ \mathbf{E}[(\mathrm{signal}\ \mathbf{S}\ \mathbf{p})]) \equiv^{E} (\varrho\ \langle\{\}, \mathbf{A}\rangle.\ (\mathrm{signal}\ \mathbf{S}\ \mathbf{E}[\mathbf{p}]))$

By the above and theorem 29 (Soundness), we may conclude that $(\varrho\ \langle\{\}, \mathbf{A}\rangle.\ \mathbf{E}[(\mathrm{signal}\ \mathbf{S}\ \mathbf{p})]) \simeq^{E} (\varrho\ \langle\{\}, \mathbf{A}\rangle.\ (\mathrm{signal}\ \mathbf{S}\ \mathbf{E}[\mathbf{p}]))$.

□

Index