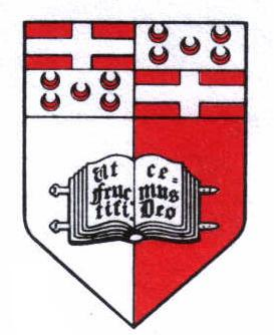
Real-Time Logics and Slowdown Invariance for Runtime Verification

Ingram Bondin

Supervised by Dr.Gordon Pace



University of Malta
Faculty of ICT
Department of Computer Science and AI

May 2009

Submitted in partial fulfillment of the requirements for the degree of B.Sc. I.T. (Hons.)

I would like to thank Dr. Gordon Pace and Christian Colombo for their help regarding the achievement of this work. I also thank Daniel D'Agostino for proof reading parts of this text.

Abstract

In many cases, it is desirable to check whether a program satisfies some constraint on its behaviour. Such a constraint is known as a property. Sometimes, these constraints involve the element of time. For example, the software controlling a car spraying arm might be required to move away from the assembly line within 3 seconds of completing its task lest it crash into an oncoming car. Properties are expressed in various logics. Amongst the logics used to express time related properties is the formalism known as timed regular expressions. Properties can be very useful. In runtime verification, for example, they can be automatically compiled into monitors which can be added to the system's code in order to check whether a system's runtime behaviour is obeying the constraints or not.

A system's behaviour can be affected in various ways. One common effect is for systems to slow down or speed up. For example, upgrading the underlying hardware will speed up a system. On the other hand, a system which is being subjected to a large number of requests will invariably slow down due to its load. Yet another scenario occurs when one introduces monitors into the system. In this case, adding monitors means that the system has more code to execute, slowing it down. On the other hand, removing the monitors once one has sufficient confidence in the system will speed up the system. Unfortunately, speeding up and slowing down of a system is a very severe problem when time constraints are involved. The implications are that systems which previously satisfied the constraints, might not satisfy them anymore. One tool which can be used to analyse when properties could be potentially broken, and when they are safe, is the theory of slowdown and speedup introduced by Colombo et al. The authors use this framework to reason about the real-time logic duration calculus, in order to identify a fragment of this logic which is safe from the problems of speedup, slowdown, or both. Once such a fragment has been identified, one can be sure that properties written using that fragment have certain guarantees.

The theory of slowdown and speedup can be applied to other logics besides duration calculus. However, when the semantics of the logic in question are defined using a different model from that used by duration calculus, this can prove to be quite challenging. To be able to apply the theory in such scenarios we propose two different approaches, and substantiate them by applying them to prove slowdown and speedup results for timed regular expressions.

Contents

List of Figures

Chapter 1

Introduction

1.1 Introduction and Background

In the field of verification, we often want to study the behaviour of programs. One way of doing this is to describe, in some way, the behaviour of a system as a set of execution traces, which in effect, is a formal language. There is more than one way to represent these traces. For example, in a suitably modular program, one could represent a trace as a sequence of method calls.

Once one is capable of representing the behaviour of the system as a set of traces, it then becomes easier to ask whether a system satisfies some particular constraint. A constraint on a system is often referred to as a *property*. In essence, a property also characterises a set of traces, namely those traces which satisfy the constraint it represents. Checking whether a system obeys a property is therefore a question of deciding whether all the traces executable by the system are also in the set characterised by the property. Any trace which is not in the property's set, violates that property. This is described pictorially in figure 1.1, with the system on the left satisfying the property, and that on the right failing to do so.

How does one write properties? Or rather, how does one characterise legal sets of traces? This is done by using some particular logic, such as *regular expressions* or *finite state automata*. An expression in such a logic is essentially a compact way to characterise an entire set of behaviour. For example, using the regular expression $(login \circ request \circ logout)^*$, one accepts all traces in which the user logs in before making a request, and in which he/she subsequently logs out. Due to the star operation, a user can make any number of these transactions without breaking the property.

Having different logics available for modeling properties is useful, even if they are equivalent in power, since one logic might be more suitable for modeling certain kinds of properties than another.

There are many types of properties. Sometimes we wish our properties to characterise constraints involving time. Here one must distinguish between a *qualitative* notion of time and a quantitative or *real-time* notion [AD94]. The logics discussed previously are useful if one wants to reason about a qualitative notion of time, such as the sequence in which the method calls occour. However, on other occasions, we require our properties to express quantitative constraints on time, in which we are interested in the actual duration

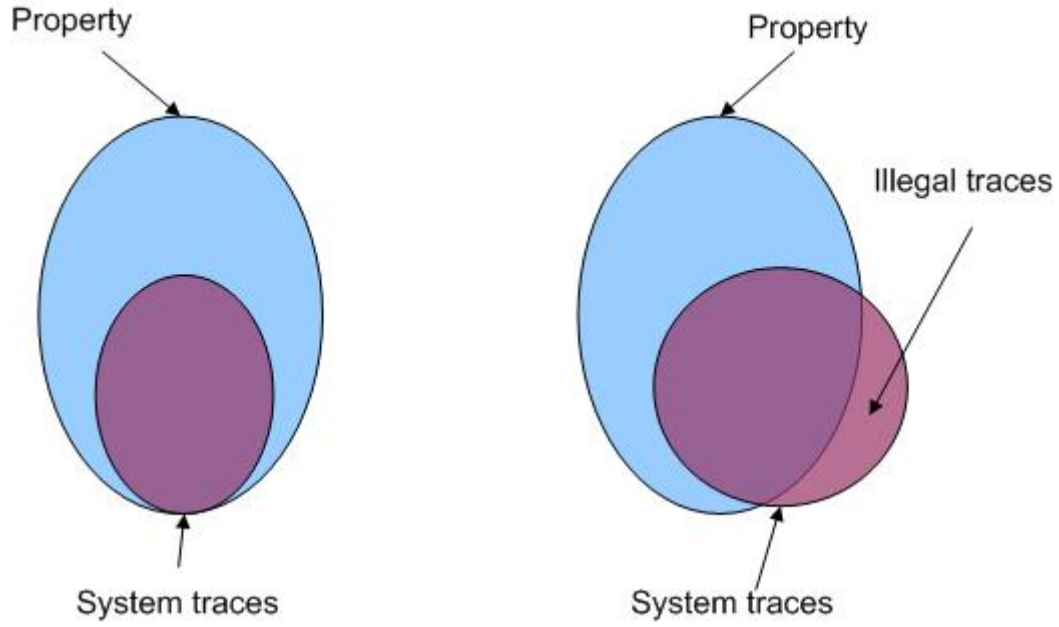


Figure 1.1: Properties and System Behaviour.

of the events.

As an example of a real-time constraint, we might wish a car spraying arm to move off the conveyor belt within at most three seconds of completing its task, lest it crash into another oncoming car. Specifying real-time properties requires logics that are intended for that purpose, which are referred to as *temporal logics*. Examples of such logics are timed automata [AD94], timed regular expressions [ACM97] and duration calculus [CHR91].

Regardless of the logic the properties are expressed in, they are useful because they can be used for a variety of applications, such as in runtime verification.

1.1.1 Runtime Verification

In runtime verification, segments of code called monitors are used to observe the runtime behaviour of a system. This information is then used to consult an oracle which will determine whether the execution trace satisfies the required behaviour [CM04]. In this case, the required behaviour is effectively a property, and the oracle is a way of deciding whether the trace satisfies the property of not. Runtime verification frameworks such as LARVA [Col08][CPS08], enable the user to formulate a property in some logic which a system must satisfy. They will then automatically generate monitors and an oracle, and will instrument (insert) them into the system's code. More generally, we shall use the term *monitor* to denote both the monitoring code (which detects the events) and the oracle sub-system which tells us whether a violation has occured. LARVA also allows monitors to take remedial action if the system violates the property. Figure 1.2 summaries what we have said.

We note here that the monitor and the system are essentially merged during the instrumentation process and thus become a new system. Thus monitoring (observing) the system changes the system, which [CPS] refers to as a Heisenberg effect. We shall

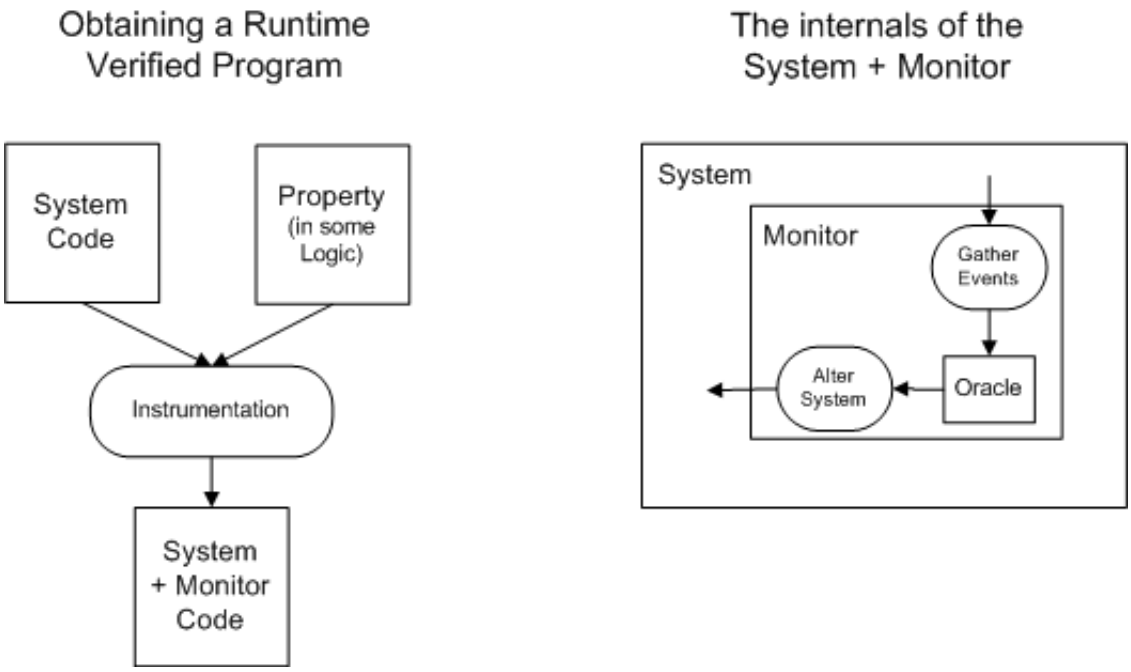


Figure 1.2: Using Properties to Generate Monitors and The Monitored System.

soon see that this factor will introduce a number of side effects. But first, we must discuss the phenomenon of system slowdown and system speedup.

1.1.2 Slowdown and Speedup of Systems

There are various scenarios in which systems can slow down or speed up. For example, upgrading the underlying hardware on which a piece of software is executing can speed up the system. On the other hand, a system which is being subjected to a large number of requests will invariably slow down due to its load. Yet another scenario occurs when one introduces monitors into the system. In this case, adding monitors means that the system has more code to execute, slowing it down. On the other hand, if for example, we use monitors to test the system during development and remove them once we have enough confidence in it, then it stands to reason that the release version will be faster than the version which was being monitored. In the case of monitoring, the slowdown experienced might reduce the effectiveness of the verification process, since the response time of the system decreases. However, besides this nuisance, slowing down and speeding up can have more sinister consequences, if the property we want to check for involves real-time constraints. We shall discuss these effects in the next section. The discussion is based on material from [CPS].

1.1.3 Slowdown, Speedup and Runtime Verification

Let us consider again, for the sake of illustration the car spraying arm example. Suppose for a moment that the software *does* satisfy its real time constraint, such that the arm moves away from the assembly line within 3 seconds of completing its task. Now suppose

we add a monitor to check for this property. It might be the case that the monitoring code slows down the system sufficiently to cause it to stop satisfying the property. Thus, when this happens, the monitor will report (wrongly) that the system does not satisfy the property. Why does this happen? It occurs because when monitoring the system using runtime verification, we are not checking whether the system obeys the property. We are checking whether the system plus the instrumented monitor obey the property.

There is a similar problem concerning the possible speed up of a system. Suppose that after using monitors to test a system, we are sufficiently confident of its behaviour that we decide to remove them. However, removing this code will speed up the system. It might be the case that the reason why some property was being satisfied was because the monitor was slowing it down sufficiently for this to be the case. Clearly this is another undesirable effect; the system will fail to satisfy the property when it is deployed. How can one reason about this phenomenon? One possible way is discussed in the next section.

1.1.4 The Theory of Slowdown and Speedup

In [Col08], Colombo suggests that if one had monitors with negligible overhead, then this problem could be attenuated (since the resulting slowdown/speedup effect would also be negligible). However, the author also asserts that this is unrealistic. As an alternative solution, in [CPS] Colombo, Pace and Schneider, devise a theory of slowdown and speedup for properties, in order to study which properties are affected by this phenomenon.

How does such a theory work? Recall that properties are written using the formulas of some logic. These formulas are built from the operators of the logic. Now, it is the case that some operators are inherently safe from the effects of system slowdown and system speedup. If properties are written using only these operators, it can be guaranteed that these properties will not be affected adversely by system slowdown or system speedup. The major contribution of this theory is that it allows one to analyse which operators are safe and which are not. This is depicted in figure 1.3.

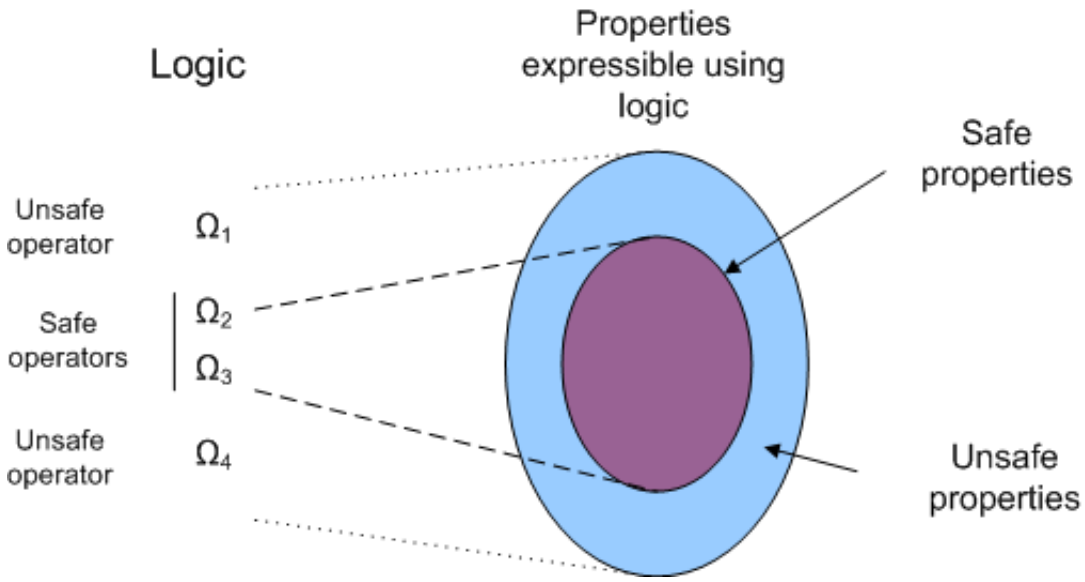


Figure 1.3: Using Safe Operators Yields Safe Properties.

In the figure we show a hypothetical logic with four operators, Ω_1, Ω_2, Ω_3 and Ω_4, out of which Ω_1 and Ω_4 are unsafe, and the other two are safe. We see that by using only combinations of safe operators, we will end up with a property out of the set of safe properties which that logic can express. As a small example of a safe property, suppose our logic has an operator $\oplus$ which takes an event and a duration k. The property $\oplus(event_x, 3)$ states that $event_x$ must last for 3 seconds or longer. This property satisfies a trait called *slowdown truth preservation*. This means that if a system satisfies the property, then, slowing down the system by any amount will not break the property. This is the case because slowing down the system will force $event_x$ to take longer, which is allowed. In fact, an operator such as $\oplus$ can be proven to be always *slowdown truth preserving*, which means that any properties written using it and other slowdown truth preserving operators, are guaranteed to be unaffected by system slowdown. In the next section we shall show some possible hurdles to applying the theory to temporal logics.

1.1.5 Applying The Theory of Slowdown and Speedup

In order to show the power of this concept in [Col08] and [CPS] fragments of duration calculus are proved to be safe with respect to slowdown, speedup or both. The problem with this theory is that it cannot be applied directly to all kinds of temporal logics. The reason for this has to do with the fact that logics may use different models of time when defining their semantics. By this we mean that they have different ways of how to express the traces which are then characterised by the formulas of the logic. For example, in duration calculus, the behaviour of a system as time passes (its trace) is represented by a total function which given a time and an event will tell us whether that event is executing at that point in time. This concept is called an *interpretation*. Asarin et al.'s timed regular expressions [ACM97], on the other hand, use a model known as *signals*. A signal is represented as a string of events and their durations. For example, the signal $request^1 download^{7.5}$ tells us that the system has taken 1 unit of time executing the *request* method and 7.5 units of time executing the *download* method. We now have enough background to understand the aims of this work.

1.2 Aim of this Work

As we have said before, having different logics is useful since different properties are modeled more easily using some logics than others. We also know that a fragment of duration calculus has been proved to be safe by Colombo et al. What we propose is to derive similar results for another logic, namely Asarin et al.'s timed regular expressions [ACM97]. As we shall see, this will present several challenges.

The major problem to be solved is that the theory of slowdown and speedup works with interpretations as its underlying model. The reason it could easily be applied to duration calculus, was because the semantics of duration calculus are also defined using interpretations. However, as we have seen in the previous section, timed regular expressions work using signals. This problem is not unique to timed regular expressions; it needs to be faced whenever we want to apply the theory of slowdown and speedup to a logic which does not use interpretations as its underlying model.

Let us now consider the approaches available to us through which we can apply the theory of slowdown and speedup to such logics. In this discussion, we assume that the logic we want to derive results for is that of timed regular expressions. Consider the diagram in figure 1.4.

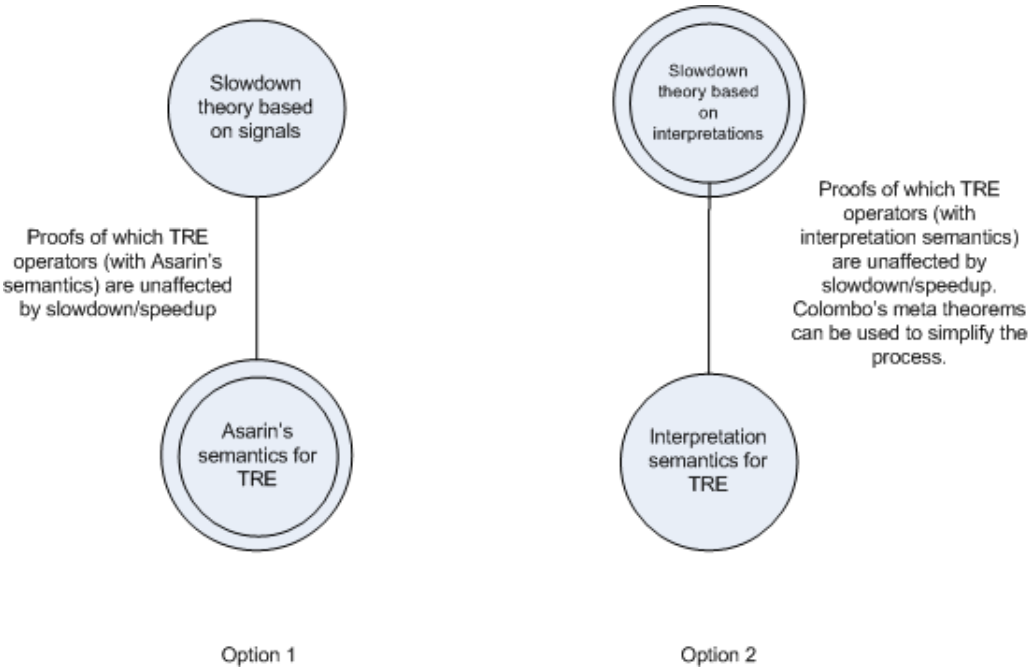


Figure 1.4: Context of the work

The double circled objects in this diagram indicate the prior work on which we can build. The double circle on the left shows that we possess a semantics for timed regular expressions (TRE) based on signals (as characterised by Asarin et al.). The double circle on the right shows that we possess a theory of slowdown and speedup which works for any logic which can be given a semantics in terms of interpretations.

This leaves us with two options, in order to achieve our goal. The first option, shown on the left of the diagram, is to develop a new theory of slowdown and speedup which uses signals as an underlying model. Then one would prove slowdown and speedup results for TRE operators using their signal semantics and this new theory. The second option, shown on the right, would be to give timed regular expressions a new semantics based on interpretations, and then use the existing theory to prove that TRE operators under these semantics satisfy certain slowdown/speedup properties.

Both these options have their deficiencies. In the first option, we have no guarantee that the theory of slowdown/speedup based on signals is equivalent to Colombo et al.'s formulation. The problem with the second option is that it would be unclear whether or not the interpretation semantics for TRE are equivalent to those given by Asarin et al.

This leaves us with two choices, each arising from Option 1 and Option 2, which we shall refer to as Options $1'$ and $2'$ respectively. We can see this arrangement in figure 1.5.

In Option $1'$, we take the approach of Option 1, but we also prove that the new theory of slowdown and speedup based on signals is sound with Colombo et al.'s existing theory and vice-versa.

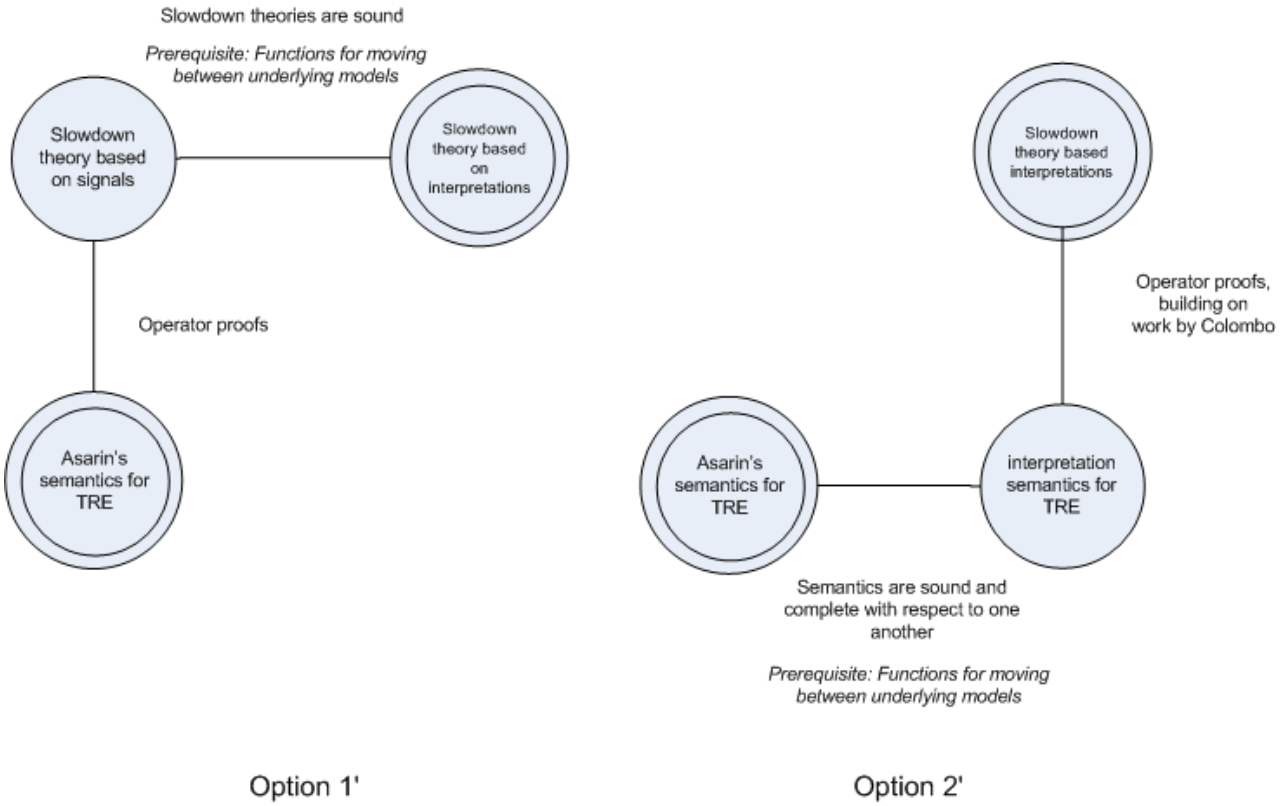


Figure 1.5: Options 1' and 2'

In Option 2′, we define a new semantics for TRE based on interpretations, such that we can apply Colombo et al.'s theory directly in order to prove the slowdown and speedup results for TRE. In addition to this we then prove that the new TRE semantics based on interpretations are sound and complete with respect to Asarin et al.'s semantics. We can visualise these two options together in figure 1.6.

Due to the layout of this diagram we shall sometimes refer to task 1′ as the upper triangle and task 2′ as the lower triangle. Irrespectively of which option we choose, one should note that we are working with two different notions, the signal one given by Asarin et al., and the interpretation one used by Colombo et al. If we are going to relate the two notions, we are going to need some way of moving from one model to the other. This will be the object of sections 2.2 and 2.3, and will act as the groundwork for the rest of the work.

What are the advantages of each of the approaches? If Option 1′ is chosen, then, if we need to prove slowdown and speedup results for some new logic which has its semantics grounded in signals (such as Asarin et al.'s formulation of Timed Automata [ACM97]), we could do this by directly appealing to the theory of slowdown/speedup based on signals. The soundness result between the slowdown/speedup theories would guarantee that the results proven are consistent with Colombo et al.'s theory.

If Option 2′ is chosen, then it becomes possible to reuse many of Colombo's theorems

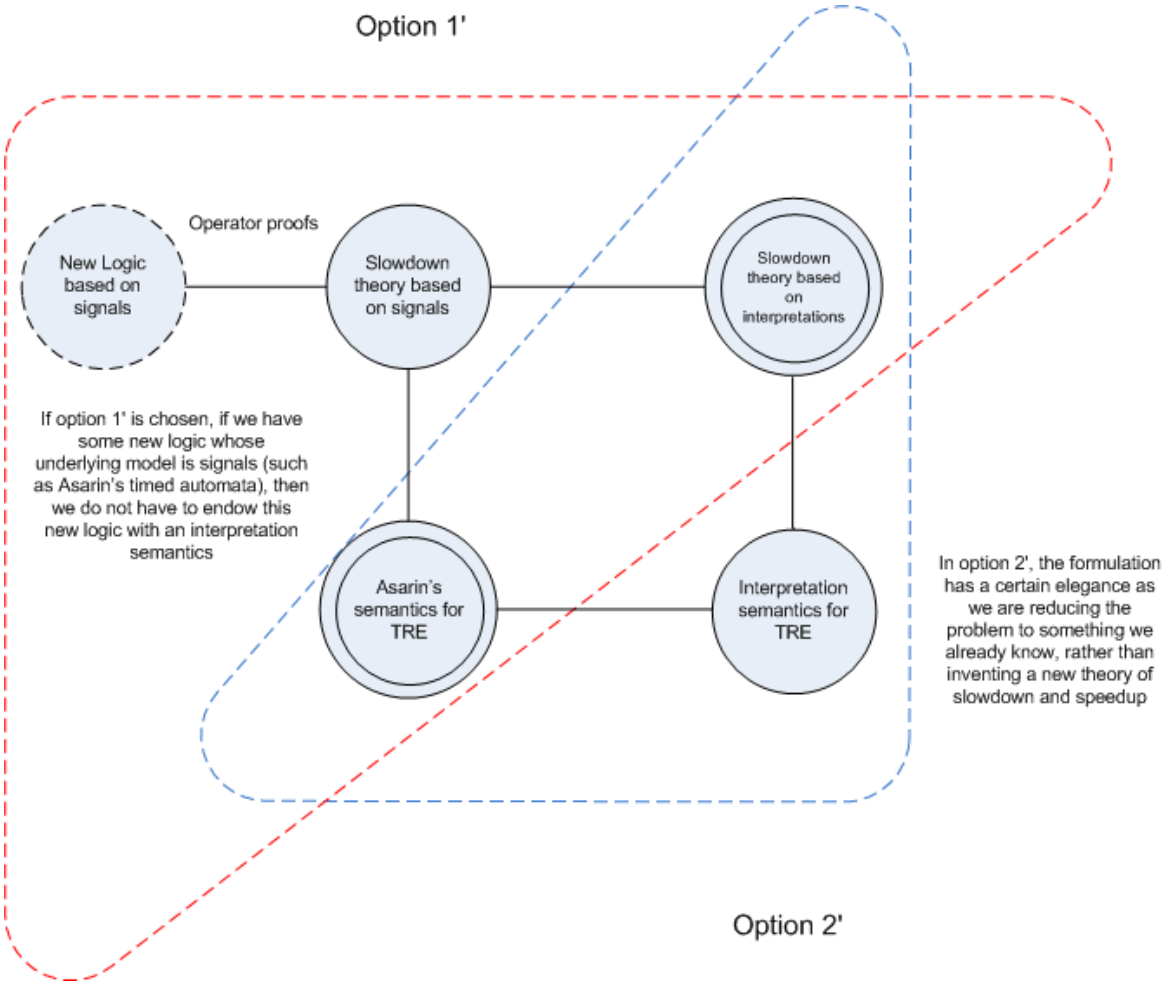


Figure 1.6: Upper and Lower Triangle

relating to slowdown and speedup without having to prove them from scratch. This results in a more elegant formulation of speedup/slowdown results for TRE. The focus of this work will thus be on Option $2'$, although in section 4.4 we shall also solve elements of Option $1'$ which will allow us to obtain the advantage mentioned above.

In investigating these two approaches, we shall also discover the details which are involved in applying each one of them. We shall also get an idea of the relative complexity of applying the theory of slowdown and speedup to logics which do not have interpretation based semantics.

Since our major thrust will be on the approach we have denoted as $2'$, below we enumerate the sub problems we need to solve in order to complete this task. These are as follows.

1. Define an interpretation based semantics for timed regular expressions.

2. Prove that Asarin et al.'s semantics are sound and complete with respect to the interpretation semantics.

3. Prove which TRE operators are safe with respect to slowdown and speedup and which are not.

Before we embark on any of these tasks, we shall first lay the groundwork which we shall use in the rest of this work.

Chapter 2

Models of Time

2.1 Introduction

In this chapter we lay the foundations for the rest of this work. In the previous chapter, we outlined the fact that there can be different models which describe a system's behaviour through time. Amongst these, we mentioned briefly the two concepts of signals and interpretations. This chapter consists of two major sections. In section 2.2, we shall make these two notions unambiguous by providing a formal theory of how they can be manipulated. Once this has been achieved, in section 2.3, we shall provide constructions which will allow us to relate the models of signals and interpretations. Being able to relate signals and interpretations in this way is important since often a signal and an interpretation will be describing the behaviour of exactly the same system. Throughout the rest of the work, we shall delegate all the proofs to an appendix in order not to disrupt the main flow of the argument.

2.2 A Formal Theory for Models of Time

In chapter 1, we mentioned vaguely the notions of a signal and that of an interpretation, without going into any detail. We also said that signals were the underlying model for TRE, and that interpretations were the underlying model for duration calculus. It is thus crucial to make rigorous what is meant by the notion of a signal, and by that of an interpretation. We shall start by stating Asarin et al.'s definition of a signal [ACM97].

2.2.1 Signals

A signal is an object defined over some finite set of symbols Σ. These symbols can be used to represent events, such as method calls. The entire signal ranges over a finite amount of time. This means that it can represent the behaviour of a system which is online for only a finite number of time units. At any point in time, a signal tells us which symbol (event) is active. The constraint here, is that only one symbol (event) can be active at any one time. This is similar to a program running on a single-processor, single tasking system; only one method can be executing at any given time. Moreover, some symbol

must always be active at a given time. Figure 2.1 shows an example of a signal $a^1b^2c^1$ over the alphabet $\Sigma = \{a, b, c\}$.

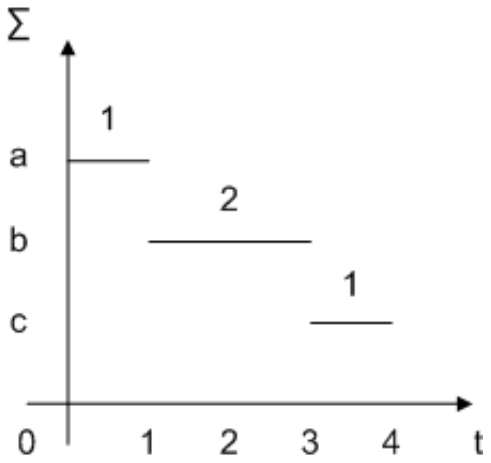


Figure 2.1: A Signal

Let us now be slightly more formal in defining signals. We have already determined that symbols are active for a certain duration. In Asarin et al.'s model, these durations are represented by the positive real numbers. We also said that one and only one symbol can be active at a time. How can one characterise this mutual exclusion requirement? One way is to formally define the concept of a signal as a *total function* which for every point in time, will give us the *one symbol active* at that time point. The following definition, adapted from [ACM97] captures this notion.

Definition 1. *Let Σ be a finite set of symbols. Then, a signal over Σ is a piecewise constant function from an interval (0,k] to Σ, where $k \in \mathbb{R}_0^+$, and which has a finite number of discontinuities.*

The interval which is mentioned in the above definition represents the finite span of time over which we are observing the system. The term piecewise constant indicates that a signal is made up of a number of segments, with each segment representing some event which is active for some duration, in the same spirit as the signal shown in figure 2.1. The final requirement, sometimes called *finite variability*, prohibits the signal from changing value continuously. If a symbol is active, it cannot be active for an infinitesimal amount of time.

How does this definition of a signal differ from the concept of an interpretation? Essentially, an interpretation does not have the mutual exclusion requirement required from signals. It can have more than one symbol active at the same time. Figure 2.2 shows an interpretation which is not a legal signal. It is not legal, because symbols b and c overlap between the times 2 and 3.

Definition 1 is not the only way to characterise the notion of a signal. An alternative definition, given in the same paper by Asarin et al. is the following.

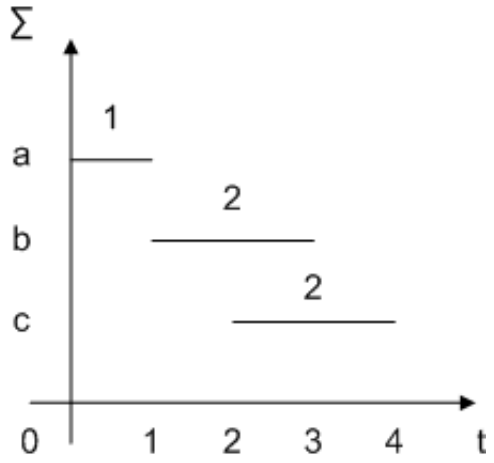


Figure 2.2: A Duration Calculus Interpretation

Definition 2. *A signal ζ of length k can be written as*

$$\zeta = a_1^{r_1} a_2^{r_2} \dots a_n^{r_n}$$

where $a_i \in \Sigma$, $r_i \in \mathbb{R}^+$, $a_i \neq a_{i+1}$ and $\Sigma r_i = k$

Note the constraint in which no two consecutive symbols can be the same. It is always possible, of course, to merge two adjacent symbols which are identical into one symbol with the sum of their lengths as its duration.

We shall use this definition since we feel it encapsulates better the correct intuition about a signal being an object which describes a sequence of events, with each event being active for some duration. Moreover, as we shall see later, signals are easier to manipulate in this form. Asarin et al. define only two operations on signals. The first is called the *length* of a signal.

Definition 3. *If ζ is a signal, then we shall denote its length by $|\zeta|$. The length of a signal has the same value as the sum of the exponents in Definition 2.*

The second operation is called *signal concatenation*, and can be used to append a signal to the end of another signal. Asarin et al. define concatenation using Definition 1 as follows.

Definition 4. *Let ζ_1 and ζ_2 be two signals. The result of their concatenation, denoted by $\zeta_1 \circ \zeta_2$, is to append ζ_2 to ζ_1, yielding a new signal which is equal in length to the sum of the lengths of ζ_1 and ζ_2. If we let $\zeta = \zeta_1 \circ \zeta_2$ we can state this as follows.*

$$\zeta(t) = \begin{cases} \zeta_1(t) \text{ for } t \in (0, |\zeta_1|] \\ \zeta_2(t - |\zeta_1|) \text{ for } t \in (|\zeta_1|, |\zeta_1| + |\zeta_2|] \end{cases}$$

The above definition is saying that the new signal agrees with the first one for all points between 0 and its length and with the second one for all points of the appended part. An example of signal concatenation is shown in Figure 2.3.

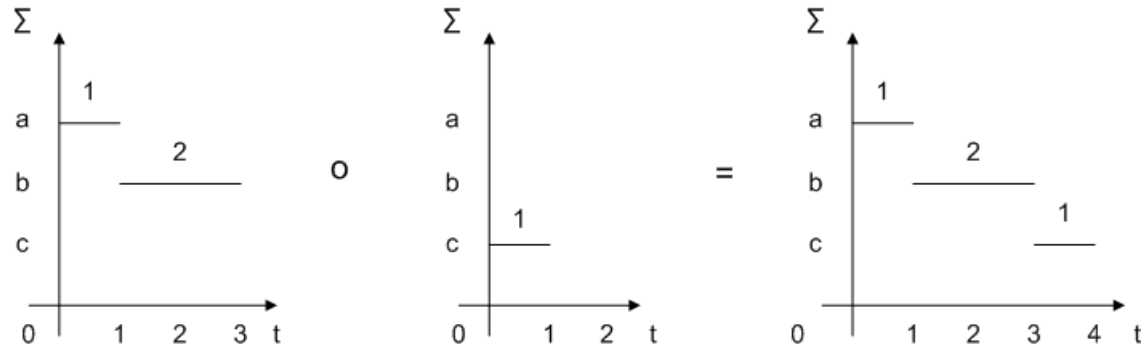


Figure 2.3: Signal Concatenation

Concatenation can also be lifted to operate on sets of signals, and the result is similar to language concatenation, in which each signal from one set is concatenated to all possible signals from the other set.

We have thus made clear what the notion of a signal is. We shall now develop a formal theory for signals. By this we mean that we shall define a precise set of rules of how to build and manipulate signals. As we shall see later, we shall do the same thing for interpretations, creating a framework in which there is no ambiguity in the steps that can be performed when proving results about these structures.

2.2.2 Signal Lists

In order to differ the formal theory of signals from Asarin et al.'s definition, we shall refer to the objects that we shall manipulate formally as signal lists. As we shall see this notion is identical to that of signals.

Definition of Signal Lists

The signal list model characterises a signal as a list of tuples, with each tuple indicating a symbol and the duration of that symbol. The reader will note that this is just syntactic sugar for the notion in Definition 2. If one keeps figure 2.1 in mind, each tuple can be viewed as characterising a segment of the signal, with the sequence of tuples forming the signal itself. We define the notion of a signal list below.

Definition 5. *A signal list over* Σ *is an element of the set*

$$SIG \subset seq(\Sigma \times \mathbb{R}^+)$$

We shall use ξ as a variable denoting a signal list in general. As an example, if the signal is a^3b^2, then we would write it in signal list form as:

$$\xi = \langle (a, 3), (b, 2) \rangle$$

So far we have said that SIG is a subset of $seq(\Sigma \times \mathbb{R}^+)$. But we haven't identified the particular subset. We shall define it as follows.

Definition 6. *Let k be a positive real and let* a *be a symbol from some alphabet Σ. Then, the set of signals over Σ denoted by SIG is defined as*

$$SIG \triangleq \langle\rangle \mid SIG \overset{sig}{;} (a, k)$$

We have thus defined the set as the closure of an operator $\overset{\text{sig}}{;}$ over the *empty signal list* construct $\langle\rangle$. As one can see, signal lists are built by starting with the empty signal lists and adding tuples via the operator $\overset{\text{sig}}{;}$. We define this operator below.

Definition 7.

$$\overset{sig}{;} :: SIG \to (\Sigma \times \mathbb{R}^+) \to SIG$$

$$\langle\rangle \overset{sig}{;} (\alpha, k) \triangleq \langle(\alpha, k)\rangle$$

$$\langle(\alpha_1, k_1), ..., (\alpha_n, k_n)\rangle \overset{sig}{;} (\alpha_{n+1}, k_{n+1}) \triangleq \langle(\alpha_1, k_1), ..., (\alpha_n, k_n), (\alpha_{n+1}, k_{n+1})\rangle$$

where $\alpha_n \neq \alpha_{n+1}$.

If we go back to Definition 2, we see that this arrangement produces a sequence of tuples which corresponds to the sequence of symbols and exponents given in the definition. At this point, one may ask why we have included the constraint $\alpha_n \neq \alpha_{n+1}$. This constraint is a simple mechanism through which we can make sure that no signal list with two identical consecutive symbols is produced, which conforms to the constraint in Definition 2. Other than this constraint, the $\overset{\text{sig}}{;}$ operator is identical to the normal *snoc* operator on lists. We can also lift the append operator to accept entire signal lists as its second parameter, which we call *signal list concatenation*. The simplest way to do this is the following.

Definition 8.

$$\overset{sig}{+\!+} :: SIG \to SIG \to SIG$$

$$\xi \overset{sig}{+\!+} \langle\rangle \triangleq \xi$$

$$(\xi_1 \overset{sig}{+\!+} \xi_2 \overset{sig}{;} (\alpha, k)) \triangleq (\xi_1 \overset{sig}{+\!+} \xi_2) \overset{sig}{;} (\alpha, k)$$

This operator tells us that a signal list with the empty list returns the starting signal list. It also tells us that if we concatenate a signal ξ_1, with another signal whose head is (α, k), the result is equivalent to the concatenation of ξ_1 with the first part of that signal, to which the head is appended later. Essentially this definition decomposes the second signal into a sequence of tuples connected by $\overset{\text{sig}}{;}$, giving an expression which we know how to compute.

We stop for a moment to reflect on this operator. Is it a good model for Asarin et al.'s $\circ$ operator? In fact, it is slightly different. The $\circ$ operator does not work with the symbol list notation of Definition 2, but with the functions of Definition 1. This allows it to easily concatenate two signals which begin and end with the same symbol. However, in the case of signal lists, we cannot naïvely string the two lists together (since we cannot have two consecutive symbols which are identical). Thus, if we were to allow *circle-like*

concatenation, the definition for $\overset{\text{sig}}{+\!\!+}$ would have to be more complex, because it would have to conditionally merge symbols when the first signal ends with the same symbol as the starting symbol of the second one. This definition is not as clean and would, in turn, complicate the proofs.

In fact, *circle-like* concatenation does not give us any additional power to construct signals which otherwise we would not be able to. We shall show this when we will have enough theory to prove that the definition we have given for the set SIG satisfies all the requirements set in Definition 2. To this end, we shall abide by the definition for $\overset{\text{sig}}{+\!\!+}$ given above. We shall now introduce an operator which will allow us find the length of a signal list.

Operators on Signal Lists

We have already seen that Asarin et al. have an operator for finding the length of a signal. To this end, we shall need to define a length operator for signal lists. Obviously we wish for our operator to emulate Asarin et al.'s length operator. Given a signal list, our length operator will return the sum of all the lengths of the segments making up the signal. We denote this operator by $|\ |$ and define it recursively below.

Definition 9.

$$|\ | :: SIG \to \mathbb{R}_0^+$$
$$|\langle\rangle| \triangleq 0$$
$$|\xi \overset{sig}{;} (k, \alpha)| \triangleq |\xi| + k$$

In the above definition, we define that the length of an empty signal list as being equal to zero. For a non empty signal list, its length is recursively obtained by examing each tuple. These are all the operators we shall need. In the next section we prove several useful lemmas about the behaviour of our operators.

Lemmas on Signal List Operators

One property that we would like the signal list concatenation $\overset{\text{sig}}{+\!\!+}$ operator to possess is associativity.

Lemma 1. *$\overset{sig}{+\!\!+}$ is associative. Let ξ_1, ξ_2 and ξ_3 be signal lists. Then,*

$$(\xi_1 \overset{sig}{+\!\!+} \xi_2) \overset{sig}{+\!\!+} \xi_3 = \xi_1 \overset{sig}{+\!\!+} (\xi_2 \overset{sig}{+\!\!+} \xi_3)$$

It is also desirable to show that concatenating a signal list to the empty signal list in any order will yield the same result. Technically, we say that the empty signal list $\langle\rangle$ is the zero under $\overset{\text{sig}}{+\!\!+}$.

Lemma 2. *Let ξ be a signal list. Then, the empty signal list $\langle\rangle$ is the zero of the operation $\overset{sig}{+\!\!+}$. The laws below thus hold.*

$$\xi \overset{sig}{+\!\!+} \langle\rangle = \xi$$

and

$$\langle\rangle \overset{sig}{+\!\!+} \xi = \xi$$

We have seen that the $\overset{sig}{;}$ operator adds one tuple at a time, whilst the $\overset{sig}{+\!\!+}$ operator allows us to concatenate lists together. A law which we expect to hold is that if one is appending a tuple to a signal list, then this is the same as concatenating a list with just that tuple to the signal list, and vice versa.

Lemma 3. *It is always possible to replace an application of* $\overset{sig}{;}$ *with an application of* $\overset{sig}{+\!\!+}$ *on a singleton list, and vice versa. This is represented as the following law.*

$$\xi \overset{sig}{;} (\alpha, k) = \xi \overset{sig}{+\!\!+} \langle (\alpha, k) \rangle$$

We now consider a lemma about the length operator. Something that we expect to hold is that the length of a signal list is the sum of all the lengths of the tuples.

Lemma 4. *The length of a signal list is the sum of the lengths of the segments. If* ξ *is a signal list, and* k_i *is the length of the* i^{th} *segment, then the following holds.*

$$|\xi| = \sum_{k_i \in \xi} k_i$$

Remark: One should note that by combining this lemma with the definition of $\overset{sig}{;}$ (namely the part stating that no two consecutive tuples have the same symbol), our notion of signal lists satisfies all the constraints set by Asarin et al. in Definition 2.

Another property that we expect the length operator to have is for it to distribute over the $\overset{sig}{+\!\!+}$ operator.

Lemma 5. *The length operator over signal lists distributes over the signal list concatenation operator* $\overset{sig}{+\!\!+}$*. If* ξ_1 *and* ξ_2 *are signal lists, then the following holds.* $|\xi_1 \overset{sig}{+\!\!+} \xi_2| = |\xi_1| + |\xi_2|$

This brings us to the end of our formalisation of signals in terms of signal lists. In the next section, we shall perform a similar formalisation for the notion of interpretations.

2.2.3 Interpretations

In this section we shall formalise the notion of an interpretation. These definitions will be useful when we come to give an interpretation based semantics to timed regular expressions. At that point in time, we shall want the interpretation semantics to characterise the same notions as the signal based semantics. To this end, as we define interpretations below, we should keep in mind that we are trying to use them to describe the same information held in signals.

Also recall that earlier we said that duration calculus interpretations are more powerful than signals since they can have multiple symbols active at the same time. Therefore to differ the restricted interpretations with which we will simulate signals from duration calculus interpretations, we shall refer to these restricted interpretations as *signal interpretations*.

A Partial Definition of Signal Interpretations

Before defining the concept of signal interpretations, let us consider again the notion of a duration calculus interpretation. Earlier, we had said that a duration calculus interpretation is a function which given an event and a time, will tell us whether that event is active or not at the given time. Since signal interpretations are a restricted version of duration calculus interpretations, the set of signal interpretations I must be a subset of the set of duration calculus interpretations.

Definition 10. *If i is a signal interpretation over Σ, then $i \in I$, where,*

$$I \subset \Sigma \rightarrow \mathbb{R}_0^+ \rightarrow \mathbb{B}$$

where $\mathbb{B}$ is the set of boolean values $\{0, 1\}$. So, technically, to describe a signal interpretation, we would have to define what each symbol is doing at all points in time. As an example, consider the signal a^3b^2. If we wish to express this as a signal interpretation, which we shall denote by i, we could have the following definition.

$$i(a,t) = \begin{cases} true\ for\ 0 < t \leq 3 \\ false\ otherwise \end{cases}$$

$$i(b,t) = \begin{cases} true\ for\ 3 < t \leq 5 \\ false\ otherwise \end{cases}$$

In general, we shall use the letters i,j and k to stand for signal interpretation variables. Until now, we have said that the set of signal interpretations is a subset of the set of functions of the form

$$\Sigma \rightarrow \mathbb{R}_0^+ \rightarrow \mathbb{B}$$

Therefore, we know that I is a subset of the set of duration calculus interpretations, but which subset?

Well Formed Interpretations

Having the set I equal to the set of duration calculus interpretations in the previous section is not restrictive enough to guarantee that a signal interpretation reflects the form of a signal given by Asarin et al. In fact, we can identify two problems here.

- Firstly, Asarin et al.'s signals are finite, while duration calculus interpretations do not place any constraint on the potential length of a signal.
- Secondly, in the general form of duration calculus interpretations, multiple symbols can be true at the same time, whilst in signals, these symbols occour with mutual exclusion.

We need to ensure that our signal interpretations satisfy these constraints if we intend to use them to simulate signals. To this end, we propose the following measures. Firstly, we reconcile the finititude of signals with the fact that duration calculus interpretations operate over the entire time domain, by insisting that for signal interpretations there exists

a time T (called the cutoff point), at which, and after which, none of the symbols in Σ yield true. We shall sometimes refer to this period beginning at T as *silence*. Anything which happens before the period of silence essentially corresponds to the finite signal, whilst the period of silence allows the interpretation to be infinite. In order to cater for the mutual exclusion requirement, we insist that before the time T, the following conditions are to hold, assuming α and β are elements of the alphabet Σ.

1. At any point in time, α and β cannot both be true.
2. At any point in time, some $\alpha \in \Sigma$ must be true.

Every signal interpretation will need to satisfy the above requirements. A duration calculus interpretation which subscribes to such a form is also a signal interpretation. We state this wellformedness requirement below.

Definition 11. *Let i be a duration calculus interpretation. i is said to be well formed, if it satisfies the following formula.*

$$\exists T : \mathbb{R}_0^+ .$$
$$(\forall t : \mathbb{T}, \alpha, \beta : \Sigma .\ t < T \Rightarrow (\neg(i(\alpha, t) \wedge i(\beta, t)) \wedge\ \exists \alpha : \Sigma .\ i(\alpha, t)))$$
$$\wedge\ (\forall t' : \mathbb{T} .\ t' \geq T \Rightarrow \forall \alpha : \Sigma .\ \neg i(\alpha, t'))$$

We therefore define the set of signal interpretations I as the set of wellformed (in the above sense) duration calculus interpretations. Before proposing a constructive definition of I, we ask the reader to consider two special types of signal interpretations.

Empty Signal Interpretation

The *empty signal interpretation*, is the duration calculus interpretation in which no symbol is ever active for any duration. We thus represent it by using an interpretation which attains the value of silence immediately.

Definition 12. *Let i be a signal interpretation. Then, i is said to be empty if $i = \lambda v.\lambda t.false$.*

Sometimes, instead of writing the empty interpretation as a lambda expression, we shall simply refer to it by the name $False$.

Singleton Signal Interpretations

We shall also need the concept of a singleton signal interpretation. This notion corresponds to any interpretation which has one symbol attaining the value of true for some amount of time and which attains the value of silence immediately after.

Definition 13. *Let i be a duration calculus interpretation over Σ. We say that i is a singleton interpretation if there exist $\alpha \in \Sigma$ and some $k \in \mathbb{R}^+$ such that i has the following form*

$$\begin{cases} i(\alpha, t) = \textit{true for } 0 \leq t < k \\ i(\alpha, t) = \textit{false for } k \geq t \\ i(\beta, t) = \textit{false for } t \geq 0 \end{cases}$$

where β represents all the other symbols in Σ. We refer to the set of singleton interpretations over Σ as $SING$.

2.2.4 A Constructive Definition

We can now give a constructive definition of the set I. The proposed definition of the set I is the closure of an operator over the empty signal interpretation.

Definition 14. *Let j be a singleton interpretation over Σ. Then the set of signal interpretations over Σ is defined as*

$$I = False \mid I \overset{I}{;} j$$

We still have to define the $\overset{I}{;}$ operator, which we shall call *signal interpretation chop*. This operator appends a singleton interpretation to an already existing interpretation. It can be used to construct signal interpretations one singleton interpretation at a time, starting from the empty interpretation.

Definition 15. *Let i be an interpretation and j be a singleton interpretation. Then, $i \overset{I}{;} j$ yields the interpretation defined as follows.*

$$\overset{I}{;} :: \; I \to SING \to I$$

$$\begin{aligned} i \overset{I}{;} j(\alpha, t) \;\; = \;\; & i(\alpha, t) \textit{ for } 0 \leq t < |i| \\ & j(\alpha, t - |i|) \textit{ for } |i| \leq t \end{aligned}$$

We can envisage the $\overset{I}{;}$ operation as working as in figure 2.4.
How does the above definition work? Intuitively, we want $i \overset{I}{;} j$ to behave as if it were i with j glued on after it. If i had length $|i|$ we want $i \overset{I}{;} j$ to behave like i for each time point less than $|i|$. For any value after $|i|$, we want it to behave like j. How should it behave at the point $|i|$? We shall use the convention that it should behave like j. This is due to the fact that if we append a singleton interpretation to the empty interpretation, we will not want the result to have a value of false for all symbols at the point $t = 0$.

The problem with the above approach is that j is defined from 0 to $|j|$, and does not start at $|i|$. The solution is to subtract $|i|$ units from the input time and then feed this input time to j whenever we want the value of a point which occours at or after $|i|$. As such we are not *gluing* the two interpretations together, but the definition behaves as if

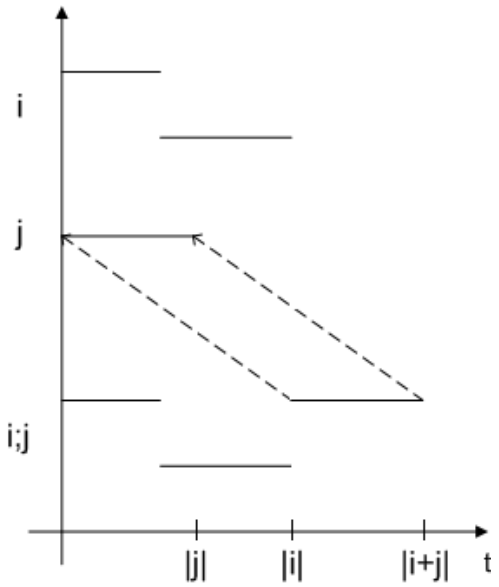


Figure 2.4: A Pictorial Representation of $\overset{\mathrm{I}}{;}$

we did so.

We also insist that in $i \overset{\mathrm{I}}{;} j$, besides j being singleton, i and j must satisfy a requirement which we shall refer to as *chop compatibility*.

Definition 16. *Let i be an interpretation, and j be a singleton interpretation. Then we say that i and j are chop compatible if, as we approach i's cutoff point T (before it achieves the value of silence), a symbol α is true, and the first symbol active for j is not α. This can be expressed as follows.*

$$\lim_{t \to T^-} i(\alpha, t) = true \wedge (\exists \beta : \Sigma.\ \beta \neq \alpha \wedge\ j(\beta, 0) = true)$$

What we mean by the above requirement is that if we are going to join two interpretations, then, we require the last active symbol of the first interpretation to be different from the first symbol of the second interpretation. When j is chop compatible with i, then the interpretation chop operation emulates the $\overset{\mathrm{sig}}{;}$ operator which does not allow the same symbol to end the first signal and start the second one.

It can be proven that the constructive definition for the set I given in Definition 14 satisfies the wellformedness requirement, and is thus an acceptable characterisation.

Theorem 6. *The constructive definition of I satisfies wellformedness.*

Remark: From this point onwards, when the word interpretation is used, one should assume that it refers to signal interpretations.

Sometimes it is useful to have a definition similar to $\overset{\mathrm{I}}{;}$ which does not need the second parameter to be a singleton interpretation. We refer to this operator as $\overset{\mathrm{I}}{+\!+}$, and call it *interpretation concatenation*.

Definition 17.

$$\overset{I}{+\!\!+} :: I \to I \to I$$
$$i \overset{I}{+\!\!+} False = i$$
$$(i \overset{I}{+\!\!+} j \overset{I}{;} k) = (i \overset{I}{+\!\!+} j) \overset{I}{;} k$$

The reader can easily note that this operation is identical in spirit to the operation of signal list concatenation. We shall now define a useful function on interpretations.

Operators on Signal Interpretations

In this section we define a function which we shall call the length of an interpretation. The intuition behind it is that it will give us the point in a signal interpretation at which silence begins. This is in fact the point before which interesting behaviour still happens.

Definition 18. *The length of an interpretation is a function* $|\ |$

$$|\ | : I \to \mathbb{R}_0^+$$

and it is defined as being the cutoff point T in the definition of wellformedness of an interpretation.

In the next section we shall state a number of lemmas for the operators defined in previous sections.

Lemmas on Signal Interpretation Operators

First of all, we note that the operator $\overset{\mathrm{I}}{+\!\!+}$ is associative.

Lemma 7. *The operator* $\overset{I}{+\!\!+}$ *is associative. If i,j and k are interpretations, then the following holds.*

$$(i \overset{I}{+\!\!+} j) \overset{I}{+\!\!+} k = i \overset{I}{+\!\!+} (j \overset{I}{+\!\!+} k)$$

It can also be shown that appending a singleton interpretation to $False$, will give the singleton interpretation.

Lemma 8. *Appending a singleton interpretation to the empty interpretation will yield back the singleton interpretation. Let j be a singleton interpretation, then the following holds.* $False \overset{I}{;} j = j$

Similarly to signal lists, the empty interpretation $False$ is also the zero of the interpretation concatenation operation $\overset{\mathrm{I}}{+\!\!+}$.

Lemma 9. *The empty interpretation is the zero of the operator* $\overset{I}{+\!\!+}$. *Let i and j be interpretations. Then, the following two laws hold.*

$$i \overset{I}{+\!\!+} False = i$$

and

$$False \overset{I}{+\!\!+} j = j$$

Recall that the $\overset{\text{I}}{;}$ operator will append a singleton interpretation to another interpretation. In this case, we can show that we can always replace $\overset{\text{I}}{;}$ with $\overset{\text{I}}{+\!\!+}$. The reverse also holds if the interpretation being concatenated via $\overset{\text{I}}{+\!\!+}$ is a singleton interpretation.

Lemma 10. *An application of $\overset{I}{;}$ can always be replaced with an application of $\overset{I}{+\!\!+}$. The opposite is also true if the second parameter of $\overset{I}{+\!\!+}$ is a singleton interpretation. Let i be an interpretation, and j a singleton interpretation, then the following holds.*

$$i \overset{I}{;} j = i \overset{I}{+\!\!+} j$$

We shall conclude this section by noting that the length of an interpretation distributes over $\overset{\text{I}}{;}$.

Lemma 11. *The length over an interpretation distributes over the operator $\overset{I}{;}$. Let i be an interpretation and j be a singleton interpretation, then the following holds.*

$$|i \overset{I}{;} j| = |i| + |j|$$

This brings us to the conclusion of our theory for signal interpretations. We shall now consider a third structure which lies somewhere in between the two notions of signal lists and signal interpretations.

2.2.5 Critical Point Lists

Introduction

So far, we have defined two notions for signals. The signal list view is a discrete representation of signals, which emphasises the lengths of each segment. On the other hand, the signal interpretation view, grants us a continuous description of signals in the form of a function. Later on, we will want to prove that these two notions are equally powerful. Before we do this, however, we shall define an additional notion called critical point lists. In later sections, it will become apparent that these critical point lists lie somewhere in between these other two notions, and that they will help us to move from the signal list model to the signal interpretation model. Although, they are not useful as a notion on their own, for now we shall take the approach of discussing critical point lists as an independent notion, without entering into the merits of why certain aspects were defined as they were. These will become apparent in section 2.3.

Definition

A critical point list is a list of tuples. The first element of each tuple is a real number, whilst the second element is a symbol. The tuples must also satisfy certain constraints. We give the formal definition below.

Definition 19. *A critical point list over an alphabet Σ, is an element of the set*

$$CPL \subset seq(\mathbb{R}_0^+ \times \Sigma)$$

obeying the following three properties:

1. $xs_i = (r_1, \alpha_1) \wedge xs_{i+1} = (r_2, \alpha_2) \Rightarrow r_1 < r_2$
2. $xs_i = (r_1, \alpha_1) \wedge xs_{i+1} = (r_2, \alpha_2) \Rightarrow \alpha_1 \neq \alpha_2$
3. $xs = \langle (0, \perp) \rangle$ *or its last element is* $(r, \perp)$*, for some positive real* r*.*

By the notation xs_i, we mean the i^{th} element of the list. Let us go over these constraints. The first constraint states that the number parts of successive tuples must be monotonically increasing. The second constraint tells us that successive tuples must have different symbols. The third constraint tells us that a critical point list is either of the form $\langle (0, \perp) \rangle$, or that it must end with a tuple containing the special symbol $\perp$, which is not part of Σ. In the work that follows, we shall use the symbols xs, ys and ws to stand as critical point list variables. We now propose a constructive definition of the set CPL, which satisfies these properties.

Definition 20. *Let* k *be a positive real and* α *be an element of an alphabet* Σ*. Then, the set of all critical point lists over* Σ *is defined as*

$$CPL = \langle (0, \perp) \rangle \mid CPL \overset{cp}{;} (0, \alpha)(k, \perp)$$

CPL is defined as the closure of the $\overset{\text{cp}}{;}$ operator over the empty critical point list $\langle (0, \perp) \rangle$.

We define the $\overset{\text{cp}}{;}$ operator as follows. In the definition below, we shall call the two tuple parameters $(0, \alpha)(k, \perp)$ a *critical point pair*.

Definition 21.

$$\overset{cp}{;} :: CPL \to (\{0\} \times \Sigma) \to (\mathbb{R}^+ \times \{\perp\}) \to CPL$$
$$\langle (0, \perp) \rangle \overset{cp}{;} (0, \alpha)(t, \perp) = \langle (0, \alpha), (t, \perp) \rangle$$
$$xs : (t_0, \beta) : (t_1, \perp) \overset{cp}{;} (0, \alpha)(t_2, \perp) = xs : (t_0, \beta) : (t_1, \alpha) : (t_1 + t_2, \perp)$$

where : is the standard list append *snoc* operator.

Remark: We note that in the recursive rule for $\overset{\text{cp}}{;}$,

$$xs : (t_0, \beta) : (t_1, \perp) \overset{\text{cp}}{;} (0, \alpha)(t_2, \perp) = xs : (t_1, \alpha) : (t_1 + t_2, \perp)$$

the tuple (t_0, β) is only present to make sure that the symbol in the critical point pair being added is not equal to the last symbol in the critical point list. In practice, the application of the rule does not require this tuple to work. Since we do not have a rule for appending a critical point pair whose symbol α is the same as the last symbol of the CPL ($\perp$ does not count as a symbol), in the proofs we can omit the tuple (t_0, β), and use the simplified rule

$$xs : (t_1, \perp) \overset{\text{cp}}{;} (0, \alpha)(t_2, \perp) = xs : (t_1, \alpha) : (t_1 + t_2, \perp)$$

instead.

It can be proven that the above definition satisfies the requirements set in the beginning of this section.

Theorem 12. *The constructive definition of CPL satisfies the requirements of Definition 19.*

Sometimes it is convenient to have an operator similar to $\overset{\text{cp}}{;}$ which does not require its left operand to be a critical point pair. We thus define the *critical point list concatenation* operator as follows.

Definition 22.

$$\overset{cp}{+\!\!+} :: CPL \to CPL \to CPL$$
$$xs \overset{cp}{+\!\!+} \langle (0, \bot) \rangle = xs$$
$$(xs \overset{cp}{+\!\!+} ys \overset{cp}{;} (0, \alpha)(t, \bot)) = (xs \overset{cp}{+\!\!+} ys) \overset{cp}{;} (0, \alpha)(t, \bot)$$

We now introduce another operator which is not related to building critical point lists.

Operators over Critical Point Lists

As with the other models we shall define the length of a critical point list. This is defined as follows.

$$|\ | :: CPL \to R_0^+$$
$$|\langle (0, \bot) \rangle| = 0$$
$$|xs : (t, \bot)| = t$$

The length of an empty critical point list is thus 0, whilst the length of a non empty one is equal to the number inside its last tuple. We now prove some lemmas relating to the operators on critical point lists.

Lemmas on Critical Point Lists

We start by showing that a property relating to $\overset{\text{cp}}{+\!\!+}$ is associativity.

Lemma 13. *The $\overset{cp}{+\!\!+}$ operator is associative. Let xs, ys and zs be critical point lists, then the following holds.*

$$(xs \overset{cp}{+\!\!+} ys) \overset{cp}{+\!\!+} zs = xs \overset{cp}{+\!\!+} (ys \overset{cp}{+\!\!+} zs)$$

Similarly to the other models, it can be shown that the empty critical point list is the zero of $\overset{\text{cp}}{+\!\!+}$.

Lemma 14. *The empty critical point list $\langle (0, \bot) \rangle$ is the zero of the operator $\overset{cp}{+\!\!+}$. Let xs be a CPL. Then the following two laws hold.*

$$xs \overset{cp}{+\!\!+} \langle (0, \bot) \rangle = xs$$

and

$$\langle (0, \bot) \rangle \overset{cp}{+\!\!+} xs = xs$$

We can also show that we can use the $\overset{\text{cp}}{+\!\!+}$ operator to replace the $\overset{\text{cp}}{;}$ operator, and vice versa when the second argument to $\overset{\text{cp}}{+\!\!+}$ consists of a list with just two tuples.

Lemma 15. *The $\overset{cp}{;}$ operator can always be replaced with a $\overset{cp}{+\!\!+}$ operator over a critical point list with two elements. Let xs be a critical point list. Then, the following holds.*

$$xs \overset{cp}{;} (0, \alpha)(t, \bot) = xs \overset{cp}{+\!\!+} \langle (0, \alpha), (t, \bot) \rangle$$

It can also be proven that the length operator has a distributivity-like property over the $\overset{\text{cp}}{;}$ operator.

Lemma 16. *The length operator over a critical point list, distributes over $\overset{cp}{;}$. Let xs be a critical point list. Then the following holds.*

$$|xs \overset{cp}{;} (0, \alpha)(k, \bot)| = |xs| + k$$

The previous lemma can be used to show that the length operator over critical point lists also distributes over $\overset{\text{cp}}{+\!\!+}$.

Lemma 17. *The length operator over critical point lists distributes over $\overset{cp}{+\!\!+}$. Let xs and ys be critical point lists. Then the following holds*

$$|xs \overset{cp}{+\!\!+} ys| = |xs| + |ys|$$

We now conclude this section by stating that if we have a critical point list, we can split it into two via the $\overset{\text{cp}}{+\!\!+}$ operator at any tuple.

Lemma 18. *A critical point list can be split into two parts at any tuple by using the operator $\overset{cp}{+\!\!+}$. Let*

$$xs = \langle (k_1, a_1), \ldots, (k_n, \bot) \rangle$$

For any tuple (k_i, a_i) there exists a critical point list zs such that

$$xs = ys : (k_i, \bot) \overset{cp}{+\!\!+} zs$$

where ys consists of all the tuples from (k_1, α_1) to (k_{i-1}, α_{i-1}).

This brings us to the end of our formal theory. We shall now make some reflections on the above concepts and conclude this part of the work.

2.2.6 Conclusion

We started this section by showing how Asarin et al. define the concept of signals. We have then formalised this notion into signal lists by introducing a set of rules which allow us to manipulate signals more formally. After we completed this task, we introduced a formal theory for a restricted form of interpretations, which we claimed that we could use to represent the same information as signals. Finally we concluded by introducing formally a third structure which we called critical point lists, remarking that they acted as the middle ground between interpretations and signals. Having a set of formal rules

for all these models will prove to be crucial when we come to prove statements about the models and how they relate. The observant reader will also notice that there are parallels between these three structures; both in the way in which they are defined, the operators which act upon them, and the properties satisfied by these operators.

We also remark here that there is an important difference between the signal list view and the interpretation view of signals. The interpretation view, is an absolute view, and indicates the symbol active at each time point. On the other hand, the signal list view is a relative view. Since each tuple specifies the length of each segment, the position of the segment is only known relative to the positions of all the segments preceding it. As a side note, we note that in order to convert from a signal list representation to a signal interpretation representation, we would need to switch from this relative view to the absolute view. This would force us to perform an extra step in order to obtain the absolute position of the segments.

In the next section we shall see issues such as these in action, as we attempt to prove that the three notions involved in this section are in fact equally powerful (i.e. we can encode the information held in one of these structures by using any of the other structures).

2.3 Moving Between Models

In the previous section, we defined three different models of time. We can envisage these models as being the three sets SIG, I and CPL.

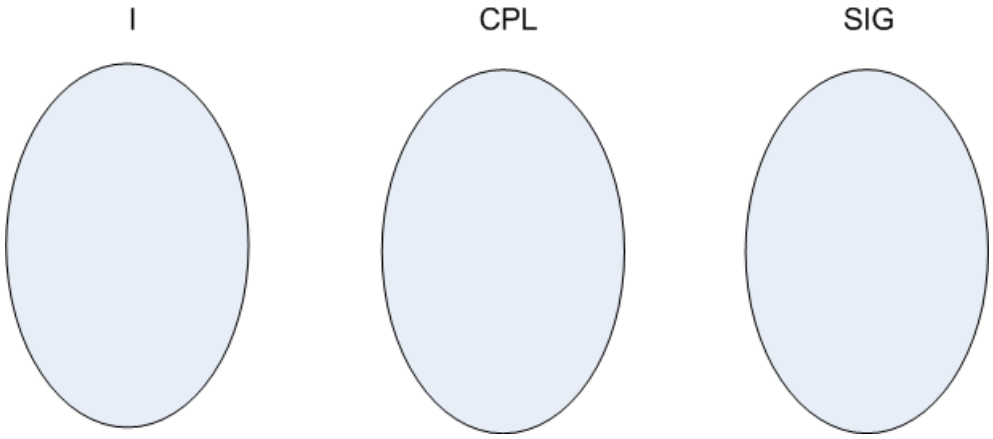


Figure 2.5: The Three Models

Later on we intend to use signal interpretations to simulate signals. In fact, we claim that using any one of these notions, we can represent the information held in any of the other notions. If we wanted to produce a proof of this claim, we would have to show that these notions are equally powerful. What does this mean? It means that for every object in one of the sets there is a corresponding object in each of the other sets. In order to do this, we will need to provide a means to move between these sets. These will take the form of functions which given an object in one model can build another object in another model.

In the previous section, we also defined the concept of critical point lists, and we claimed that this notion was somewhere in between signal lists and signal interpretations. When we defined critical point lists, it might not have been apparent that such a notion can encode the same information as a signal list or a signal interpretation. One of the first things we shall do in this section will be to show how the concept of critical point lists is closely related to that of an interpretation, and by doing so, provide the first function to move from the model of signal interpretations to the model of critical point lists, which we shall call i2c. However, before doing this, we shall introduce the concept of folds which we shall use later on in this section. The next section can be safely skipped on first reading, but has to be covered in order to fully understand the techniques employed later on in section 3.5.

2.3.1 Folds

A *fold* is a concept commonly found in functional programming languages such as Haskell. It normally takes the shape of a function which takes a data structure as an input and combines the elements of that data structure in some way in order to produce a single value. (A reference about folds is [Tho99]). Although there are various flavours of fold, what we are concerned with here is a function similar to what is called foldr (fold right)

in Haskell. (In fact foldr is more general than this, but here we only aim to provide the intuition behind folding). The definition for this function is given below.

Definition 23.

$$fold :: (\alpha \to \alpha \to \alpha) \to [\alpha] \to \alpha \to \alpha$$
$$fold \oplus (xs \overset{\alpha}{;} x)\ z = (fold \oplus xs\ z) \oplus x$$
$$fold \oplus []\ z = z$$

So, our version of fold, takes a function which takes two elements of type α and returns one element of that type (which we shall call the combining function), a list of type α (the data structure), a terminating value z, and finally returns the target value. The first rule tells us that if we call fold on the data structure xs, which has a unique way of destructing itself, $\overset{\alpha}{;}$, then the result is equal to using the combining function $\oplus$ on x and the recursive call on xs. The second case tells us what happens when the empty list is finally produced as a parameter. In this case, one simply substitutes the call with some default terminating value.

Folds can be very useful. For example, we can easily compute the sum of a list of numbers $[1, 2, 3]$ by using

$$fold + [1, 2, 3]\ 0$$

What we shall consider now is a generalisation of this concept. We shall retain the main concept behind fold which disassembles the structure (such as a list) and computes a final value from the elements. However, instead of folding into a single value, we shall use fold to create a second structure (such as a second list) of items.

In the definition below, we shall denote by $T(\alpha)$ a structure whose elements are of type α (such as a list). These structures are recursively defined via some base case which we shall call nil_α (such as the empty list), and some operator $\overset{\alpha}{;}$ which adds elements to a structure of type $T(\alpha)$, returning another structure of type $T(\alpha)$ (such as the list append operator :). We define this type of fold, in which the source structure is of type $T(\alpha)$, as $fold_\alpha$, and call it an alpha-fold.

Definition 24.

$$fold_\alpha :: (T(\beta) \to T(\beta) \to T(\beta)) \to T(\alpha) \to T(\beta) \to (\gamma \to T(\beta)) \to T(\beta)$$
$$fold_\alpha\ \oplus\ (xs \overset{\alpha}{;} x)\ nil_\beta\ f = (fold_\alpha\ \oplus xs\ nil_\beta\ f) \oplus f\ x$$
$$fold_\alpha\ \oplus\ nil_{alpha}\ \ nil_\beta\ f = nil_\beta$$

This function takes a combining function, an input structure $T(\alpha)$, a terminating value nil_β, which is, in fact the base case of $T(\beta)$, a conversion function from γ to $T(\beta)$, and returns a result of type $T(\beta)$. The function works as follows. Assuming that the input is not the base case for the input data structure, we use the conversion function to change the head of the data structure from x (of type γ) to f x (of type $T(\beta)$). Why is the head of the structure not treated as an element of type α instead of one of type γ? Suppose that the structure was a signal list. In this case, we can talk about $\xi \overset{\text{sig}}{;} (a, k)$. For this case, ξ is of type $T(\alpha)$ and (a, k) is indeed of type α. However, there might be other cases,

where the element destructed by $\overset{\alpha}{;}$ is of a different type than the elements still in the list. This occurs for example, in the case where the structure is a CPL. Consider the critical point list $xs : (t_1, \alpha) : (t_2, \bot)$. The elements are of type $(\mathbb{R}_0^+ \times \Sigma)$. However, we can use $\overset{\text{cp}}{;}$ to destruct this list into $xs : (t_1, \bot) \overset{\text{cp}}{;} (0, \alpha)(t_2 - t_1, \bot)$. Thus the type of the head of the list in this case is $(\{0\}, \Sigma) \times (\mathbb{R}^+, \{\bot\})$, which is different from the type of the elements of the list.

The conversion function is essentially converting the head of the input structure into a (normally singleton) structure of another type. This is then combined with the recursive call using the combining function, which takes two structures of type $T(\beta)$ and returns a new element of type $T(\beta)$. The base case which occurs when we end up with the empty data structure simply results in returning the terminating value nil_β of the structure $T(\beta)$. To clarify the types, one can imagine the combining function to be similar to the various list concatenation operators, such as $\overset{\text{sig}}{+\!\!+}$, and the terminating value to be their base case, which in the case of $\overset{\text{sig}}{+\!\!+}$ is $\langle\rangle$. The definition shall become more clear when we apply it later on in this section.

This concept shall be important for us since by using standard fold theorems, one can also state the following, useful result.

Theorem 19. *If $\oplus$ is associative, and $x = x_1 \overset{\alpha}{+\!\!+} x_2$ then, an alpha fold over the input structure can be split into two alpha folds as follows.*

$$fold_\alpha \ \oplus \ (x_1 \overset{\alpha}{+\!\!+} x_2) \ nil_\beta \ f = (fold_\alpha \oplus x_1 \ nil_\beta \ f) \oplus (fold_\alpha \oplus x_2 \ nil_\beta \ f)$$

where $\overset{\alpha}{+\!\!+}$ is defined as

$$\overset{\alpha}{+\!\!+} :: T(\alpha) \to T(\alpha) \to T(\alpha)$$
$$w \overset{\alpha}{+\!\!+} \ nil_\alpha = w$$
$$(w \overset{\alpha}{+\!\!+} \ y \overset{\alpha}{;} z) = (w \overset{\alpha}{+\!\!+} \ y) \overset{\alpha}{;} z$$

The result asserts that if we can divide the input data structure into two pieces, then the fold operation distributes through these components, using $\oplus$ to join the results. After this brief digression into this important concept, we can now return to our original goal of discussing the relationship between critical point lists and interpretations.

2.3.2 The Critical Points of Interpretations

So far, we have defined formally the concept of a critical point list. But why have we given these structures that name? What is a critical point? Consider an interpretation i. i changes its value at a number of points. In a way, these points are the times at which the interesting behaviour happens, since nothing changes between them. Alternatively, one could say that it seems that knowing these points is sufficient to encode the entire interpretation. We shall return to this claim later. For now, we refer to these special points, at which an interpretation changes value, as *critical points*. Amongst the critical points of an interpretation, we shall distinguish between two types, *degenerate* and *non-degenerate critical points*. A degenerate critical point is a critical point at which the

interpretation changes to the silent value. Moreover, we shall also treat t = 0, the start of an interpretation, as a critical point.
We shall use the above discussion to draw a diagram of the signal interpretation which has the symbol a active for the first 2 time units (2 excluded, due to the mechanism employed by $\overset{\text{I}}{;}$) and the symbol b active for the following 3 time units, showing an intuitive picture of all the notions we have identified thus far (figure 2.6).

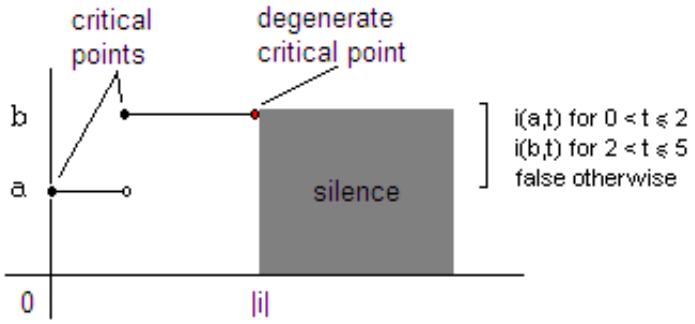


Figure 2.6: The Critical Points of an Interpretation

However, also note that so far, we have not proven any results which would enable us to conclude that the critical points of an interpretation should be distributed as they have been depicted above. The diagram should be taken in the spirit of suggesting the path in which we should proceed when formalising the notion of critical points, and of suggesting the propositions we need to prove later on. We can now proceed to formally define the notion of critical points. But how should we characterise the set of critical points of an interpretation? For this, we define the operator C(i) which will return to us the set of critical points of an interpretation i.

Definition 25. *Let i be an interpretation. Then,*

$$C :: I \to \mathbb{P}(\mathbb{R}_0^+)$$
$$C(i) = \{x \mid \lim_{t \to x^-} i(\alpha, t) = true \wedge i(\alpha, x) = false\} \cup \{0\}$$

The above set comprehension will yield those points of an interpretation at which a jump discontinuity occurs. Sometimes, besides knowing the position of the critical points, we also need to know the symbol associated with each critical point. We shall also associate the special symbol $\perp$ to the critical point which changes the interpretation to the value of silence. We denote the function giving us the points and their associated symbols as $\overline{C}$.

Definition 26. *Let i be an interpretation. Then,*

$$\overline{C} :: I \to \mathbb{P}(\mathbb{R}_0^+ \times \Sigma)$$
$$\overline{C(i)} = \{(x, \alpha) \mid x \in C(i) \wedge \exists \alpha : \Sigma.\ i(\alpha, x)\} \cup$$
$$\{(x, \perp) \mid x \in C(i) \wedge \forall t : \mathbb{T}, \alpha : \Sigma.\ t \geq x \Rightarrow \neg i(\alpha, t)\}$$

The first set comprehension associates the symbol which happens to be active at a particular critical point with that point. The second set comprehension associates the symbol $\perp$ to the critical point at which and after which no symbol is active. It is immediately obvious that $C(i)$ is simply the projection of $\overline{C(i)}$ over the first element of its tuples. We now turn our attention to degenerate critical points.

Definition 27. *Let x be a critical point of the interpretation i. We call x a degenerate critical point if it is followed by silence, that is no symbol is true at and after this point. This can be formulated as*

$$degenerate(x, i) \triangleq \forall t : \mathbb{T}, \alpha : \Sigma.\ t \geq x \Rightarrow i(\alpha, t) = false$$

As a consequence of the above, we note that if a critical point is degenerate, by our definition of $\overline{C(i)}$, its associated symbol is $\perp$ and vice versa.

Proposition 20. *x is a degenerate critical point of an interpretation i, if and only if it is associated with the symbol $\perp$, and vice versa.*

$$degenerate(x, i) \Leftrightarrow (x, \perp) \in \overline{C(i)}$$

We now know what critical points are. The next step shall be to prove a set of propositions which shall give us more information about how these points are distributed.

Propositions About Critical Points

In this section we prove several propositions which give us more information about critical points. These ideas will lead us to the conclusion that there is a clear link between the set of critical points given by C() and the concept of critical point lists. In reality, all the propositions we prove in this section are in fact theorems about the operator C().

Proposition 21. *Every well formed signal interpretation i has one and only one degenerate critical point. Moreover, this point is the same point as the interpretation's cutoff point T. Consequently, the position of the degenerate critical point is equal to the length of the interpretation.*

The following corollary then follows immediately.

Corollary 22. *If i is an interpretation, the point $(|i|, \perp)$ is in $\overline{C(i)}$, that is, i's degenerate critical point occurs at the value which is equal to its length.*

We can also prove that the degenerate critical point is the largest point amongst all the other critical points.

Proposition 23. *Let i be a well formed signal interpretation, and C(i) be the set of its critical points. If x is i's degenerate critical point, then x is its largest critical point. This can be expressed as*

$$degenerate(x, i) \Rightarrow x = \max\{C(i)\}$$

We are now in a position to assert the link between the critical points of an interpretation and the notion of critical point lists. We shall do this by noting that it is possible to order the set of critical points of an interpretation i. This essentially yields a sequence (list) of points. Under the correct order, we shall see that the result is in fact a critical point list.

The Critical Point Lists of Interpretations

Proposition 24. *Let i be an interpretation. Then $\overline{C(i)}$ can be ordered, and the resulting sequence is a Critical Point List.*

We have thus established the first link between interpretations and critical point lists. We use the result to construct a function which we call $i2c$, which given an interpretation yields a critical point list.

Definition 28. *Let i be an interpretation. Then, we refer to $i2c(i)$ as the Critical Point List of i.*

$$i2c :: I \to CPL$$

It is achieved by ordering $\overline{C(i)}$ under the relation $\geq$.

Thus, $i2c$ is a function, which given an interpretation, will find its critical points, order them and return the resulting critical point list. Thus we can say that through $i2c$ we know how to find a critical point list for every interpretation. Now, since $i2c$ relies on the points of C(i) to work, all results about C(i) will give us corollaries about $i2c$. These will be useful because they will act as rules telling us the behaviour of $i2c$ in response to inputs of a certain form. We state the first corollary below, which follows from the Propositions that an interpretation has its degenerate point at $|i|$ (Proposition 21) and the fact that a degenerate point is the maximum point of C(i) (Proposition 23).

Corollary 25. *Let i be an interpretation. Then, $i2c(i)$ has the form $xs : (|i|, \bot)$*

We can also prove a proposition about the critical points of empty signal interpretations.

Proposition 26. *If a signal is empty, it only has one critical point at $t = 0$, and this is degenerate. Hence the length of the empty interpretation is 0.*

The above proposition has an immediate corollary which tells us the output of $i2c$ when its input is an empty interpretation.

Corollary 27. *Invoking $i2c$ over the empty interpretation will yield the empty critical point list.*

$$i = False \Rightarrow i2c(i) = \langle (0, \bot) \rangle$$

It can also be stated that when a signal is not empty, it must have at least two critical points.

Proposition 28. *If an interpretation is not empty, then it has at least two critical points.*

So far we know that an empty interpretation has only one critical point and that any interpretation which is not empty will have two or more. Let us now focus our attention to what happens *between* the critical points. It can be proven that there can be no change in the value of a signal between two critical points.

Proposition 29. *There is no change in the value of an interpretation between two consecutive critical points. Let i be an interpretation, and $i2c(i)$ be its critical point list. Then, for any two consecutive critical points c_i and c_{i+1} in $i2c(i)$, if $i(\alpha, t)$ is true, there is no point t in $[c_i, c_{i+1})$ at which $i(\alpha, t) = false$. Similarly, If $i(\alpha, t)$ is false, there is no point t in $[c_i, c_{i+1})$ at which $i(\alpha, t) = true$.*

Earlier, we introduced the notion of a singleton interpretation as a special form of an interpretation. We can refine the result about the number of critical points, when we are dealing with singleton interpretations.

Proposition 30. *Let i be a singleton interpretation. Then, i has only two critical points.*

The corollary for $i2c()$, when its input is a singleton interpretation follows.

Corollary 31. *Let i be a singleton interpretation over an alphabet Σ, and with length k. Then there exists some $\alpha \in \Sigma$ such that the result of invoking i2c on it is characterised by the following law.*

$$i2c(i) = \langle (0, \alpha), (k, \perp) \rangle$$

It is easy to see why this happens; First of all, a singleton interpretation is not empty, so there must be 2 critical points. We also know that 0 is always a critical point, and that there is a critical point whose position is equal to the length of the interpretation. Finally, we also know that the last critical point (at k) has to be associated with $\perp$ and that the value of the interpretation at 0 is α. By putting all this together, we can conclude the form that the critical point list of a singleton interpretation will have. We are now finally in a position to explain why the $\overset{\text{cp}}{;}$ operator was defined in the way it was.

The operator $\overset{\text{cp}}{;}$ explained

The last proposition we considered told us that if we started with a singleton interpretation, and took its critical point list, we would end up with a critical point list of two elements, namely $\langle (0, \alpha), (k, \perp) \rangle$. Now, as we saw in Definition 14, an interpretation is constructed by appending singleton segments together via the $\overset{\text{I}}{;}$ operator. Obviously, given the close relationship between interpretations and their critical points, we would have wanted critical point lists to be constructed in a similar way via the $\overset{\text{cp}}{;}$ operator. It is perhaps now clear what the critical point pair parameter in the function $\overset{\text{cp}}{;}$ is. Essentially, it represents the two critical points of a singleton interpretation, in this case, $(0, \alpha)$ and $(k, \perp)$.

What is the intuition behind its operation? Consider merging two interpretations i and j via $\overset{\text{I}}{;}$, of which j is singleton. What happens to their critical points? The degenerate critical point of i is not degenerate anymore, but acquires the symbol of the latter interpretation. However, the first critical point of j is then merged into this critical point. What happens to the last critical point of j? It is moved, by the length of the first interpretation. This is exactly the process performed by the $\overset{\text{cp}}{;}$ operator.

How can we show that the $\overset{\text{cp}}{;}$ operator correctly emulates the $\overset{\text{I}}{;}$ operator? One useful property that it should have is the following.

Lemma 32. *The function i2c has a distributive-like property over the* $\overset{I}{;}$ *operator. Let i be an interpretation, and j be a singleton interpretation with symbol* α *and length k. Then,*

$$i2c(i) \overset{cp}{;} (0, \alpha)(k, \perp) = i2c(i \overset{I}{;} j)$$

This means, that if we take the two interpretations, merge them via $\overset{\text{I}}{;}$, and transform them into a critical point list, then we should get the same critical point list as if we had first transformed i and applied $\overset{\text{cp}}{;}$ to it and the critical point pair of j.

We shall now consider a number of other properties which are possessed by i2c.

Properties of i2c

So far we know that an interpretation has a number of critical points, and we have provided a function which given an interpretation, will give us a critical point list.

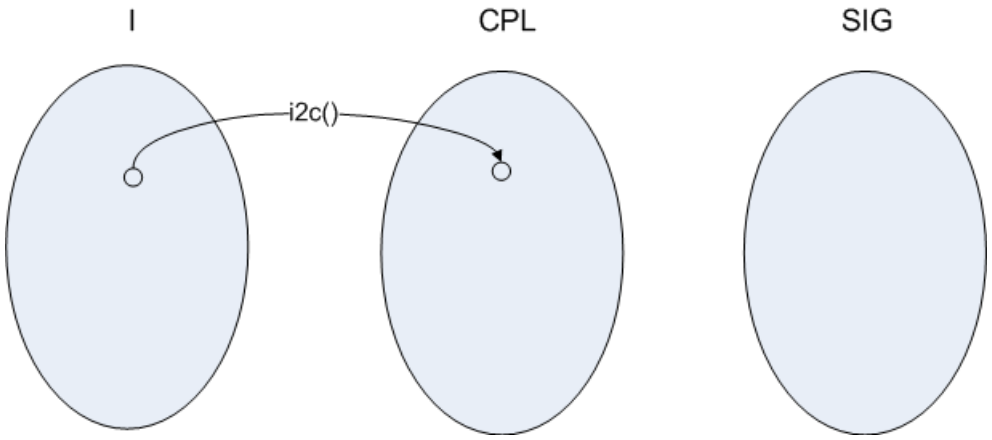


Figure 2.7: The Function i2c

We must be careful here not to assume that properties possessed by the interpretation, are preserved in the critical point list, even though intuitively the notions are analogous. If we wish to show that some property is indeed preserved by the function, this must be proved. A reason for these precautions is that we have no guarantee that we have defined the function in the desired way, for example.

By providing the construction $i2c$, we have shown that for every interpretation we can construct a critical point list. We still do not know, however whether we can get back the original interpretation from this critical point list, or put in another way, whether a critical point list has enough information to characterise an interpretation. We also have no guarantee that $i2c$ is injective, that is, whether it grants a unique critical point list to each interpretation.

But we are digressing. In this section we shall want to prove some properties which hold when we call $i2c$ to move from one domain to another. One particular property which is worthwhile to investigate is how $i2c$ relates to the length function. Essentially we shall see that if an interpretation has a certain value under the length operator on interpretations, then its critical point list will also have the same value under the length of a critical point list.

Lemma 33. *The length of an interpretation is equal to the length of the CPL obtained when i2c is invoked on that interpretation. Let i be an interpretation, then the following holds.*

$$|i2c(i)| = |i|$$

We shall now attempt to give a function which given a critical point list, will give us an interpretation.

2.3.3 From Critical Point Lists to Interpretations

So far, we have been able to move from the domain of interpretations to that of critical point lists. We now consider the function $c2i$, which allows us to move from the domain of critical point lists to that of interpretations. We depict this in figure 2.8.

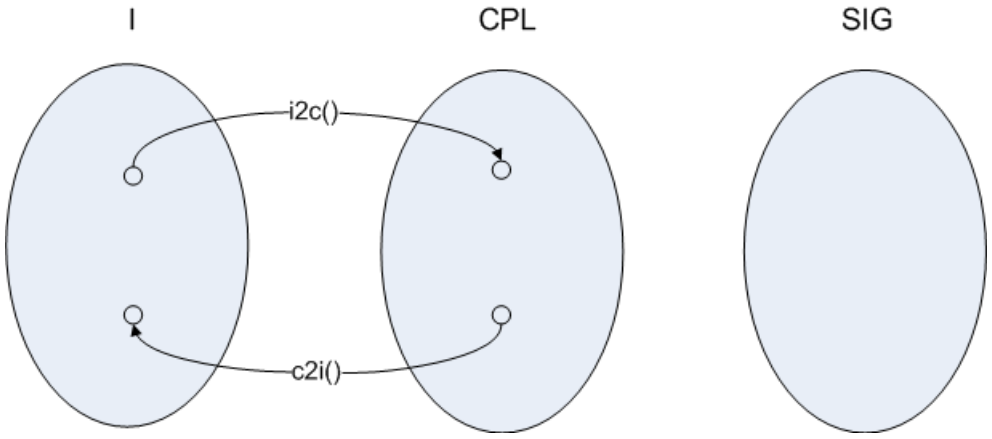


Figure 2.8: The Functions i2c and c2i

Note that we make no claim on the relationship of $i2c$ and $c2i$ yet. It is entirely possible that starting with an interpretation, and applying $i2c$, we will get a critical point list which, when we apply $c2i$ to it, will not give us the starting interpretation. Obviously, this is a situation which we want to avoid; however, we shall have to check for this fact later on.

How should we define $c2i$? The idea will be to examine successive pairs of critical points. Since we know that an interpretation's value between critical points does not change, we will introduce a singleton interpretation equal in length to the distance between the critical points, and append it, via the $\overset{I}{;}$ operator to the result of recursively calling $c2i$ on the rest of the critical point list.

Definition 29.

$$c2i :: CPL \to I$$

$$c2i(\langle(0, \bot)\rangle) = False$$

$$c2i(xs : (t_1, \alpha) : (t_2, \bot)) = c2i(xs : (t_1, \bot)) \overset{I}{;} \begin{cases} i(\alpha, t) = true\ for\ 0 \leq t < t_2 - t_1 \\ i(\alpha, t) = false\ for\ t \geq t_2 - t_1 \\ i(\beta, t) = false\ for\ t \geq 0 \end{cases}$$

The base case is straightforward. It will transform the empty critical point list into the empty interpretation. The second rule, will reduce the input critical point list by one critical point and will use the times t_1 and t_2 of the last 2 critical points to build a new interpretation starting at 0 and ending at $t_2 - t_1$. It then uses the $\overset{\text{I}}{;}$ operator to append this interpretation to the result of recursively calling the function on the remainder of the critical point list. As with $i2c$, we shall now see some properties satisfied by $c2i$.

Properties of c2i

In this section we show a number of properties enjoyed by $c2i$. We start by showing that $c2i()$ can be expressed in terms of an alpha-fold. In this case, the input is a critical point list with destructor $\overset{\text{cp}}{;}$. The function will disassemble the critical point list by extracting critical point pairs from it. To these pairs, it applies the conversion function f, which calculates a corresponding interpretation which can be concatenated via $\overset{\text{I}}{+\!\!+}$ to the rest of the recursive call.

Proposition 34. *The function c2i can be expressed in terms of alpha-fold.*

$$c2i(xs) = fold_{CPL} \ \overset{I}{+\!\!+} \ xs \ \ False \ \ f$$

where

$$f :: (0 \times \Sigma) \to (\mathbb{R} \times \bot) \to I$$

$$f(0, \alpha)(t_1, \bot) = \begin{cases} i(\alpha, t) = \textit{true for } 0 \leq t < t_1 \\ i(\alpha, t) = \textit{false for } t \geq t_1 \\ i(\beta, t) = \textit{false for } t \geq 0 \end{cases}$$

Note how the conversion function f takes the critical pair at the head of the critical point list as input. In this case, the input parameter of f is a pair of tuples. However, when defining alpha folds, f had only one object of some type γ as its input. In this case however, we know that we can use the technique of currying, which enables us to compress a number of arguments n into just one by treating them as a single n-tuple whose elements are the n arguments.

We also state the fact that $c2i$, when moving from the domain of critical points to that of interpretations, in a way, preserves the length of the critical point list. By this we mean that the length of the original critical point list will be equal to the length of the resulting interpretation, even though the length functions work on parameters of different types.

Lemma 35. *The length of a critical point list is equal to the length of the interpretation returned by the invocation of c2i on that critical point list. This can be expressed as*

$$|xs| = |c2i(xs)|$$

Many times it is useful to know that a function terminates in a finite number of recursive calls. In fact, $c2i$ satisfies this property, as long as its input list is finite.

Proposition 36. *Let xs be a finite critical point list. Then, $c2i(xs)$ terminates.*

With these properties settled, we shall now perform a sanity check to ensure that $c2i$ and $i2c$ are consistent with one another. This will increase our confidence in the fact that we have correctly defined the functions which allow us to move between the models of interpretations and critical point lists.

2.3.4 Sanity of Constructions c2i and i2c

So far, we have described two procedures. Given an interpretation, the function $i2c$ gives us a corresponding critical point list. The second procedure, $c2i$, given a critical point list, will return an interpretation. Earlier we said that we were not sure that the functions were defined in such a way that if we moved from the domain of interpretations to the domain of critical point lists and back to the domain of interpretations we would end up with the interpretation we started with. The same holds for when we start with a critical point list.

How can we prove that this property holds? What we would like is for the functions $i2c$ and $c2i$ to be inverses of one another. If this was true, then if one started with an interpretation, applied $i2c$, getting a critical point list, and then applied $c2i$ to the result, one would get exactly the same interpretation one started with. The same holds if we start with a critical point list instead of an interpretation. We can envisage this desirable situation in figure 2.9.

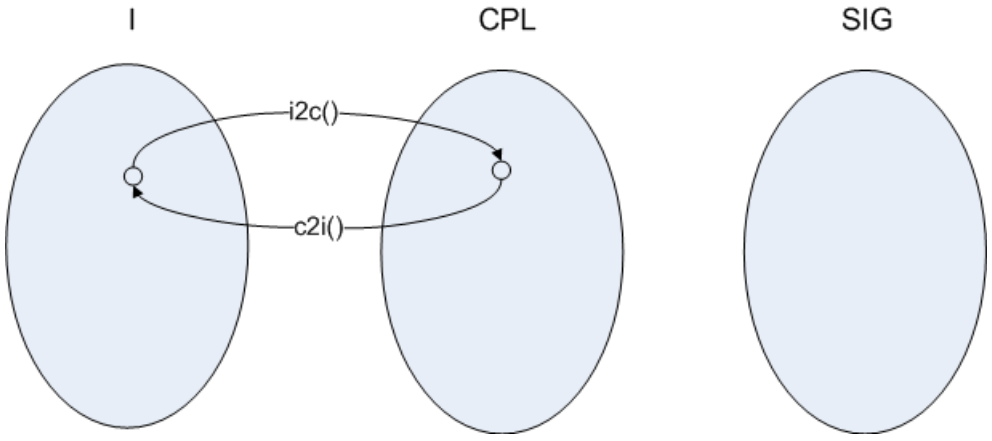


Figure 2.9: i2c and c2i are Inverses

Proving that the two functions are inverses, also guarantees that they are bijections. This is very useful since it makes sure that critical point lists and interpretations map to one another in a one to one fashion. This guarantees that the two models are in fact equally powerful, and that we can encode a critical point list as an interpretation, and an interpretation as a critical point list.

How can we prove that two functions are inverses of one another? One way of proving such a fact is to show that their composition yields the identity function. Since function composition is not necessarily commutative, we have to prove that both composing $i2c$ with $c2i$ and composing $c2i$ with $i2c$ yield the identity function. Let us start with the first case.

Theorem 37. *Composing i2c with c2i yields the identity function. Let xs be a critical point list, then the following law holds.*

$$i2c(c2i(xs)) = xs$$

One can also prove the second direction.

Theorem 38. *Composing c2i with i2c yields the identity function. Let i be an interpretation, then the following law holds*

$$c2i(i2c(i)) = i$$

The above two theorems are very important, since they give us the confidence that our definitions are correct with respect to what we want to achieve. We now know that we can move from interpretations to critical point lists and vice versa, knowing that we can always return to the original entity. This has an important repercussion. If we can prove that one of these notions is equivalent to some third notion, then, both will be equivalent to the third notion. We now consider such a third notion, namely signal lists and how it can be incorporated into our growing framework.

2.3.5 Signal Lists and Critical Point Lists

What we shall do in the following sections is somewhat similar to what we have done previously. We shall show how one is able to move from the model of critical point lists CPL to the model of signal lists SIG, via a construction $c2s$ and vice versa via a construction $s2c$. Moreover, we shall also prove that they are inverses, allowing us to move from one particular point in one domain to the other, and from the other point back to the original one.

The net effect is that we would have shown that signal lists and critical point lists are equally powerful. By consequence, we would also know that critical point lists and signal interpretations are equally powerful. From these two statements we would be able to conclude that all the three models are thus equal in power. By applying the functions $s2c$ and $c2i$ in succession, we would be able to move from a signal list to a unique interpretation. If we want to move in the reverse direction, we can do so by invoking in sequence the functions $i2c$ and $c2s$. This is shown in figure 2.10.

It is now perhaps beginning to become clear why we have taken the trouble to define critical point lists. Essentially, signal lists are a discrete notion, whilst interpretations are a continous one. It would have proven to be relatively hard to translate directly from one notion to the other. However, we were lucky in this case to note that an interpretation's critical points are sufficient to characterise an interpretation. Thus we were able to discretise, in a a relatively straightforward way the notion of an interpretation. Our problem thus becomes easier since it is simpler to prove equivalence between the two discrete notions (of signal lists and critical point lists). We shall now consider the two constructions $s2c$ and $c2s$.

2.3.6 From Critical Point Lists to Signal Lists

In this section we give a construction for converting a critical point list into a signal list. This is done through the function $c2s$.

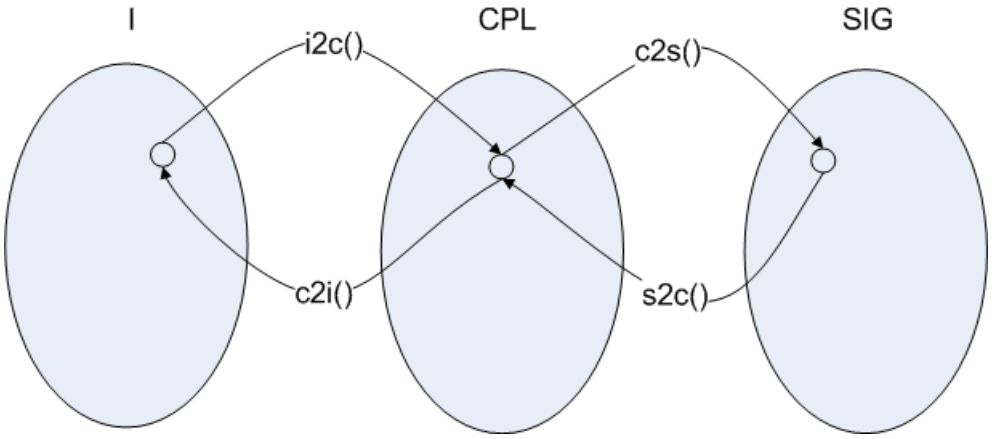


Figure 2.10: Functions Allowing One to Move Between Domains

Definition 30.

$$c2s :: CPL \to SIG$$

$$c2s(\langle(0, \bot)\rangle) = \langle\rangle$$

$$c2s(xs : (t_1, \alpha) : (t_2, \bot)) = c2s(xs : (t_1, \bot)) \overset{sig}{;} (\alpha, t_2 - t_1)$$

We first start with the base case. When c2s is passed an empty critical point list, it converts it into an empty signal list. For the recursive case, the idea is to detect the times of successive critical points, starting from the end of the critical point list. For each pair, we calculate the distance between these points to get the length of the signal list segment to be added to the output. The function then invokes itself again on the input, but with the last critical point removed.

In the next section we shall see some properties satisfied by c2s.

Properties of c2s

Firstly, it can be shown that $c2s$ has an alternative definition in terms of an alpha-fold. In this case, the input list is a critical point list whose destructor is $\overset{cp}{;}$. The function will disassemble the critical point list through this destructor into critical point pairs. It then invokes the conversion function f to construct a signal list of one tuple from these pairs. The way the new tuple is obtained is in fact the same method used by the original $c2s$ function. The list of one tuple is then concatenated with the result of the remaining computation.

Proposition 39. *The function $c2s$ can be expressed in terms of alpha-fold.*

$$c2s(xs) = fold_{CPL} \overset{sig}{+\!\!+} xs \langle\rangle f$$

where

$$f :: (0 \times \Sigma) \to (\mathbb{R} \times \bot) \to SIG$$

$$f(0, \alpha)(t, \bot) = \langle(\alpha, t)\rangle$$

A property which is enjoyed by $c2s$ is that if we have a function call of $c2s$ on a critical point list xs, then the length of the signal list resulting from that function call is equivalent to the length of the critical point list.

Lemma 40. *The length of a critical point list is equal to the length of the signal obtained by invoking c2s on it. Let xs be a critical point list. Then the following holds*

$$|c2s(xs)| = |xs|$$

We now consider the opposite construction, $s2c$.

2.3.7 From Signal Lists to Critical Point Lists

In this section, we tackle the opposite problem, and give a construction for converting a signal list to a critical point list. This is done through the function $s2c$, which we define below.

Definition 31.

$$s2c :: SIG \to CPL$$
$$s2c(\langle\rangle) = \langle(0, \bot)\rangle$$
$$s2c(\xi \overset{sig}{;} (k, \alpha)) = s2c(\xi) \overset{cp}{;} (0, \alpha)(k, \bot)$$

The function works as follows. Calling $s2c$ on an empty signal list will return the critical point list with the degenerate point at 0. Otherwise, this function will convert the last segment of the signal list to a critical point pair and use the $\overset{\text{cp}}{;}$ operator to append it to the result of applying s2c recursively on the rest of the signal. We now show some properties enjoyed by $s2c$.

Properties of s2c

In this section we enumerate some properties satisfied by $s2c$. We start by stating that it is possible to write down $s2c$ in terms of alpha-fold. In this alternative definition, the input data structure is a signal list with destructor $\overset{\text{sig}}{;}$. The function will disassemble this list segment by segment, and will convert each segment using the function f to a critical point list with two elements (using exactly the same technique applied by the original definition of $s2c$ to convert the segment into a critical point pair). The function then appends this list via the combining function $\overset{\text{cp}}{;}$ to the result of the remaining computation.

Proposition 41. *The function s2c can be expressed in terms of alpha-fold.*

$$s2c(\xi) = fold_{SIG} \overset{cp}{+\!\!+} \xi \langle(0, \bot)\rangle f$$

where

$$f :: SIG \to CPL$$
$$f\langle(\alpha, k)\rangle = \langle(0, \alpha), (k, \bot)\rangle$$

Similarly to $c2s$, if we start with a signal list, and apply $s2c$, the resulting critical point list has the same length as the signal list.

Lemma 42. *The length of a signal is equal to the length of the critical point list obtained by invoking s2c on it. Let ξ be a signal list. Then, the following holds.*

$$|s2c(\xi)| = |\xi|$$

$s2c$ can also be proven to terminate, assuming that its input is a finite signal list.

Proposition 43. *Let ξ be a finite signal list. Then $s2c(\xi)$ terminates.*

This brings us to the conclusion of our discussion for $s2c$. In the next section we shall prove that c2s and s2c are in fact inverses of one another.

2.3.8 Sanity of the Constructions c2s and s2c

As we have done before, we shall now check for the sanity of these constructions by proving that they are in fact inverses of each other. We prove this by using the same technique as before, that is by proving that composing $s2c$ and $c2s$ in either order will give back the identity function. We start with the first direction.

Theorem 44. *Composing s2c with c2s yields the identity function. Let xs be a critical point list. Then the following law holds.*

$$s2c(c2s(xs)) = xs$$

The second direction can also be proven to hold.

Theorem 45. *Composing c2s with s2c yields the identity function. Let ξ be a signal list. Then the following law holds*

$$c2s(s2c(\xi)) = \xi$$

With these theorems, we confirm that for every critical point list there is one signal list, and vice versa, and that we can move between the models freely whilst being sure that we can always get back to where we started. In the next section we include some concluding thoughts.

2.3.9 Conclusion

The objective for this section was to relate the three different models of signal lists, critical point lists and signal interpretations. We did so by defining functions which, given a signal list or a signal interpretation enabled us to create a critical point list and vice versa. By providing proofs that the functions for moving from one of these notions to critical point lists were inverses, we made sure that every signal list and interpretation has exactely one associated critical point list and vice versa. Such a proof assures us that signal lists and interpretations are exactly as powerful as critical point lists, and as a consequence, we therefore also know that signal lists are as powerful as signal interpretations. Due to this fact, we can use any one of these notions to encode any of the others, and we are able to retrieve the encoded information back through the constructions defined in this section.

The reader may note that the function $i2c$ is different from the other constructions. Whilst in the other constructions we had the luxury to start with discrete elements which

could be manipulated by Haskell like functions, in $i2c$, we start with a continous notion which cannot be manipulated as easily as the other structures. In fact in this case, what we had to do was to define the operator C(), which used limits to extract the set of critical points from an interpretation. Also, to determine the exact behaviour of $i2c$, we could not give any precise rules, as we did for the other constructions. What we had to do was to prove theorems showing how it behaved in response to certain inputs. That said, we shall see, later on, that we have enough knowledge about $i2c$ from these theorems in order to prove interesting results.

In this chapter we also introduced the notion of alpha-folds and proved that our functions (excluding $i2c$) can be written in terms of these expressions. Later on, this shall prove to be useful. By combining the fact that all our concatenation operators are associative with the fact that we have expressions for these functions in terms of alpha-folds, we shall be able to operate on any part of an input structure, rather than having to process all the input structure recursively.

This concludes this part of our work. In the last section for this chapter, we summarise what we have achieved and hint at what we shall be doing next.

2.4 Conclusion

In this chapter, we have defined rigorously a formal theory for both signals and interpretations. These rules shall allow us to reason unambiguously about these structures, with the advantage that we will not have to stop and reflect on whether an particular operation is permitted or not. We have also introduced a means of relating the signal list model with the signal interpretation model. This took the form of a number of constructions which, for every signal list assured the existence of one and only one signal interpretation, and for each signal interpretation yielded one particular signal list. Moreover, these constructions are inverses, so we can always encode a signal list into an interpretation *and* decode it back whenever necessary (the reverse also applies).

Whilst we were proving these results we also found it convenient to introduce a third structure, which we called critical point lists. As we saw, these objects lie somewhere in between the continous description of interpretations and the discrete representation offered by signal lists. Due to this fact, it proved simpler to decompose the task of proving that signals can be converted into interpretation (and vice versa) by showing that a notion can first be converted into a critical point list and then to the target notion.

Now that the mathematical groundwork has been laid, we can proceed to tackle the issues we discussed in our introduction. In the next chapter we rise to the next level; from signals and interpretations to the timed regular expressions which characterise them. One of the main aims shall be to introduce the original semantics for timed regular expressions and to give them an alternative semantics based on interpretations. Following this, these two semantics will be proved to be sound and complete.

Chapter 3

Semantics for Timed Regular Expressions

3.1 Introduction

In the previous chapter, the groundwork was laid for the important parts of this work. In this chapter we aim to provide all the prerequisites for proving slowdown and speedup results (for TRE) using the approach we called task $2'$. It might be helpful to state again what this task involved. We had said that one way of applying the theory of slowdown and speedup for logics without an interpretation based semantics was the following. We would first endow that logic with an interpretation based semantics, and then, we would prove that these interpretation based semantics were sound and complete with respect to the old ones. Once these two pieces of work were completed, one could then apply the theory of slowdown and speedup to that logic (with the interpretation semantics), whilst being guaranteed that any results, would also hold for the logic with the original semantics.

Before embarking on achieving these two goals, we will start by stating how Asarin et al. define the TRE formalism. This can then be used as a guide for giving TRE an alternative semantics which simulates the original one in terms of signal interpretations.

3.2 Asarin's Timed Regular Expression Semantics

We begin this chapter by defining syntactically and semantically the logic of timed regular expressions. A timed regular expression can be viewed as being a pattern of computation, which includes a number of different system traces. In this section, we concentrate on the definition given by Asarin et al. [ACM97], which uses signals as an underlying model. We shall start by examining the syntax for this formalism. In doing so, we shall see that timed regular expressions bear a strong resemblance to standard regular expressions.

First of all, a timed regular expression operates over a finite set of symbols Σ.The simplest timed regular expression is the expression

$$a$$

where a can be any symbol in Σ. We shall call such an expression a base case for timed regular expressions. From, such expressions, more complex expressions can be derived recursively, using the other operators in the logic.

If we were to define the legal forms of a timed regular expression, we would claim that a timed regular expression must either be a base case expression, or it must have one of the following forms.

$$\psi_1 \vee \psi_2$$
$$\psi_1 \wedge \psi_2$$
$$\psi_1.\psi_2$$
$$\psi^*$$
$$\langle\psi\rangle_I$$

where ψ_1 and ψ_2 are also timed regular expressions and I is an integer bounded interval. One should note that syntactically, the difference between regular expressions and timed regular expressions lies in the interval restrict ($\langle\psi\rangle_I$) and intersection operators. The interval restrict operator allows us impose time constraints on the duration of the symbols which make up the signals. The intersection operator, on the other hand, is necessary to be able to formulate certain types of constraint (more on this later).

So far we have just mentioned the syntax behind the logic. We shall now show the semantics given by Asarin et al. to timed regular expressions.

3.2.1 The Semantics

When giving a semantics to a logic, we must associate a mathematical object with each syntactic construct. The mathematical objects we shall use in this case will be sets of signals. Let us start with the base case timed regular expression. We shall use the operator $|[]|$ to denote the signal semantics of a timed regular expression.

When the expression is a symbol from Σ, this yields the set of all the signals made up from that symbol, with any possible duration. We can define this as follows.

$$|[a]| = \{a^r \mid r \in \mathbb{R}^+\}$$

For example, the signals b^2 and $b^{3.142}$ are in the set characterised by the timed regular expression b.

Earlier we said that a timed regular expression can be defined recursively, by applying an operator to one or more valid timed regular expressions. In this case, the semantics are obtained by resolving the internal regular expressions (which will give us one or more sets) and then by applying some set operation on these sets (depending on the operator). Let us thus define the recursive cases.

The concatenation of two timed regular expressions yields a set of signals in which the elements are formed by performing signal set concatenation on the sets yielded by the individual timed regular expressions.

$$|[\psi_1 \circ \psi_2]| = |[\psi_1]| \circ |[\psi_2]|$$

The signal set concatenation of ψ_1 and ψ_2 can be defined as follows.

$$|[\psi_1]| \circ |[\psi_2]| = \{\zeta_1 \circ \zeta_2 \mid \zeta_1 \in |[\psi_1]| \wedge \zeta_2 \in |[\psi_2]|\}$$

For example, the timed regular expression $a \circ b$, gives all the signals which are formed from the symbol a followed by the symbol b, and for which a and b can have any duration. For example, a^2b^3 would be in such a set.

The union of two timed regular expressions gives us the union of the sets of signals generated by the individual regular expressions.

$$|[\psi_1 \vee \psi_2]| = |[\psi_1]| \cup |[\psi_2]|$$

The kleene star of a regular expression, generates the sets of signals characterised by ψ^1,ψ^2... and performs their union. The exponentiation operation simply indicates timed regular expression concatenation for i times.

$$|[\psi^*]| = \bigcup_{i=0}^{\infty} |[\psi^i]|$$

For example $a^2b^3a^1b^2$ is in $(a \circ b)^*$, because it is in $(a \circ b)^2$.

These operations should be reminiscent of standard regular expressions. Even the regular expression a, is not that different since it essentially ignores the effect of time by allowing symbols to take any value as their length.

We now allow timing constraints to occour by defining the semantics of the interval restrict operator. If I is an integer bounded interval,

$$|[\langle\psi\rangle_I]| = |[\psi]| \cap \{\xi : Signal \mid |\xi| \in I\}$$

The above definition is saying that to obtain the set of signals generated by $\langle\psi\rangle_I$, we generate those strings in ψ not taking care of any time constraint, and then, we accept only those strings whose length falls in the interval I.

For example, the timed regular expression $\langle a\rangle_{[1,3]}$ generates the set of signals $\{a^r \mid 1 \leq r \leq 3\}$.

The other interesting operator not normally found in normal regular expressions is the intersection operator.

$$|[\psi_1 \wedge \psi_2]| = |[\psi_1]| \cap |[\psi_2]|$$

This operator can allow time constraints to hold between expressions which are in separate interval restriction braces. For example, suppose we want a timed regular expression which characterises the following signal family

$$a^{(1-x)}b^x c^{(1-x)}$$

where x is between 0 and 1. The above definition is instructing us to find a timed regular expression with three constraints. The first is that the length of a and b together is equal to 1, which can be expressed as $\langle a \circ b \rangle_{[1,1]}$. The second constraint insists that the length of b and c also sums to 1, which can be expressed as $\langle b \circ c \rangle_{[1,1]}$. However the third constraint requires that a is equal in length to c. This is impossible to express due to the syntax of the interval restriction operation. The only way to achieve the same effect is to take the above two constraints as they were formulated separately, and enforce them simultaneously by using the intersection operator as follows.

$$\langle a \circ b \rangle_{[1,1]} c \cap a \langle b \circ c \rangle_{[1,1]}$$

Before concluding this section we give some additional examples of recursively defined expressions.

Example 46. *The TRE $\langle a \circ b \rangle_{[1,3]} \cup c$ gives the set of all signals in which event a is followed by event b and where the total time taken by both events is between 1 and 3 time units, as well as the signals in which c is active for any length.*

Another example is given below.

Example 47. *The TRE $(\langle a \rangle_{[2,2]} \circ b)^*$ characterises the scenario in which when a occurs, b occurs exactly 2 seconds after it, and in which this pattern can be played out any number of times.*

3.2.2 Conclusion

In this section we have seen the syntax of TRE and the semantics given to them by Asarin et al. In the next section, we shall give an alternative semantics to timed regular expressions, based on interpretations as an underlying model. In doing so we shall wish to mirror the semantics of Asarin as much as possible, so that we can ensure that the different semantics are consistent with one another.

3.3 An Interpretation Based Semantics for TRE

3.3.1 Introduction

In this section, we shall provide a semantics for timed regular expressions based on the concept of signal interpretations. However unlike in the approach adopted by Asarin et al., we shall not denote the syntactic structures with sets. We shall give the semantics using the notion of satisfaction. What do we mean by this? A timed regular expression will characterise a number of objects, in this case, interpretations. We shall say that an interpretation is characterised by a particular TRE if it satisfies that TRE, where satisfaction means conforming to some definition. The notation that we shall use to show that an interpretation i satisfies an expression ψ, is $i \vDash \psi$.

It is often convenient to restrict the notion of satisfaction by replacing it with *satisfaction over an interval of time*. This situation makes sense if we consider interpretations

as being traces of program execution, with each symbol from Σ, representing a method call, for example. Interpretations can be used to model finite computation by introducing artificial constraints such as silence (see chapter 1). However, a system might not necessarily terminate. In such cases it might be difficult to answer the question of whether an interpretation (a computation) satisfies a timed regular expression (a pattern of method calls with certain durations). However, if we permit satisfaction to range over an interval of time, we may still be able to get interesting satisfaction results over parts of a computation, even when a computation does not terminate.

We therefore denote the fact that an interpretation i satisfies a TRE ψ over an interval of time $[b, e]$, by writing $i \vDash_{[b,e]} \psi$. We now give the semantics in terms of this notion. In order to help the reader's intuition, it might be helpful to regard an interpretation which satisfies a particular TRE as modeling a signal, which would be in the set defined by the TRE (under signal semantics).

3.3.2 The Semantics

Before we start, we shall introduce the empty timed regular expression ε. Although this is not available in Asarin et al.'s original semantics, it is a notion that we shall find useful. The empty TRE has the feature that any interpretation satisfies it over a point interval. Therefore, we define it as follows.

$$i \vDash_{[b,e]} \varepsilon \triangleq b = e$$

We now pass to the TRE base case. With signal semantics, a TRE a, characterised a set of signals which had their symbol a active for some duration of time and then terminated. Now, we wish that an interpretation satisfying the TRE a, has a similar behaviour over its observation interval. To this end, we shall say that an interpretation satisfies the TRE a, if during its entire observation interval it attains the symbol a.

$$i \vDash_{[b,e]} a \triangleq \forall t : \mathbb{T}.\ b \leq t < e \Rightarrow i(a, t)$$

Note that in the constraint on the time, we omit the condition that the value of i has to be a at the point e. Why is this so? The reason has to do with the way our interpretations are built. A singleton segment of length k and symbol a will have its symbol a active between 0 and k, with k not included. However, we still would expect such an interpretation to satisfy the TRE a over the interval $[0, k]$. (i.e. we do not care about what happens at the last point). This definition, allows such satisfaction to occour. An alternative would be to use the integral over the interval in this fashion.

$$i \vDash_{[b,e]} a \triangleq \int_b^e i(a, t)dt = e - b$$

This definition is more complex since it includes the integral of a function; however, it allows us to hide cleanly the fact that we are removing a point from the end of the signal, since removing a finite number of points does not change the value of the integral of a function.

We now pass to the other operators. We shall say that an interpretation satisfies the TRE $\psi_1 \circ \psi_2$ over an interval, if there is a point in the interval such that the interpretation

satisfies ψ_1 before the point and ψ_2 after that point. This is similar in spirit to the signal semantics, in which a signal is in the set $|[\psi_1 \circ \psi_2]|$ because it consists of two parts, one satisfying ψ_1 and the other ψ_2.

$$i \vDash_{[b,e]} \psi_1 \circ \psi_2 \triangleq \exists m : [b, e].\ i \vDash_{[b,m]} \psi_1 \wedge i \vDash_{[m,e]} \psi_2$$

As a side note, this operator is in fact identical to the chop operator normally found in Duration Calculus.

We also define the notion of expression exponentiation to denote the concatenation of an expression multiple times. Here we define raising an expression to the power of zero as ε. This enables us to terminate cleanly the recursive definition of an expression raised to the n^{th} power.

$$i \vDash_{[b,e]} \psi^0 \triangleq i \vDash_{[b,e]} \varepsilon$$
$$i \vDash_{[b,e]} \psi^n \triangleq \psi \circ \psi^{n-1}$$

We shall say that an interpretation satisfies the TRE $\psi_1 \vee \psi_2$ over an interval, if the interpretation satisfies either ψ_1 or ψ_2 over the interval. This is similar to signal semantics; a signal is in the set $|[\psi_1 \vee \psi_2]|$ if it is in either one of the sets described by the individual expressions.

$$i \vDash_{[b,e]} \psi_1 \vee \psi_2 \triangleq i \vDash_{[b,e]} \psi_1 \vee i \vDash_{[b,e]} \psi_2$$

We say that an interpretation satisfies the TRE $\psi_1 \wedge \psi_2$ over an interval, if the interpretation satisfies both ψ_1 and ψ_2 over the interval. The parallel with signal semantics can be seen easily.

$$i \vDash_{[b,e]} \psi_1 \wedge \psi_2 \triangleq i \vDash_{[b,e]} \psi_1 \wedge i \vDash_{[b,e]} \psi_2$$

We now consider the case where an interpretation satisfies ψ^*. Note that this could mean two things. The first is that it satisfies some ψ^n, for $n > 0$. The second case is that ψ^* is regarded as optional, and ignored. We use exponentiation to define it below.

$$i \vDash_{[b,e]} \psi^* \triangleq \exists n : \mathbb{N}.\ i \vDash_{[b,e]} \psi^n$$

This notion is also very similar to that found in signal semantics; a signal is in the set $|[\psi^*]|$ if it is in some set $|[\psi^n]|$.

The last operator is the interval restrict operator. We shall say that i satisfies $\langle\psi\rangle_{[c,d]}$ over the interval $[b, e]$ if it satisfies ψ structurally over $[b, e]$, and if the length of the observation interval is between c and d.

$$i \vDash_{[b,e]} \langle\psi\rangle_{[c,d]} \triangleq i \vDash_{[b,e]} \psi \wedge c \leq e - b \leq d$$

Again, the parallel with signal semantics is clear. For the interval restrict operator, we also have the following useful special cases, where the interval is either bounded from above only or from below only.

$$i \vDash_{[b,e]} \langle\psi\rangle_{[0,d]} \triangleq i \vDash_{[b,e]} \psi \wedge e - b \leq d$$
$$i \vDash_{[b,e]} \langle\psi\rangle_{[c,\infty]} \triangleq i \vDash_{[b,e]} \psi \wedge e - b \geq c$$

3.3.3 Conclusion

This completes our interpretation based semantics for TREs. The way we defined the semantics seems to simulate well the signal semantics. However, we shall not be sure of this before we prove that the semantics are in fact sound and complete.

The semantics we have given to some of the operators strongly resemble the semantics given to some duration calculus operators (for example concatenation resembles chop). This is in part to be expected. Since we are using the same underlying model as duration calculus, some methods of manipulating interpretations will inevitably be reused.

We have also seen the notion of satisfaction over an interval, which does not have any parallel in Asarin et al.'s semantics. This creates an impasse. We want to prove that the signal and interpretation semantics are sound and complete with respect to each other. Intuitively, this means that whenever a signal is characterised by a TRE with signal semantics, a corresponding interpretation (obtained via the functions which allow us to move between models) will satisfy the same TRE using interpretation semantics (and vice versa). The problem here is that satisfaction (for interpretation semantics) is defined over an interval, and this notion is missing from Asarin et. al's semantics.

To this end, in the next chapter, we shall revisit the signal semantics for timed regular expressions. In doing so, we will wish to introduce the notion of satisfaction over an interval in this logic as well. However, as we shall see, this will require a bit of formal manipulation, so instead of using signals as the underlying model, we shall use the notion of signal lists, which is more suited for this purpose.

3.4 A Signal List Based Semantics for TRE

3.4.1 Introduction

We have already introduced Asarin et al.'s semantics for TRE which allow TRE to characterise sets of signals. However, when we were defining the interpretation semantics of TRE, we made the decision to allow a signal to satisfy a TRE over an interval. However, this feature is not available in Asarin et al.'s semantics. We have also said that signal lists are in fact syntactic sugar for signals. However, when working with signal lists we have the advantage of having a full formal theory to use. Therefore, what we shall attempt to do, is to define how a signal list can satisfy Asarin et al.'s TRE over an interval on top of the original TRE semantics, preferably without changing the latter. How can we do this? Remember that a signal list represents a computation of some system. When we ask whether it satisfies a TRE, we are asking whether the computation conforms to the computation pattern characterised by the TRE.

Let us start by considering the special case where we want to ask whether a signal list satisfies a TRE in its entirety. Analogously, we can ask whether the signal satisfies the TRE over the interval from 0 to its length. In this case, it is clear that a signal satisfies the TRE if it is in the set characterised by it. In such a case we can write:

$$\xi \vDash_{[0,|\xi|]} \psi \triangleq \xi \in |[\psi]|$$

However, what should one do in the case where the interval does not cover the entire signal? The best option would be to alter the signal. One would then ask whether the part of the signal which lies between the interval endpoints b and e is in the set characterised by the regular expression. If we assume for the moment the existence of a hypothetical function slice which takes a signal and returns the portion of that signal between b and e, we can write the above as the following statement.

$$\xi \vDash_{[b,e]} \psi \triangleq slice(\xi)_{[b,e]} \in |[\psi]|$$

Thus, in order to be able to define these new signal semantics which include the notion of satisfaction over an interval, we have to define the function *slice* and prove that it satisfies certain properties.

3.4.2 The Function Slice

The function slice takes a signal list and an interval, and returns the portion of the signal lying within that interval. This might involve altering the lengths of some segments. For example,

$$slice(\langle (a,2),(b,3),(c,4)\rangle)_{[1,6]} = \langle (a,1),(b,3),(c,1)\rangle$$

Of course we would like this function to have the property that if we concatenate the output of the function with the sliced off parts to the left and right of the output signal, we would get back the original signal list.

In this case, the left part which has been sliced off is $\langle (a,1)\rangle$, while the right part is $\langle (c,3)\rangle$. What happens if we append them to one another? We would get the expression

$$\langle (a,1) \rangle \overset{\text{sig}}{+\!\!+} \langle (a,1),(b,3),(c,1) \rangle \overset{\text{sig}}{+\!\!+} \langle (c,3) \rangle$$

We immediately note that we do not know how to compute this expression. This is because in order to simplify the theory of signal lists in section 2.2, we did not allow our $\overset{\text{sig}}{+\!\!+}$ operator to concatenate 2 signals which start and end with the same symbol respectively. In this case, we therefore need a new operator which can cope with this situation. Essentially, when faced with a situation in which two signal lists end and start with the same symbol, it should merge these symbols into one symbol. We shall define this kind of gluing concatenation by first defining its underlying append operator, $\overset{\text{sig glue}}{;}$ which we shall call *gluing signal list append*.

Definition 32.

$$\overset{sig\ glue}{;} :: SIG \to (\mathbb{R}^+ \times \Sigma) \to SIG$$

$$\langle \rangle \overset{sig\ glue}{;} (a,k) = \langle (a,k) \rangle$$

$$\xi \overset{sig}{;} (\alpha, k_1) \overset{sig\ glue}{;} (a,k_2) = \xi \overset{sig}{+\!\!+} \mu((\alpha,k_1),(a,k_2))$$

This operator works as follows. If we append a tuple to an empty list, then we end up with a list with one tuple. However, if we are performing a gluing append with a list containing some elements, then, the outcome depends on the value of α. The function μ will test whether α is the same as a. If it is, it will merge the tuples and return a list with just one tuple. Otherwise, it will return a list with both of the tuples. In fact, we shall define μ as follows

Definition 33.

$$\mu :: (\mathbb{R}^+ \times \Sigma) \to (\mathbb{R}^+ \times \Sigma) \to SIG$$

$$\mu(a,k_1)(a,k_2) = \langle (a, k_1 + k_2) \rangle$$

$$\mu(b,k_1)(a,k_2) = \langle (b,k_1),(a,k_2) \rangle$$

Similarly to the other append operators, we can generalise this one to obtain *gluing signal list concatenation*.

Definition 34.

$$\overset{sig\ glue}{+\!\!+} :: SIG \to SIG \to SIG$$

$$\xi \overset{sig\ glue}{+\!\!+} \langle \rangle = \xi$$

$$\xi \overset{sig}{;} (a,k_1) \overset{sig\ glue}{+\!\!+} \langle (\alpha,k_2) \rangle = \xi \overset{sig}{;} (a,k_1) \overset{sig\ glue}{;} (\alpha,k_2)$$

$$(\xi_1 \overset{sig\ glue}{+\!\!+} \xi_2 \overset{sig}{;} (a,k)) = (\xi_1 \overset{sig\ glue}{+\!\!+} \xi_2) \overset{sig}{;} (a,k)$$

The first rule tells us what happens if we are trying to concatenate with an empty list. If the list which we are concatenating contains just one element, then this reduces to an application of gluing signal list append. On the other hand, if the list being concatenated contains more than one element, we recursively call the function until we obtain a list with one element. Each of the excess tuples from the second list, is appended back to the

end of the list.

It is easy to see that *gluing signal list concatenation* is in fact analogous to Asarin et al.'s $\circ$ operator. One should note the increased complexity when compared to the $\overset{\text{sig}}{+\!+}$ operator.

Note that the gluing append operator has arisen because of the needs of the slice function. Thus, if we have an expression which invokes the slice operation, we might well end up with a gluing operator in that expression. However, note that in previous sections, no function dealt with this operator. Therefore, it is desirable to have a lemma which will allow us to get rid of the gluing operator by transforming it into a standard operator. We define the lemma below.

Lemma 48. *The operator* $\overset{sig\ glue}{+\!+}$ *can always be replaced with* $\overset{sig}{+\!+}$. *Let* ξ_1 *and* ξ_2 *be signal lists. Then the following holds.*

$$\xi_1 \overset{sig}{;} (a, k_1) \overset{sig\ glue}{+\!+} \langle(\alpha, k_2)\rangle \overset{sig}{+\!+} \xi_2 = \xi_1 \overset{sig}{+\!+} \mu(a, k_1)(\alpha, k_2) \overset{sig}{+\!+} \xi_2$$

We also have the following observation about μ which follows immediately from its definition. What we are saying here is that if we append the output of a μ operation to a signal list, the duration of the last tuple of the result is greater or equal to the duration of the last parameter of μ.

Corollary 49. *The function* μ *takes two parameters and returns a signal list. The length of the head tuple of its output is always equal to (if no merge occurs) or greater (if a merge occurs) than the length of the second parameter of the* μ *function. We can express this as follows. Let* $\xi = \xi_1 \overset{sig}{+\!+} \mu(\alpha_1, k_1)(\alpha_2, k_2)$. *Also let* $\xi = \xi_2 \overset{sig}{;} (\alpha_2, k)$. *Then* $k \geq k_2$.

Finally, we also state the useful fact that the empty signal list is also the zero of $\overset{\text{sig glue}}{+\!+}$.

Lemma 50. *The empty signal list is the zero of the* $\overset{sig\ glue}{+\!+}$ *operator. Let* ξ *be a signal list. Then the following two laws hold.*

$$\xi \overset{sig\ glue}{+\!+} \langle\rangle = \xi$$

and

$$\langle\rangle \overset{sig\ glue}{+\!+} \xi = \xi$$

We now investigate a way of how one could define the slice function.

Intuition Behind the Definition

As always, it is good practice to begin by stating the signature of the function. Since it will take a signal list and an interval and return a signal list, we can write the type signature as

$$slice :: (SIG \to Interval) \to SIG$$

Now, in order to decide which tuples will be ignored and which will be included, the function slice will have to process each tuple in the signal list. Depending on how the tuple relates to the interval endpoints, b and e, the function will either discard that tuple, return an altered version of it, or return it as a whole.

How could this function work? Recall that a signal list representation is a relative view of a signal, with each tuple only storing the length of the present segment. On the other hand, the interval endpoints are on an absolute time scale. What one can do is to see how the present segment relates to the endpoints b and e, process that tuple in some fashion, and then remove it from the signal list. Then, before processing the next tuple, one can move the endpoints backwards by the amount one has eliminated by removing the prior tuple. The end result is that the relative view of the signal is not affected. Each tuple left has exactly the same relationship to the endpoints of the interval as it had prior to the processing of the last tuple.

The next thing to ask would be what kind of processing should be done for each segment, given the arrangement presented above. Each segment, when processed, if viewed in an absolute context, will be starting at 0 and ending at k, its length. There are then, 3 possibilities of how k can relate to the interval, and these are shown in figure 3.1.

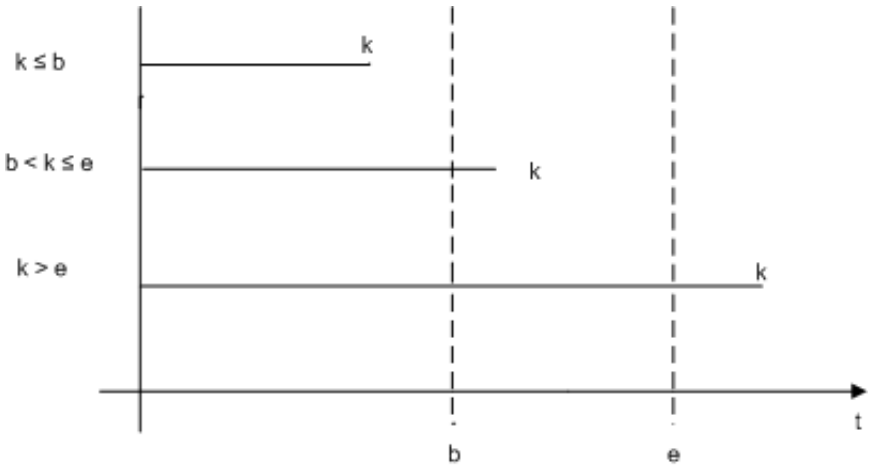


Figure 3.1: Rules of The Slice Function

Before defining the actual slice function, we note that in the explanation above, we found it convenient to traverse the tuples of the signals from the head to the tail of the list in a *cons* like fashion. However, the $\overset{sig}{;}$ operator will not do for this purpose, since it operates in a *snoc* like fashion. To this end, we define the cons version of $\overset{sig}{;}$, which we denote by $\overset{sigcons}{;}$.

Definition 35.

$$\overset{sigcons}{;} :: (\mathbb{R}^+ \times \Sigma) \to SIG \to SIG$$
$$(a,k) \overset{sigcons}{;} \langle\rangle = \langle (a,k) \rangle$$
$$(a,k) \overset{sigcons}{;} xs = \langle (a,k) \rangle \overset{sig}{+\!\!+} xs$$

As with the normal $\overset{sig}{;}$ operator, we can lift the $\overset{sigcons}{;}$ operator to accept lists as its first parameter to obtain a $\overset{sigcons}{+\!\!+}$ operator.

Definition 36.

$$\overset{sigcons}{+\!\!+} :: SIG \to SIG \to SIG$$
$$\langle\rangle \overset{sigcons}{+\!\!+} \xi = \xi$$
$$((a,k) \overset{sigcons}{;} \xi_1 \overset{sigcons}{+\!\!+} \xi_2) = (a,k) \overset{sigcons}{;} (\xi_1 \overset{sigcons}{+\!\!+} \xi_2)$$

Since snoc like traversal is analogous to cons like traversal, it can be proven that these operators satisfy all the lemmas satisfied by the snoc-like operators, namely

- associativity of $\overset{\text{sigcons}}{+\!\!+}$.
- $\langle\rangle$ is the zero of $\overset{\text{sigcons}}{+\!\!+}$.
- switching of $\overset{\text{sigcons}}{;}$ and $\overset{\text{sigcons}}{+\!\!+}$ on a singleton signal list.
- distributivity of $|\,|$ over $\overset{\text{sigcons}}{+\!\!+}$.

The last thing we shall mention is a lemma which allows us to introduce or get rid of $\overset{\text{sigcons}}{+\!\!+}$ operators.

Lemma 51. *The operator* $\overset{sigcons}{+\!\!+}$ *can always be replaced with the operator* $\overset{sig}{+\!\!+}$. *Let* ξ_1 *and* ξ_2 *be signal lists. Then, the following holds.*

$$\xi_1 \overset{sigcons}{+\!\!+} \xi_2 = \xi_1 \overset{sig}{+\!\!+} \xi_2$$

We now formally define the slice function.

Definition of Slice

Definition 37.

$$\begin{aligned} slice(\xi)_{[e,e]} &= \langle\rangle \\ slice((\alpha,k) \overset{sigcons}{;} xs)_{[b,e]} &= slice(xs)_{[b-k,e-k]} \text{ for } k \leq b \\ &= (\alpha, k-b) \overset{sigcons}{;} slice(xs)_{[0,e-k]} \text{ for } b < k \leq e \\ &= \langle (a, e-b) \rangle \text{ for } k > e \end{aligned}$$

Further more, we constrain the interval such that $e \leq |\xi|$. In this way, one cannot invoke the slice function over a segment of the signal which does not exist.

An explanation of the mechanics of the function is in order. The first case is telling us that if we call slice over a point interval, we will get an empty list. This is consistent with what we want to achieve, since we want to avoid point like signal segments.

There are then three other cases which do not depend on the interval parameter being a point interval. These stem directly from the observations in figure 3.1. We shall refer to these cases as cases 2, 3 and 4. As can be seen in the diagram, for case 2, if k is less than or equal to b, then the tuple is not in the interval, and is discarded. In the third case, if k is between b and e, then we want to keep the part of the segment lying between the endpoints and discard the rest of the segment. If k exceeds e, as one can see in the fourth case, we return the part of the segment between b and e.

We now prove a number of results about the slice function.

Properties of Slice

It can be shown that the slice function terminates for both finite and interestingly, also infinite signal lists.

Theorem 52. *Let ξ be a (possibly infinite) signal list. Then $slice(\xi)_{[b,e]}$ terminates.*

Something which we wish to hold is that *slice*, when given an interval $[b, e]$, will return a signal of length $e - b$. In order to prove this, one also needs the following interesting invariant, which effectively states that whenever slice adds a segment to its output and recursively calls itself, the interval of the recursive call is shortened by the amount that was added.

Lemma 53. *The length of the segment that is added to the output, by any invocation of slice is equal to the difference between the lengths of the present interval and the interval of the next call. We can express this as follows. Consider $slice((\alpha, m) \overset{sigcons}{;} \xi)_{[b_n,e_n]}$, and suppose that it expands into a call with interval $[b_{n+1}, e_{n+1}]$, and adds a segment of length k_n to its output. Then, the equality*

$$k_n + e_{n+1} - b_{n+1} = e_n - b_n$$

holds.

The above lemma can then be used to prove the next theorem.

Theorem 54. *The length of the result of a slice call is equal to the length of the call's interval. Let ξ be a signal list. Then $|slice(\xi)_{[b,e]}| = e - b$.*

A very important property satisfied by slice is that it can be split into two slice calls through the use of gluing signal concatenation.

Lemma 55. *Calling slice with interval $[b, e]$ on a signal is equivalent to gluing the signals obtained from calling slice with interval $[b, m]$ and $[m, e]$, for any m between b and e inclusive. We can express this as follows. Let ξ be a signal list. Then for $b \leq m \leq e$,*

$$slice(\xi)_{[b,e]} = slice(\xi)_{[b,m]} \overset{sig\ glue}{+\!\!+} slice(\xi)_{[m,e]}$$

The following two results about the behaviour of slice can also be derived. The first one tells us the precise form which the traces of slice can take. In reading the following theorem, one must keep in mind that earlier we have said that slice would unroll into one of four possible cases, depending on the guard that is satisfied by the length of the tuple being processed at that step in the computation.

Theorem 56. *Consider the execution of slice(). At each step it can unroll into cases 1,2,3 or 4. Then the execution trace of slice() is of one of the following forms.*

$$1$$

$$2^* \vdash 4$$

$$2^* \vdash 3^+ \vdash (1|4)$$

If one reads the proof of the last theorem one can draw the following corollary about the behaviour of the slice function. It tells us the exact conditions when a call on a point interval can be made.

Corollary 57. *Consider the function call* $slice(\xi)_{[b,e]}$*, with* $b \neq e$*. At no point can a recursive call unroll into* $slice(\xi)_{[n,n]}$ *unless* $n = 0$*. It is also the case that for the latter to happen, the recursive call must have been a case 3 call.*

We now prove the intuitive but important fact that if we start with a signal, and apply slice to it, we are essentially dividing the signal into three pieces, each of which is possibly empty.

Lemma 58. *Calling slice over a signal divides it into 3 pieces. The original signal can be recovered by using gluing concatenation. We express this as follows. Let* ξ *is a signal list, then*

$$\forall b, e : \mathbb{R}.\ b \leq e \Rightarrow\ \exists\, \xi_1, \xi_2 : SIG.\ \xi = (\xi_1 \overset{sig\ glue}{+\!\!+} slice(\xi)_{[b,e]} \overset{sig\ glue}{+\!\!+} \xi_2)$$

We shall now extend the above result to say that if we use slice, splitting the original list into three parts, then the length of the first part, $|\xi_1|$ must be equal to b, where b is the original interval's right endpoint.

Theorem 59. *The slice function divides its input into three pieces. The length of the first piece is equal to b, the original left endpoint. We express this below. Consider* $slice(\xi)_{[b,e]}$*. Recall that* $\xi = \xi_1 \overset{sig\ glue}{+\!\!+} slice(\xi)_{[b,e]} \overset{sig\ glue}{+\!\!+} \xi_2$*. Then,* $|\xi_1| = b$

We now prove a theorem which will allow us to 'shortcut' the execution of slice under certain conditions.

Theorem 60. *If we can divide the input to slice such that the length of the first part is less than the left interval endpoint b, then this is equivalent to calling slice on the second part with altered endpoints. We formulate this as follows. Let* ξ_1 *and* ξ_2 *be signal lists. If* $|\xi_1| \leq b$*, then*

$$slice(\xi_1 \overset{sigcons}{+\!\!+} \xi_2)_{[b,e]} = slice(\xi_2)_{[b-|\xi_1|, e-|\xi_1|]}$$

This concludes our discussion of the slice function. We are now in a position to give the Signal List Semantics for TRE.

3.4.3 Signal List Semantics

In this section we finally give the signal list semantics for timed regular expressions, which includes the notion of satisfaction over an interval. As we said before, we would do this by combining Asarin's TRE semantics with the function slice.

In general, we will say that a signal ξ satisfies a timed regular expression ψ over $[b, e]$ if its sliced version is an element of the set generated by ψ. We state this general form below.

$$\xi \vDash^{SIG}_{[b,e]} \psi \triangleq slice(\xi)_{[b,e]} \in |[\psi]|$$

The reader may note that when defining the interpretation semantics, we introduced the timed regular expression ε, which was not present in Asarin et al.'s original semantics. When we used it for interpretations, we said that this TRE had the property of being satisfied by any interpretation on a point interval. What should the corresponding notion be for signals? When we introduced the notion of satisfaction over an interval for signals, we agreed that a signal ξ would satisfy a TRE over an interval, if $slice(\xi)_{[b,e]}$ was in the set characterised by that TRE. But what does a signal over a point interval look like? From the definition of slice, we know that for any signal, $slice(\xi)_{[n,n]}$ is the empty signal list. Now, we said that ε has the property of being satisfied by any interpretation over a point interval. Since we want the signal semantics and the interpretation semantics to be consistent, we want to ensure ε also has the property of being satisfied by any signal over a point interval. Since all signals resolve to empty lists on point intervals, the set generated by ε *has* to be the set containing the empty signal list. We define this below.

$$\xi \vDash^{SIG}_{[b,e]} \varepsilon \triangleq slice(\xi)_{[b,e]} \in \{\langle\rangle\}$$

We now define the rest of the semantics. By using the general form of satisfaction for signals over intervals defined in the beginning of this section, these follow easily.

$$\xi \vDash^{SIG}_{[b,e]} a \triangleq slice(\xi)_{[b,e]} \in \{a^r \mid r \in \mathbb{R}^+\}$$

$$\xi \vDash^{SIG}_{[b,e]} \psi_1 \vee \psi_2 \triangleq slice(\xi)_{[b,e]} \in |[\psi_1]| \cup |[\psi_2]|$$

$$\xi \vDash^{SIG}_{[b,e]} \psi_1 \wedge \psi_2 \triangleq slice(\xi)_{[b,e]} \in |[\psi_1]| \cap |[\psi_2]|$$

We now arrive to defining what it means for a signal list to satisfy an $\psi_1 \circ \psi_2$ over an interval. We shall say that this is the case if the sliced signal can be split into two signals; where the first one satisfies ψ_1 and the second one satisfies ψ_2. But what operator should we use to perform this split? The best way is to use gluing signal concatenation, since we know that it is analogous to the $\circ$ operator. Also note that in this case gluing concatenation allows us to split the signal list at any point we wish, simulating the interpretation semantics. Had we used $\overset{\text{sig}}{+\!\!+}$, we would have only been able to split the list between the tuples. By using $\overset{\text{sig glue}}{+\!\!+}$ we are able to split the tuples themselves.

$$\xi \vDash^{SIG}_{[b,e]} \psi_1 \circ \psi_2 \triangleq slice(\xi)_{[b,e]} \in \{\xi_1 \overset{\text{sig glue}}{+\!\!+} \xi_2 \mid \xi_1 \in |[\psi_1]| \wedge \xi_2 \in |[\psi_2]|\}$$

The semantics for the kleene star and interval restriction operators do not change from Asarin's formulation.

$$\xi \vDash^{SIG}_{[b,e]} \psi^* \triangleq slice(\xi)_{[b,e]} \in \bigcup_{i=0}^{\infty} |[\psi^i]|$$

$$\xi \vDash^{SIG}_{[b,e]} \langle\psi\rangle_I \triangleq slice(\xi)_{[b,e]} \in |[\psi]| \cap \{\xi : Signal|\ |\xi| \in I\}$$

3.4.4 Conclusion

We have thus managed to augment TRE (with signal semantics) with the notion of satisfaction over an interval. We saw that in order to be able to do this, we first had to define the slice function and prove a number of properties about it to make sure its behaviour was as we expected. The reason why we embarked on some of these proofs may not seem clear right now. However, we shall see that when we come to prove that signal list and signal interpretation semantics are sound with respect to one another, we will have to work with expressions which include the slice function. To this end we will want to have certain lemmas with which we can manipulate the slice function as desired.

The notion of satisfaction over an interval itself was achieved by combining Asarin et al.'s original semantics with the slice function. This proved to be a very elegant way of adding a layer on top of Asarin et. al's original semantics without altering the original semantics. In the next section we shall put everything together in order to prove that TRE with signal semantics are sound and complete with respect to TRE with interpretation semantics.

3.5 Soundness and Completeness of Semantics

3.5.1 Introduction

So far we have given two different semantics to TRE which have the ability to handle satisfaction over an interval. The first uses interpretations as its underlying model, whilst the other uses signal lists. How can one make sure that these semantics are consistent with respect to one another? And what does one mean by consistent? The ideal situation would be that if we had some timed regular expression ψ, a signal list ξ and a signal interpretation i, such that ξ and i encode the same object, then, if i satisfied ψ, so would ξ and vice versa. In this case, by encoding the same object we mean that i would be the result of applying the constructions for moving between models to ξ, and vice versa.

More formally, we shall say that the interpretation semantics are sound with respect to the signal list semantics if the following is true.

$$\forall i : I, \psi : TRE.\ i \vDash^{I}_{[b,e]} \psi \Rightarrow c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi$$

It is also complete with respect to signal list semantics, if the reverse holds.

$$\forall i : I, \psi : TRE.\ c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi \Rightarrow i \vDash^{I}_{[b,e]} \psi$$

We also have to prove that signal list semantics are sound and complete with respect to the signal interpretation semantics. In this case, we have the following statement for soundness.

$$\forall \xi : SIG, \psi : TRE.\ \xi \vDash^{SIG}_{[b,e]} \psi \Rightarrow c2i(s2c(\xi)) \vDash^{I}_{[b,e]} \psi$$

and the following corresponding statement for completeness.

$$\forall \xi : SIG, \psi : TRE.\ c2i(s2c(\xi)) \vDash^{I}_{[b,e]} \psi \Rightarrow \xi \vDash^{SIG}_{[b,e]} \psi$$

Therefore, in this section, we shall have to prove the above four theorems. We shall start by proving the soundness results. Then, we shall see that by combining the soundness theorems with the fact that the functions allowing us to move between models always have an inverse, completeness follows immediately.

How shall we prove the soundness statements? We note that we are proving a statement about timed regular expressions in general, and that timed regular expressions are defined recursively. Therefore, one technique that we can use, is structural induction. How does this technique work? First we prove the statement for all base cases. In this case, we have two base cases; one when the timed regular expression is ε and another when it is $a \in \Sigma$. As we shall see, this will prove to be the hardest part. Then we perform the inductive step, in which we assume soundness for the component timed regular expressions and prove that when we combine them via the various operators, soundness will still hold.

We shall start by proving the soundness of interpretation semantics with respect to signal list semantics. We shall call this the First Soundness Theorem. In order not to clutter the text, the proofs can be found in the appendix.

3.5.2 The First Soundness Theorem

As we have discussed above, we shall first prove the soundness of the base cases. The first result regards the soundness of the expression ε.

Theorem 61. *ε's interpretation semantics are sound with respect to its signal list semantics. Let i be an interpretation. Then,*

$$i \vDash^{I}_{[b,e]} \varepsilon \Rightarrow c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \varepsilon$$

The interpretation semantics for $a \in \Sigma$ can also be proven to be sound with respect to the signal semantics. If fact, this is the core soundness proof. It is rather difficult because, one has to show that if we start with an interpretation which satisfies the TRE, after moving through the two constructions in order to change our model to a signal list, we end up with a structure that has the property of satisfying the signal semantics. The reader is referred to the proof itself for the details.

Theorem 62. *a's interpretation semantics are sound with respect to its signal list semantics. Let i be an interpretation. Then,*

$$i \vDash^{I}_{[b,e]} a \Rightarrow c2s(i2c(i)) \vDash^{SIG}_{[b,e]} a$$

With the base case complete, we pass to the recursive cases. The recursive cases are easier, since one can assume the soundness of the component regular expressions. What has to be shown, in these cases is that the function performed by the operators when given interpretation semantics can be reduced to the function performed by the operators when given signal semantics. One can start by showing the soundness of the $\wedge$ operator.

Theorem 63. *If ψ_1 and ψ_2 under interpretation semantics are sound with respect to their signal list semantics, then $\psi_1 \wedge \psi_2$ is also sound. Let i be an interpretation. Then,*

Assuming

$$i \vDash^{I}_{[b,e]} \psi_1 \Rightarrow c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi_1$$

and

$$i \vDash^{I}_{[b,e]} \psi_2 \Rightarrow c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi_2$$

we can conclude that

$$i \vDash^{I}_{[b,e]} \psi_1 \wedge \psi_2 \Rightarrow c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi_1 \wedge \psi_2$$

The $\vee$ operator can also be proven to be sound.

Theorem 64. *If ψ_1 and ψ_2 under interpretation semantics are sound with respect to their signal list semantics, then $\psi_1 \vee \psi_2$ is also sound. Let i be an interpretation. Then,*

Assuming

$$i \vDash^{I}_{[b,e]} \psi_1 \Rightarrow c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi_1$$

and

$$i \vDash^{I}_{[b,e]} \psi_2 \Rightarrow c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi_2$$

we can conclude that

$$i \vDash^{I}_{[b,e]} \psi_1 \vee \psi_2 \Rightarrow c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi_1 \vee \psi_2$$

We can also state that the concatenation operator is sound.

Theorem 65. *If ψ_1 and ψ_2 under interpretation semantics are sound with respect to their signal list semantics, then $\psi_1 \circ \psi_2$ is also sound. Let i be an interpretation. Then,*

Assuming

$$i \vDash^{I}_{[b,e]} \psi_1 \Rightarrow c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi_1$$

and

$$i \vDash^{I}_{[b,e]} \psi_2 \Rightarrow c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi_2$$

we can conclude that

$$i \vDash^{I}_{[b,e]} \psi_1 \circ \psi_2 \Rightarrow c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi_1 \circ \psi_2$$

Given that concatenation is sound it follows that exponentiation is also sound. This is due to the fact that exponentiation is simply repeated concatenation. Proving that ψ^n is sound, will also simplify the soundness proof for ψ^*.

Lemma 66. *If ψ under interpretation semantics is sound with respect to its signal list semantics, then ψ^n is also sound. Let i be an interpretation. Then,*

Assuming

$$i \vDash^{I}_{[b,e]} \psi \Rightarrow c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi$$

we can conclude that

$$i \vDash^{I}_{[b,e]} \psi^n \Rightarrow c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi^n$$

The soundness of Kleene Star follows easily from the soundness of exponentiation. This occurs because satisfying Kleene Star means satisfying the expression inside the * for some particular exponent.

Theorem 67. *If ψ under interpretation semantics is sound with respect to the signal list semantics, then ψ^* is also sound. Let i be an interpretation. Then,*

Assuming

$$i \vDash^{I}_{[b,e]} \psi \Rightarrow c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi$$

we can conclude that

$$i \vDash^{I}_{[b,e]} \psi^* \Rightarrow c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi^*$$

The final element required in order to complete the second soundness theorem is a proof that interval restriction under interpretation semantics in sound with respect to interval restriction under signal semantics.

Theorem 68. *If ψ under interpretation semantics is sound with respect to the signal list semantics, then for any interval [c,d], $\langle\psi\rangle_{[c,d]}$ is also sound. Let i be an interpretation. Then,*

Assuming

$$i \vDash^{I}_{[b,e]} \psi \Rightarrow c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi$$

we can conclude that

$$i \vDash^{I}_{[b,e]} \langle\psi\rangle_{[c,d]} \Rightarrow c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \langle\psi\rangle_{[c,d]}$$

With this last proof, the first soundness theorem is complete. We now pass to proving the opposite soundness result; namely that signal list semantics are sound and complete with respect to signal interpretation semantics. We shall refer to this collection of results as the *Second Soundness Theorem.*

3.5.3 The Second Soundness Theorem

In the second soundness theorem, we prove that the signal semantics we have given for TRE, are sound with respect to the interpretation semantics. If we were to state the requirement in general, with ξ being a signal list and ψ a timed regular expression, we would write:

$$\forall \xi : SIG, \psi : TRE.\ \xi \vDash^{SIG}_{[b,e]} \psi \Rightarrow c2i(s2c(\xi)) \vDash^{I}_{[b,e]} \psi$$

We shall do this by structural induction. As with the previous soundness proof, the difficult part will be to prove the base cases. Moreover, once one has completed the proof of the base cases, we shall see that the proofs in the inductive cases mimic the proofs in the previous section.

We start with the base case TRE ε.

Theorem 69. *ε's signal list semantics are sound with respect to its interpretation semantics. Let ξ be a signal list. Then,*

$$\xi \vDash^{SIG}_{[b,e]} \varepsilon \Rightarrow c2i(s2c(\xi)) \vDash^{I}_{[b,e]} \varepsilon$$

One also has to show soundness for the second base case, the $a \in \Sigma$ operator.

Theorem 70. *a's signal list semantics are sound with respect to its interpretation semantics. Let ξ be a signal list. Then,*

$$\xi \vDash^{SIG}_{[b,e]} a \Rightarrow c2i(s2c(\xi)) \vDash^{I}_{[b,e]} a$$

We can now proceed with the soundness proofs for the operators. The $\wedge$ operator under signal list semantics is in fact sound with respect to interpretation semantics.

Theorem 71. *If ψ_1 and ψ_2 under signal list semantics are sound with respect to their interpretation semantics, then $\psi_1 \wedge \psi_2$ is also sound. Let ξ be a signal list. Then,*

Assuming

$$\xi \models^{SIG}_{[b,e]} \psi_1 \Rightarrow c2i(s2c(\xi)) \models^{I}_{[b,e]} \psi_1$$

and

$$\xi \models^{SIG}_{[b,e]} \psi_2 \Rightarrow c2i(s2c(\xi)) \models^{I}_{[b,e]} \psi_2$$

we can conclude that

$$\xi \models^{SIG}_{[b,e]} \psi_1 \wedge \psi_2 \Rightarrow c2i(s2c(\xi)) \models^{I}_{[b,e]} \psi_1 \wedge \psi_2$$

The next operator for which we will prove soundness is the $\vee$ operator.

Theorem 72. *If ψ_1 and ψ_2 under signal list semantics are sound with respect to their interpretation semantics, then $\psi_1 \vee \psi_2$ is also sound. Let ξ be a signal list. Then,*

Assuming

$$\xi \models^{SIG}_{[b,e]} \psi_1 \Rightarrow c2i(s2c(\xi)) \models^{I}_{[b,e]} \psi_1$$

and

$$\xi \models^{SIG}_{[b,e]} \psi_2 \Rightarrow c2i(s2c(\xi)) \models^{I}_{[b,e]} \psi_2$$

we can conclude that

$$\xi \models^{SIG}_{[b,e]} \psi_1 \vee \psi_2 \Rightarrow c2i(s2c(\xi)) \models^{I}_{[b,e]} \psi_1 \vee \psi_2$$

It can also be proved that the $\circ$ operator is also sound.

Theorem 73. *If ψ_1 and ψ_2 under signal list semantics are sound with respect to their interpretation semantics, then $\psi_1 \circ \psi_2$ is also sound. Let ξ be a signal list. Then,*

Assuming

$$\xi \models^{SIG}_{[b,e]} \psi_1 \Rightarrow c2i(s2c(\xi)) \models^{I}_{[b,e]} \psi_1$$

and

$$\xi \models^{SIG}_{[b,e]} \psi_2 \Rightarrow c2i(s2c(\xi)) \models^{I}_{[b,e]} \psi_2$$

we can conclude that

$$\xi \models^{SIG}_{[b,e]} \psi_1 \circ \psi_2 \Rightarrow c2i(s2c(\xi)) \models^{I}_{[b,e]} \psi_1 \circ \psi_2$$

Since exponentiation is just repeated concatenation, exponentiation will also turn out to be sound.

Lemma 74. *If ψ under signal list semantics is sound with respect to its interpretation semantics, then ψ^n is also sound. Let ξ be a signal list. Then,*

Assuming

$$\xi \models^{SIG}_{[b,e]} \psi \Rightarrow c2i(s2c(\xi)) \models^{I}_{[b,e]} \psi$$

we can conclude that

$$\xi \models^{SIG}_{[b,e]} \psi^n \Rightarrow c2i(s2c(\xi)) \models^{I}_{[b,e]} \psi^n$$

Once we know that exponentiation is sound, it is trivial to show that Kleene Star is also sound.

Theorem 75. *If ψ under signal list semantics is sound with respect to its interpretation semantics, then ψ^* is also sound. Let ξ be a signal list. Then,*

Assuming

$$\xi \vDash^{SIG}_{[b,e]} \psi \Rightarrow c2i(s2c(\xi)) \vDash^{I}_{[b,e]} \psi$$

we can conclude that

$$\xi \vDash^{SIG}_{[b,e]} \psi^* \Rightarrow c2i(s2c(\xi)) \vDash^{I}_{[b,e]} \psi^*$$

Finally, one can complete the second soundness theorem by proving the soundness of the interval restrict operator.

Theorem 76. *If ψ under signal list semantics is sound with respect to its interpretation semantics, then $\langle\psi\rangle_{[c,d]}$ is also sound. Let ξ be a signal list. Then,*

Assuming

$$\xi \vDash^{SIG}_{[b,e]} \psi \Rightarrow c2i(s2c(\xi)) \vDash^{I}_{[b,e]} \psi$$

we can conclude that

$$\xi \vDash^{SIG}_{[b,e]} \langle\psi\rangle_{[c,d]} \Rightarrow c2i(s2c(\xi)) \vDash^{I}_{[b,e]} \langle\psi\rangle_{[c,d]}$$

We now know that the semantics are sound with respect to each other. We shall now prove completeness.

3.5.4 Completeness

Completeness follows easily for both cases, from the soundness theorems and the fact that each of our constructions has an inverse. We state the *first completeness theorem* below. This theorem is the reverse direction of the first soundness theorem.

Theorem 77. *The interpretation semantics for TRE are complete with respect to the signal list semantics for TRE. Let i be an interpretation and ψ a TRE. Then the following holds.*

$$c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi \Rightarrow i \vDash^{I}_{[b,e]} \psi$$

Similarly, the following result, which we shall call the *second completeness theorem*, can be easily obtained. This result is in fact the reverse direction of the second soundness theorem.

Theorem 78. *The signal list semantics for TRE are complete with respect to the interpretation semantics for TRE. Let ξ be an signal list and ψ a TRE. Then the following holds.*

$$c2i(s2c(\xi)) \vDash^{I}_{[b,e]} \psi \Rightarrow \xi \vDash^{SIG}_{[b,e]} \psi$$

With this result, we conclude this section.

3.5.5 Conclusion

In this section we have shown that the signal list semantics we have given to timed regular expressions and sound and complete with respect to the signal interpretation semantics. In order to prove such a result over the logic as a whole we applied the technique of structural induction. This involved first proving that the base cases for the logic are sound, and then performing an inductive step in order to prove the soundness of the operators. This inductive step was performed by assuming that the components of the expression were sound, and by proving that under that assumption, the operator would also preserve soundness.

Whilst proving these results we noted that the hardest part was proving the base cases. Once these were proved, the inductive cases were much easier. Now that we know that signal semantics are sound with respect to interpretation semantics, we are certain that when a signal satisfies a TRE, so will the corresponding interpretation. A similar observation can be made for the soundness of interpretation semantics with respect to signal semantics. The major benefit of this is that any results we prove for TRE with one semantics, will also hold for the other semantics.

In the next section we shall summarise the achievements of this chapter.

3.6 Conclusion

In this chapter we have moved from the theory of signals and interpretations to timed regular expressions. We have seen how timed regular expressions are essentially concise representations for characterising entire groups of objects, such as signals and interpretations. Depending on the type of objects one wants to characterise, one can give a semantics to timed regular expressions in terms of the required objects.

The first semantics we considered was the original one given by Asarin et al. Following in the spirit of this semantics, we gave another semantics to TRE, based on interpretations. As we did so, we tried to use these semantics to simulate the original semantics, in terms of interpretations. The interpretation semantics were defined in terms of the notion of satisfaction. The attentive reader might have noted that these semantics were defined such that when an interpretation satisfies a particular TRE, an analogous signal would be in the set characterised by that TRE when using signal semantics. This ensured that the two semantics were kept consistent with one another, which is a prerequisite for the ability to prove that they are sound with respect to one another.

However, before we could attempt the soundness proof, we noted that whilst the interpretation semantics featured a concept of satisfaction over an interval, this was missing from the original TRE semantics. Thus, we gave a third semantics to TRE, based on signal lists and the slice function, which enabled us to cleanly build a layer on top of the original TRE semantics without changing them. This, in turn, paved the way for the soundness and completeness proofs, which were completed successfully, thus confirming that the way we defined signal list and interpretation semantics was consistent. In the next chapter, we shall consider rigorously the concepts of slowdown and speedup.

Chapter 4

Slowdown and Speedup

4.1 Introduction

In this chapter, we shall talk about matters related to system slowdown and system speedup. We shall first review the original theory of slowdown and speedup introduced by Colombo et al. [CPS][Col08] in section 4.2. Once this has been done, we shall finally be able to complete the task $2'$ as set in the introduction. Recall that this task involved giving TRE an interpretation based semantics, along with a proof that these were sound with respect to the original semantics. These two items have been successfully tackled in the previous chapter. The last step in task $2'$ was to apply the theory of slowdown and speedup to timed regular expressions with interpretation based semantics in order to discover which fragments of the logic are unaffected by the phenomena of slowdown and speedup. This will be done in section 4.3.

Following this, in section 4.4, we shall proceed to also tackle elements of task $1'$, as outlined in the introduction. Recall that in task $1'$, we determined that an alternative approach to proving slowdown and speedup results for a logic which did not have interpretation based semantics was to define an alternative theory of slowdown and speedup which was based on the same underlying model as that logic. One would then apply this theory directly to that logic in order to derive slowdown and speedup results, avoiding the intricacies we have already witnessed in completing task $2'$. To this end, what we shall do is identify the major elements which need to be defined for such a new theory. As we do so, we shall formulate a new theory of our own based on signal lists as an example. This is useful, since once it is achieved, if one wants to prove slowdown and speedup results for another logic which uses signal semantics, *in theory* one could do so directly. Let us thus start by reviewing the original theory of slowdown and speedup.

4.2 The Theory of Slowdown and Speedup

The theory of slowdown and speedup is a collection of concepts which allows us to model and to reason about what occurs when a system slows down or speeds up. In this section, we summarise the results making part of Colombo's theory of slowdown and speedup. For a more in depth treatment, the interested reader is referred to [Col08] or [CPS]. Not all of the results shall be proven, but a few which we deem to be relevant for a later purpose

will be outlined.

4.2.1 Introduction

When a system is slowed down, what happens is that system events, which in our case correspond to symbols, take longer to occour and last for a longer time. When a system is speeded up, events seem to take a shorter time before occurring, and also last for a shorter time. If one were to visualise the operation of a system as an interpretation, a slowed down version of the function would look like the original version but stretched in the time domain. Similarly a speeded up interpretation would look like the original, but compressed in the time domain. We show this in figure 4.1.

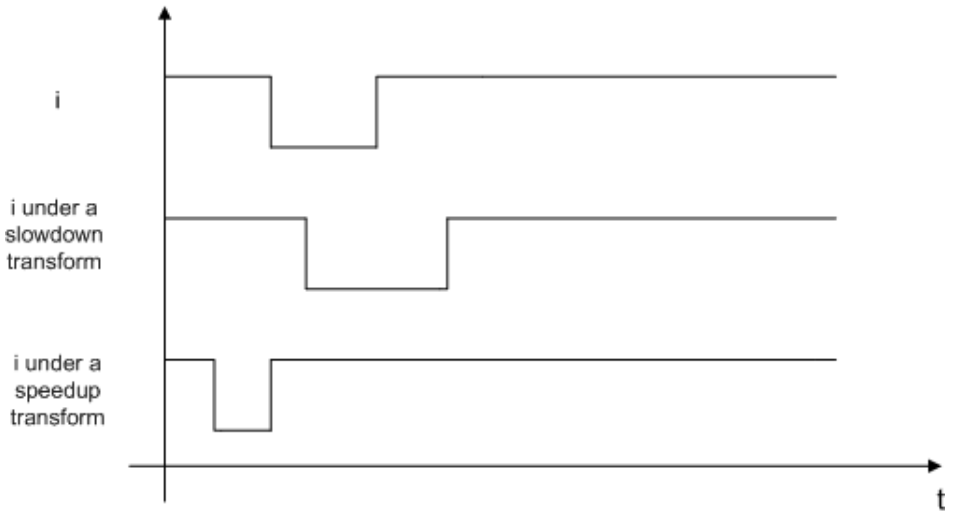


Figure 4.1: Speeded Up and Slowed Down Interpretations

A time transform is a concept which captures the relationship between the original system behaviour and that of the slowed down/speeded up system.

Time Transform Definition

We shall represent the time domain $\mathbb{R}_0^+$ by the symbol $\mathbb{T}$. Suppose that we represent a system's behaviour as an interpretation. When a system slows down or speeds up, what will happen is that events will change their time of occurrence, whilst maintaining the order in which they happen. So a time transform is a mechanism which will take each point in the time domain of the original interpretation and remap it to some other point, whilst obeying certain constraints. We define this concept below.

Definition 38. *A time transform s is a continous total function*

$$s :: \mathbb{T} \to \mathbb{T}$$

Such a function must have the following three properties.

1. $s(0) = 0$, *that is the time point 0 remains where it is.*

2. *$lim_{t\to\infty} s(t) = \infty$, that is, the function does not converge to some value.*

3. *s is monotonic, that is, for any two times t_1 and t_2, $t_1 < t_2 \Rightarrow s(t_1) < s(t_2)$. This requirement means that when transforming the time domain, the ordering of the points must be preserved.*

Let us consider an example of a time transform.

Example 79. *The function $s(t) = 2t$ is a time transform, since it satisfies all three requirements stated above. What does this function do? Suppose an event starts at $t = 2$ and ends at $t = 3$. Once the system is retimed, it will start occurring at $t = 4$ and end at $t = 6$. So the event starts later, and its duration is double the time of the original event. Clearly, this function is slowing the system down.*

What happens when we apply a time transform to an interpretation? We said that its points will be remaped. What is important here is that the old point in the original interpretation and the new point in the transformed interpretation, make part of exactly the same event. Therefore, the system must exhibit the same behaviour at the original and at the retimed points. How can we express this constraint? We do so by using the following law.

Definition 39. *Let i be an interpretation over Σ, $\alpha \in \Sigma$, s a time transform and i_s the transformed interpretation. Then,*

$$\forall \alpha : \Sigma, t : \mathbb{T}.\ i_s(\alpha, s(t)) = i(\alpha, t)$$

This asserts that for all time points, the value of i at that point for some symbol is identical to the value of i_s at the transformed time point for that symbol. So far our time transforms do not talk about speedup and slowdown. We are now in a position to refine the concept of a time transform to define the concepts of slowdown and speedup transforms.

Slowdown and Speedup Transforms

If one wanted to stretch the time domain using a time transform (to simulate slowdown), one would wish that for any two points in the transformed version, the interval between these two points is larger than the interval between the untransformed points. Similarly for time compressing transforms (simulating speedup), the interval between two transformed points, should be smaller than the interval between the original points. We capture these two notions in the definitions below.

Definition 40. *A time transform s is said to be a* time stretch transform *or a* slowdown transform *if is monotonic on intervals:*

$$\forall t_1, t_2 : \mathbb{T}.\ t_1 < t_2 \Rightarrow s(t_2) - s(t_1) > t_2 - t_1$$

Example 80. *Consider the function $s(t) = kt$. For $k > 1$ is a linear slowdown transform. This means that all events take k times longer to occour. Slowdown does not have to be linear. For example, if software is operating on a network which is becoming always more and more congested, one could have a quadratic slowdown (or some other suitable exponent).*

We now define the meaning of a speedup transform.

Definition 41. *A time transform s is a* time compress transform *or a* speedup transform *if it is anti-monotonic on intervals:*

$$\forall t_1, t_2 : \mathbb{T}.\ t_1 < t_2 \Rightarrow s(t_2) - s(t_1) < t_2 - t_1$$

Example 81. *Consider $s(t) = kt$. For $k < 1$ is a linear speedup transform. This indicates that all events take k times less to occour. A linear speedup could occour, for example, when one has upgraded the processor to be twice as fast.*

In general we use the symbols $\overleftrightarrow{\mathbb{T}}$ and $\overrightarrow{\mathbb{T}}$ to denote the sets of slowdown transforms and speedup transforms respectively. Besides the above definitions, Colombo et al. also prove the following results about time transforms.

- Time transforms are bijective
- The inverse of a slowdown transform is a speedup transform
- The inverse of a speedup transform is a slowdown transform
- When the time transform is the identity function, it does not change the interpretation, that is, for any interpretation i, $i = i_{id}$.
- If i is an interpretation and s_1 and s_2 are time transforms, then $i_{(s_1 \circ s_2)} = (i_{s_1})_{s_2}$.

We now have all we need in order to discuss the next contribution of the theory of speedup and slowdown.

4.2.2 Logic Formulas and Slowdown/Speedup Properties

Recall that in the introduction, we had said that a formula of a logic could be used to denote a property which some system would then be required to satisfy. We also said, that using the theory of slowdown and speedup we could prove that certain formulas possessed certain traits which would make them immune from the effects of slowdown and speedup.

In the theory of slowdown and speedup, there are four of these traits in all. When a formula possesses one of these properties, a special kind of relationship holds between every interpretation, its transformed version and the logic formula.

A formula ψ can be:

- Slowdown Truth Preserving, written $sdtp(\psi)$.
- Speedup Truth Preserving, written $sutp(\psi)$.
- Slowdown False Preserving, written $sdfp(\psi)$.

- Speedup False Preserving, written $sufp(\psi)$.

We shall discuss these notions below. In doing so, it shall be useful to find an operator which we can use to illustrate the point. Let us consider two operators which are similar to the duration calculus *length* operator. The operator $\int a \geq k$, when given an interpretation, and an interval of time, will return true if the total time in which the symbol a is active in that interval is greater or equal to k. Similarly, the operator $\int a \leq k$, when given an interpretation and an interval, will return true if the total time for which a is active in the given interval is less than or equal to k. These two operators are convenient for the sake of exposition because we can intuitively argue that they satisfy certain traits. We start by discussing slowdown/speedup truth preserving formulas.

Slowdown/Speedup Truth Preserving Formulas

A formula ψ representing a property is said to be *slowdown truth preserving* if whenever the system satisfies the property, slowing the system down cannot break the property. If we let the behaviour of a system be represented by an interpretation, we can formally define this concept as follows.

Definition 42. *A formula ψ is said to be slowdown truth preserving if whenever one has an interpretation i which satisfies ψ over an interval $[b, e]$,* any *slowed down interpretation i_s, will satisfy ψ over the stretched sub interval. This can be stated as follows.*

$$sdtp(\psi) \triangleq \forall s : \overleftrightarrow{\mathbb{T}}, i : I.\ i \vDash_{[b,e]} \psi \Rightarrow i_s \vDash_{[s(b),s(e)]} \psi$$

In the above and following definitions, it might help to keep the following things in mind. An interpretation describes the behaviour of a system as time passes. If i represents the behaviour of the system, then i_s represents the behaviour of the system when it has been slowed down or speeded up. Suppose the system (an interpretation) satisfies a formula (a property) over some interval. In the case above, the property was said to be slowdown truth preserving if whenever the system is slowed down, the slowed down system still satisfies the property *over the slowed down interval.*

A similar definition can be given for speedup truth preservation. We say that a formula ψ representing a property is said to be *speedup truth preserving* if whenever the system satisfies the property, speeding the system up cannot break the property.

Definition 43. *A formula ψ is said to be speedup truth preserving if whenever one has an interpretation i which satisfies ψ over an interval $[b, e]$,* any *speeded up interpretation i_s, will satisfy ψ over the stretched sub interval. This can be stated as follows.*

$$sdtp(\psi) \triangleq \forall s : \overrightarrow{\mathbb{T}}, i : I.\ i \vDash_{[b,e]} \psi \Rightarrow i_s \vDash_{[s(b),s(e)]} \psi$$

We give an example of the above properties below.

Example 82. *Suppose we have an interpretation which satisfies the formula $\int a \geq 3$, i.e. between b and e, the symbol a is active for at least 3 time units. This property is slowdown truth preserving, since slowing down the interpretation by any amount will only*

mean that this region is stretched and that a will still be active for 3 or more units.

Similarly, suppose that our interpretation satisfies the formula $\int a \leq 3$*, such that symbol a is active for less than 3 time units during the observation interval. Speeding up this interpretation means that the observation interval shrinks. This can only mean that the total time for which a is active also shrinks. Thus, the formula is speedup truth preserving.*

We now pass to the falsity preservation properties.

Slowdown/Speedup Falsity Preserving

Falsity preservation is similar to truth preservation, however it talks about when the system does *not* satisfy the property. Basically, a formula ψ representing a property is said to be *slowdown false preserving* if whenever the system violates the property, it will keep violating it even if the system is slowed down.

Definition 44. *A formula* ψ *is said to be slowdown false preserving if whenever one has an interpretation* i *which violates* ψ *over an interval* $[b, e]$*, then for* any *slowed down interpretation* i_s*,* i_s *will also violate* ψ *on the stretched sub interval.*

$$sdfp(\psi) \triangleq \forall s : \overleftrightarrow{\mathbb{T}}, i : I.\ i \not\models_{[b,e]} \psi \Rightarrow i_s \not\models_{[s(b),s(e)]} \psi$$

Again one has a similar concept applying for *speedup false preserving* formulas. In this case, a formula ψ representing a property is said to be *speedup false preserving* if whenever the system violates the property, it will keep violating it even if the system is speeded up.

Definition 45. *A formula* ψ *is said to be speedup false preserving if whenever one has an interpretation* i *which violates* ψ *over an interval* $[b, e]$*, then for* any *speeded up interpretation* i_s*,* i_s *will also violate* ψ *on the stretched sub interval.*

$$sufp(\psi) \triangleq \forall s : \overrightarrow{\mathbb{T}}\overleftarrow{}, i : I.\ i \not\models_{[b,e]} \psi \Rightarrow i_s \not\models_{[s(b),s(e)]} \psi$$

We give an example below.

Example 83. *Suppose we have an interpretation which does not satisfy the formula* $\int a \geq 3$*, i.e. between b and e, that is, the symbol a is not active for at least 3 time units. This property is speedup false preserving, since speeding up the interpretation by any amount will only mean that the total time for which a is true shrinks. Thus, any slowed down version of i will still never be able to satisfy the property.*

Similarly, suppose that an interpretation does not satisfy the formula $\int a \leq 3$*, such that symbol a is active for more than 3 time units during the observation interval. Slowing down this interpretation can only mean that the total time for which a is active increases. Thus, any slowed down interpretation will never be able to satisfy the property, and the formula is slowdown false preserving.*

As a side note, we make the observation that $\int a \geq 3$ is *both* slowdown truth preserving *and* speedup false preserving. Similarly, $\int a \leq 3$ is *both* speedup truth preserving *and* slowdown false preserving. This is no coincidence; rather, it is a consequence of a general law which we shall expose soon.

The above concepts lead us directly to the notion of Invariance.

Speedup/Slowdown Invariance

A formula/property ψ is said to be *slowdown invariant* if it is both slowdown truth preserving and slowdown false preserving.

Definition 46.

$$sdi(\psi) \triangleq sdtp(\psi) \wedge sdfp(\psi)$$

Similarly, a duration formula ψ is said to be *speedup invariant* if it is both speedup truth preserving and speedup false preserving.

Definition 47.

$$sui(\psi) \triangleq sutp(\psi) \wedge sufp(\psi)$$

If a formula is both slowdown invariant and speedup invariant, we shall simply call it *Invariant*.

Definition 48.

$$inv(\psi) \triangleq sui(\psi) \wedge sui(\psi)$$

The theory of slowdown and speedup also contains several meta theorems which relate the concepts above. We shall discuss, and provide proofs for some of them below.

Speedup and Slowdown Meta Theorems

A very important contribution as regards to slowdown/speedup truth/falsity preservation are the proofs of several meta theorems relating the four concepts described above together. These are very useful since every time we prove a result for a particular formula, essentially we get another result for free. We list two of these results below, which we refer to as the *switch meta theorems*. The first result tells us that a formula that is slowdown truth preserving, *has to be* speedup false preserving and vice versa. We have already seen this phenomenon for the operator $\int a \geq k$.

Theorem 84. *Let ψ be a formula of a logic with interpretation semantics. Then,*

$$sdtp(\psi) \Leftrightarrow sufp(\psi)$$

We shall show the proof for this result here, instead of in the appendix, since its structure will be relevant later on. The proofs for the meta theorems are all due to Colombo et al. (See [CPS][Col08]). We first prove the forward direction, that is

$$sdtp(\psi) \Rightarrow sufp(\psi)$$

This means that if we assume $sdtp(\psi)$, we have to show that

$$i \not\models_{[b,e]} \psi \Rightarrow i_s \not\models_{[s(b),s(e)]} \psi$$

where s is a speedup transform.

Proof.

$$\begin{array}{ll} & i \not\models_{[b,e]} \psi \\ = & \{ \text{ An interpretation does not change under id as a transform } \} \\ & i_{id} \not\models_{[b,e]} \psi \\ = & \{ \text{ Any function composed with its inverse yields id } \} \\ & i_{s \circ s^{-1}} \not\models_{[s \circ s^{-1}(b), s \circ s^{-1}(e)]} \psi \\ = & \{ \ i_{s_1 \circ s_2} = (i_{s_1})_{s2}, \text{ and definition of } \circ \ \} \\ & (i_s)_{s^{-1}} \not\models_{[s^{-1}(s(b)), s^{-1}(s(e))]} \psi \\ = & \{ \ sdtp(\psi), \ s^{-1} \text{ is slowdown, contrapositive of } sdtp(\psi) \ \} \\ & i_s \not\models_{[s(b),s(e)]} \psi \end{array}$$

For the other direction, if we assume $sufp(\psi)$, we have to show that

$$i \models_{[b,e]} \psi \Rightarrow i_s \models_{[s(b),s(e)]} \psi$$

where s is a slowdown transform.

Proof.

$$\begin{array}{ll} & i \models_{[b,e]} \psi \\ = & \{ \text{ An interpretation does not change under id as a transform } \} \\ & i_{id} \models_{[b,e]} \psi \\ = & \{ \text{ Any function composed with its inverse yields id } \} \\ & i_{s \circ s^{-1}} \models_{[s \circ s^{-1}(b), s \circ s^{-1}(e)]} \psi \\ = & \{ \ i_{s_1 \circ s_2} = (i_{s_1})_{s2}, \text{ and definition of } \circ \ \} \\ & (i_s)_{s^{-1}} \models_{[s^{-1}(s(b)), s^{-1}(s(e))]} \psi \\ = & \{ \ sufp(\psi), \ s^{-1} \text{ is speedup, contrapositive of } sufp(\psi) \ \} \\ & i_s \models_{[s(b),s(e)]} \psi \end{array}$$

The second switch meta theorem tells us that any formula which is speedup truth preserving must also be slowdown false preserving, and vice versa. We have seen an instance of this for the operator $\int a \leq k$. We state it below.

Theorem 85. *Let ψ be a formula of a logic with interpretation semantics. Then,*

$$sutp(\psi) \Leftrightarrow sdfp(\psi)$$

The proof is very similar to that of the first switch meta theorem. We refer to the above two theorems as *switch* theorems because if a formula possesses any two characteristics together, it will also possess the dual two characteristics together. So in a way, if a formula has any 2 characteristics, we can switch these characteristics with their dual. By switching characteristics we mean switching the slowdown and speedup properties and the truth and falsity preservation properties.

An immediate corollary is that due to the bi-implication, if a formula is slowdown or speedup invariant, then it will satisfy the other two and thus be time transform invariant.

Colombo also considers what properties a formula $\neg\psi$ will possess if the formula ψ possesses certain properties. The negation of a formula is defined using the following law.

Definition 49. *Let i be an interpretation and ψ a formula of some logic. Then, $\neg\psi$ can be defined as follows.*

$$i \models_{[b,e]} \neg\psi \triangleq i \not\models_{[b,e]} \psi$$

The meta theorems relating to negation of a formula, tell us that if ψ satisfies some property, then $\neg\psi$ can switch ψ's truth preservation with false preservation and vice versa. There are also theorems relating slowdown and speeedup with negation and others for dealing with how negation is related to different forms of invariance. The results for negation are summarised below.

Theorem 86. *Let ψ be a formula of a logic with interpretation semantics. Then the following apply.*

- $sdtp(\psi) \Leftrightarrow sdfp(\neg\psi)$
- $sutp(\psi) \Leftrightarrow sufp(\neg\psi)$
- $sdtp(\psi) \Rightarrow sutp(\neg\psi)$
- $sutp(\psi) \Rightarrow sdtp(\neg\psi)$
- $sdi(\psi) \Leftrightarrow sdi(\neg\psi)$
- $sui(\psi) \Leftrightarrow sui(\neg\psi)$

This brings to an end our exposition of the theory. In the next section, we summarise what we have learnt about the theory of speedup and slowdown.

4.2.3 Conclusion

We can now conclude the summary of Colombo et al.'s Theory of slowdown and speedup. We have seen that the theory is made up of three elements, which are the notion of a time transform on interpretations, the formulas which define what slowdown/speedup truth/falsity preservation mean, and the meta theorems which simplify the application of the theory to interpretation based logics. In the next section we shall put this concept to use by applying this theory to derive speedup and slowdown results for timed regular expressions with interpretation semantics.

4.3 Slowdown and Speedup Results for TRE

4.3.1 Introduction

One of the tasks set in the introduction of this work was to derive slowdown and speedup results for timed regular expressions. We had also determined that in order to do this, we were to use Colombo et al.'s theory of slowdown and speedup. In order to use Colombo et al.'s definition of time transforms, which are based on interpretations, we had to give a new interpretation based semantics to TRE, which we proved to be consistent with respect to Asarin et al.'s semantics.

We are now in a position to apply the theory of slowdown and speedup to derive these results. In order to have a complete picture, we must prove which fragments of the logic are slowdown/speedup truth preserving and false preserving. Our strategy will be to derive the slowdown/speedup truth preservation results from scratch. Then, our choice to use the interpretation definition of time transforms will pay off. By using the meta theorems described in section 4.2.2, we will be able to derive the falsity preservation results as immediate consequences of the truth preservation results. In the next section, we shall begin by describing the technique we will need to prove these results.

4.3.2 Slowdown and Speedup Results

How should we prove which TRE satisfies which results? We cannot possibly examine every timed regular expression. One possible approach is to use again structural induction. First of all, we will check which properties are satisfied by the base cases. Then, given some particular property (ex. slowdown truth preservation), we shall prove which operators preserve that property given that their components satisfy it.

If we approach the problem naïvely we would have to do this process four times in all, once for every trait (slowdown/speedup truth/falsity preservation). Of course, in reality we can simplify this process considerably using Colombo et al.'s meta theorems. We shall start with the proofs for slowdown truth preservation. However, before we can do this, we need to prove the following Lemma.

Lemma 87. *Consider some particular time transform s. If for every time point t between b and e, a transformed interpretation is true for some symbol at the transformed point $s(t)$, then it is also true for that symbol between $s(b)$ and $s(e)$. Let s be a time transform, and i and interpretation. Then,*

$$(\forall t : \mathbb{T}.\ b \leq t \leq e \Rightarrow i_s(\alpha, s(t))) \Rightarrow (\forall t' : \mathbb{T}.\ s(b) \leq t' \leq s(e) \Rightarrow i_s(\alpha, t'))$$

For practical purposes, the effect of this lemma is to allow us to remove $s(t)$ from inside the interpretation and apply it to the range by performing a change of variable, as long as the requirements stated are satisfied.

We can now start to prove which fragment of TRE is slowdown truth preserving. The general statement that we must prove, is

$$\forall i : I, s : \overleftrightarrow{\mathbb{T}}\ .i \vDash_{[b,e]} \psi \Rightarrow i_s \vDash_{[b,e]} \psi$$

We shall break this down by structural induction. We begin with the base case. The statement which must be shown in order to determine that the timed regular expression ($a \in \Sigma$) is slowdown truth preserving follows.

Theorem 88. *a is slowdown truth preserving. Let i be an interpretation, and s a time transform. Then,*

$$i \vDash_{[b,e]} a \Rightarrow i_s \vDash_{[s(b),s(e)]} a$$

The interested reader is referred to the proof in the appendix. Once one knows that the base case is slowdown truth preserving, one can perform the inductive step for all the operators. It can be shown that the $\vee$ operator is slowdown truth preserving, assuming its operands are also slowdown truth preserving.

Theorem 89. *If ψ_1 and ψ_2 are slowdown truth preserving, so is $\psi_1 \vee \psi_2$. Let s be a slowdown transform.*

Assuming

$$sdtp(\psi_1)$$

and

$$sdtp(\psi_2)$$

we can conclude that

$$i \vDash_{[b,e]} \psi_1 \vee \psi_2 \Rightarrow i_s \vDash_{[s(b),s(e)]} \psi_1 \vee \psi_2$$

Similarly to $\vee$, $\wedge$ is slowdown truth preserving.

Theorem 90. *If ψ_1 and ψ_2 are slowdown truth preserving, so is $\psi_1 \wedge \psi_2$. Let s be a slowdown transform.*

Assuming

$$sdtp(\psi_1)$$

and

$$sdtp(\psi_2)$$

we can conclude that

$$i \vDash_{[b,e]} \psi_1 \wedge \psi_2 \Rightarrow i_s \vDash_{[s(b),s(e)]} \psi_1 \wedge \psi_2$$

TRE Concatenation, can also be shown to be slowdown truth preserving.

Theorem 91. *If ψ_1 and ψ_2 are slowdown truth preserving, so is $\psi_1 \circ \psi_2$. Let s be a slowdown transform.*

Assuming

$$sdtp(\psi_1)$$

and

$$sdtp(\psi_2)$$

we can conclude that

$$i \vDash_{[b,e]} \psi_1 \circ \psi_2 \Rightarrow i_s \vDash_{[s(b),s(e)]} \psi_1 \circ \psi_2$$

We now consider the result for the interval restrict operator. What one will find out, is that this operator, in general, is not slowdown truth preserving. We also note that this is the only operator with which a timing constraint can be enforced on a timed regular expression. Does this mean that whenever we have a timing constraint, we cannot guarantee slowdown truth preservation? Is the only fragment of logic which is slowdown truth preserving the trivial one where we cannot enforce requirements on the length of the segments of a signal?

This result, viewed from this angle, is devastating, because it would make TRE useless in the regard of providing guarantees under system slowdown. However, although the interval restrict operator, in general, is not slowdown truth preserving, there are interesting special cases where it is. It can be shown that over the interval $[c, \infty)$, for any c, it is slowdown truth preserving. Why is this so? If a signal interpretation satisfies an interval restricted regular expression with interval $[c, \infty)$, this means that the length of its observation interval is at least c. It is thus not difficult to see that slowing it down, will only increase the length of this interval, such that it will still satisfy the regular expression. We state this result below.

Theorem 92. *If ψ is slowdown truth preserving, so is $\langle\psi\rangle_{[c,\infty]}$. Let s be a slowdown transform.*

Assuming

$$sdtp(\psi)$$

then

$$i \vDash_{[b,e]} \langle\psi\rangle_{[c,\infty]} \Rightarrow i_s \vDash_{[s(b),s(e)]} \langle\psi\rangle_{[c,\infty]}$$

Why isn't the general case of interval restriction not slowdown truth preserving? A counter example can occur whenever the restriction interval has an upperbound, or in other words, then the interval is of the form $[0, d]$ for some finite d. This occurs because, if a signal interpretation satisfies such a TRE, then the length of its observation interval must be at most d. However, there will be slowdown functions which will increase the length of the observation interval beyond d, such that it is not characterised anymore by the TRE. We formulate this statement below.

Theorem 93. *If ψ is slowdown truth preserving, $\psi_{\langle[0,d]\rangle}$ is* not*, in general, slowdown truth preserving.*

We now prove that the star operator is closed under slowdown truth preservation. Before we can do this, however we shall prove the following lemma about exponentiation. We shall find out that if a TRE is slowdown truth preserving, then, that expression raised to any power (including zero) is also slowdown truth preserving. This statement can be proved by induction on n.

Lemma 94. *If ψ is slowdown truth preserving, so is ψ^n.*

$$\forall n : \mathbb{N}.\ sdtp(\psi) \Rightarrow sdtp(\psi^n)$$

Using the lemma above, it becomes easy to prove that the star operator is also slowdown truth preserving. This is because when an interpretation is characterised by some TRE raised to the star, this means that interpretation is in fact characterised by the TRE raised to some n.

Theorem 95. *If ψ is slowdown truth preserving, so is ψ^*. Let s be a slowdown transform.*

Assuming

$$sdtp(\psi)$$

then

$$i \vDash_{[b,e]} \psi^* \Rightarrow i_s \vDash_{[s(b),s(e)]} \psi^*$$

We shall conclude the slowdown truth preservation results by stating that when a TRE is slowdown truth preserving, negating the TRE is not necessarily slowdown truth preserving. The argument is a little intricate, and the reader would do well to consult the proof. However, the intuition behind it is as follows. Suppose we choose, as ψ, some interval restricted expression with no upper bound and a lower bound c. By Theorem 92, we know that this is slowdown truth preserving. Now, an interpretation satisfying $\neg\psi$, does not satisfy ψ, so it must have an observation interval of length less than c. If we slow this interpretation down, there is a chance of it exceeding c, which means that it would satisfy ψ, and thus, not satisfy $\neg\psi$, breaking the slowdown truth preservation statement. We state this result below.

Theorem 96. *Suppose that ψ is slowdown truth preserving. Then, $\neg\psi$ is not necessarily slowdown truth preserving.*

In the next section, we shall present similar results for speedup truth preservation.

4.3.3 Speedup Truth Preserving Operators

To prove that a TRE is speedup truth preserving, we insist that if an interpretation is characterised by that TRE, then it will still be characterised by that TRE under any speedup transform. In general, we can express this requirement as

$$\forall i : I, s : \overrightarrow{\mathbb{T}}^{\leftarrow} \ . \ i \vDash_{[b,e]} \psi \Rightarrow i_s \vDash_{[s(b),s(e)]} \psi$$

Most of the proofs are identical to those for slowdown truth preservation, with s being a speedup transform instead of a slowdown transform. To this end we shall not reproduce them here.

The only difference in the fragment of the logic that is speedup truth preserving, with respect to the fragment that is slowdown truth preserving lies in the interval restrict operator. This is to be expected since it is the only operator which affects time in the TRE formalism. In this section we shall prove the results relating to the interval restrict operator as they relate to speedup truth preservation. We shall also produce a different proof for negation, since the argument for negation will change in this case.

First we prove that interval restriction over an interval $[0, d]$, for some finite d is speedup truth preserving. The intuition behind this is that if an interpretation obeys the

TRE, then the period of observation must have length of at most d. If we speedup by any amount, we are reducing the length of the period of observation, so it will still be characterised by the TRE.

Theorem 97. *If ψ is speedup truth preserving, so is $\langle\psi\rangle_{[0,d]}$. Let s be a speedup transform.*

Assuming

$$sutp(\psi)$$

we can conclude that

$$i \vDash_{[b,e]} \langle\psi\rangle_{[0,d]} \Rightarrow i_s \vDash_{[s(b),s(e)]} \langle\psi\rangle_{[0,d]}$$

We now show why the general case of interval restriction is not speedup truth preserving. This occurs whenever the interval has a lowerbound, or in other words, when the interval is of the form $[c, \infty]$ for some finite c. This occurs because, if an interpretation satisfies the TRE, the length of its observation interval must be at least c. However, there will be speedup functions which will decrease the length of the signal beyond c, such that it is not characterised anymore by the TRE. We state the result below.

Theorem 98. *Assuming*

$$sdtp(\psi)$$

then

$$\psi_{\langle[c,\infty]\rangle}$$

is, in general, not speedup truth preserving.

We conclude by showing that the negation operator is not speedup truth preserving. The reader should examine the proof, but we summarise the intuition behind it here. Suppose we choose, as ψ, some interval restricted expression with an upper bound d and no lower bound. By Theorem 97, we know that this is speedup truth preserving. Now, an interpretation satisfying $\neg\psi$, does not satisfy ψ, so it must have an observation interval of length greater than d. If we speed this interpretation up, there is a chance of it getting reduced to less than d, which means that it would satisfy ψ, and thus, not satisfy $\neg\psi$, breaking the speedup truth preservation statement. We state this result below.

Theorem 99. *Suppose that ψ is speedup truth preserving. Then, $\neg\psi$ is not necessarily speedup truth preserving.*

In the next section, we decide which operators are slowdown false preserving.

4.3.4 Slowdown False Preserving Operators

Once we know which operators are speedup truth preserving and which are not, it becomes trivial to determine which are slowdown false preserving and which are not. This is an immediate consequence of Theorem 85 (switch meta theorem two), which states that if a formula is speedup truth preserving, then it is slowdown false preserving and vice versa. We thus summarise false preservation in the following corollary.

Corollary 100. *Let ψ, ψ_1 and ψ_2 be slowdown false preserving. Then all of the following are slowdown false preserving.*

- $a \in \Sigma$
- $\langle \psi \rangle_{[0,e]}$
- $\psi_1 \vee \psi_2$
- $\psi_1 \wedge \psi_2$
- $\psi_1 \circ \psi_2$
- ψ^*

Also, the following are therefore not slowdown false preserving.

- $\langle \psi \rangle_{[b,\infty]}$
- ψ^c

In the next section we show which operators are speedup false preserving.

4.3.5 Speedup False Preserving Operators

Similarly once we know, which operators are slowdown truth preserving and which are not, it becomes trivial to determine which are speedup false preserving and which are not. This is an immediate consequence of theorem 84 (switch meta theorem number one), which states that if a formula is slowdown truth preserving, then it is speedup false preserving, and vice versa.

Corollary 101. *Let ψ, ψ_1 and ψ_2 be speedup false preserving formulas. Then all of the following are speedup false preserving.*

- $a \in \Sigma$
- $\langle \psi \rangle_{[b,\infty]}$
- $\psi_1 \vee \psi_2$
- $\psi_1 \wedge \psi_2$
- $\psi_1 \circ \psi_2$
- ψ^*

Also, due to the fact that the meta theorem is a bi-implication, the following are not speedup false preserving.

- $\langle \psi \rangle_{[0,e]}$
- ψ^c

We can now combine all our results in order to decide which operators are invariant.

4.3.6 Invariance Theorems

When a formula is invariant, it is not affected by neither the slowing down nor the speeding up of the system. Not surprisingly, the operators of the logic that are invariant are precisely those which do not include the interval restriction operator in some form or another. We can collect all the invariance results in the corollary below. One should take note the of the assumption that is made in the statement below; These operators are invariant if their operands are also invariant.

Corollary 102. *Let ψ, ψ_1 and ψ_2 be invariant under time transforms. Then all of the following are invariant.*

- $a \in \Sigma$
- $\psi_1 \vee \psi_2$
- $\psi_1 \wedge \psi_2$
- $\psi_1 \circ \psi_2$
- ψ^*

So far we have avoided mention of the negation operator, as regards invariance. We know that negation is, *in general*, neither slowdown, nor speedup truth preserving. By using the switch meta theorems, it is also neither slowdown nor speedup truth preserving. However, we shall now find out that negation is in fact invariant, as long as one makes the strong assumption that its premises are also invariant. This means that its operands cannot include the interval restriction operator.

Corollary 103. *Let ψ be invariant under time transforms. Then $\neg\psi$ is also invariant.*

This is an immediate consequence of theorem 86 (negation meta theorem)

$$invariant(\psi) \Rightarrow invariant(\neg\psi)$$

which tells us the behaviour of the traits under negation. We shall have more to say about this result in the conclusion section below.

4.3.7 Conclusion

In this section we have checked which operators satisfy certain slowdown/speedup traits. We summarise all of the results for the operators in the following table. We always assume that if the operator has certain operands, then these operands all satisfy the property that we are checking for.

	sdtp	sutp	sdfp	sufp	invariant
a	√	√	√	√	√
$< \psi >_{[b,\infty]}$	√	x	x	√	x
$< \psi >_{[0,e]}$	x	√	√	x	x
$\psi_1 \cup \psi_2$	√	√	√	√	√
$\psi_1 \cap \psi_2$	√	√	√	√	√
$\psi_1 \circ \psi_2$	√	√	√	√	√
ψ^*	√	√	√	√	√
$\neg\psi$	x	x	x	x	√

What can we observe from this table? If we disregard negation, the problematic operator seems to be interval restriction. This is to be expected, since it is the only operator which can impose time constraints. What is the extent of the problem created by this operator? If we look at the invariance column, we immediately note that the moment in which an interval restrict operator appears, we can forget about our properties having immunity from the problems introduced by adding or removing monitors. Let us examine the two cases of the interval restriction operator. We note that if we have interval restriction with an upperbound, we cannot have slowdown truth preservation. On the other hand, if we have interval restriction with a lowerbound, we lose speedup truth preservation. If we have both an upperbound, and a lowerbound, we lose both. Thus, the best we can hope for is to be able to write interesting temporal properties using just upperbounds or just lowerbounds. Aside from this, all the other operators do not affect whether a formula satisfies one of the four properties or not. The other interesting case is negation. Note that if no time constraints are involved, negation is invariant since in this case, its premises are also invariant. However, if one looks at the negation proofs, one will note that negation cannot, in general, guarantee that a property is preserved even if the formula being negated satisfies the property. The difficulty arises since negating formulas with interval restrict operators with a certain bound, will effectively create a formula with the opposite type of bound and will thus invalidate the property satisfied by the original formula.

We thus conclude the derivation of slowdown and speedup results for TRE. One should note that obtaining these results has involved a lot of work; we had to give an interpretation based semantics to TRE, and had to prove soundness and completeness with respect to the old semantics, before we could even start to apply the theory of slowdown and speedup to TRE. In the next section we shall see that if we had a theory of slowdown and speedup which works on signals, all this could have been avoided, since we could have applied that directly to TRE to derive the slowdown and speedup results.

4.4 A Theory of Slowdown and Speedup Using Signals

4.4.1 Introduction

In this section, we shall trace the major steps involved in devising another theory of slowdown and speedup, which uses on signal lists as its underlying model. Why should one define a new theory? As we have seen in previous sections, deriving slowdown and speedup results for TRE has been rather involving. We first had to endow TRE with a new interpretation based semantics. Then, we had to provide soundness and completeness proofs for the two different semantics. It was only then, that we could actually apply the theory to get the required results. If we had a new theory based on signals, then we could derive results directly for any new logic which also has its semantics grounded in signals, such as the timed automata defined in [ACM97]. Another reason why it would be useful to have this theory is to be able to use it to derive slowdown and speedup results for TRE in another way, thus acting as a sanity check for our, rather complex, original approach.

What would such a theory involve? In the concluding remarks of section 4.2, we identified three elements which made up Colombo et al.'s theory of slowdown and speedup; a definition for time transforms on interpretations, a definition of what slowdown/speedup truth/false preserving traits mean, and a set of meta theorems relating the traits to one another. What work would be required in order to convert these aspects to work for signal lists instead of for interpretations?

The crucial component for the new theory is to define how time transforms affect signal lists. This requires a definition of how a signal list relates to its transformed version. Since we are now studying how time transforms interact with signal lists, we would also need to prove that the identity time transform does not affect a signal list, and that applying two time transforms after each other on a signal list is the same as applying their composition on the signal list. The definitions for slowdown/speedup truth/false preservation remain unchanged, except that interpretations are replaced by signal lists, and that the formulae now represent TRE which work using signal list semantics. More generally (i.e. for other logics) this would require that the semantics of the new logic be formulated in terms of satisfaction on an interval. The meta theorems, while nice to have, are not strictly necessary. However, once we have lemmas about how signal lists relate to the identity function and to composition of time transforms, the meta theorems can be proved in a way very similar to that shown in section 4.2 and in Colombo et al.'s work [CPS][Col08].

Once we have such a theory, the only thing that would remain is to prove that its definition is in fact consistent with the definition given by Colombo et al. How can one do this? What we would have to prove is that if we start with a signal, transform it (using the definition of time transforms on signals), and convert it into an interpretation, we would always get the same interpretation as if we had first converted the signal and then transformed it (using the definition of time transforms on interpretations).

After having considered all this, we shall focus on giving the definition for time transforms on signal lists and proving that it is sound with respect to that on interpretations. The other parts of the theory of slowdown and speedup can be easily adapted from the original theory, and to this end we shall not treat them.

4.4.2 Time Transforms on Signal Lists

The first thing we shall do is define how a time transformed signal list relates to the original one, similarly to what we have done with interpretations in Definition 39. Remember that a signal list is a list of tuples. To this end, we cannot universally quantify over all time points, as we have done with interpretations. What we shall do is define the application of a time transform on a signal list by means of a recursive rule.

Definition 50. *let ξ be a signal list, and s a time transform. We define the application of a time transform on a signal list using the following rules.*

$$\langle\rangle_s \triangleq \langle\rangle$$

$$(\xi' : (\alpha, k))_s \triangleq \xi'_s : (\alpha, s(k + |\xi'|) - |\xi'_s|)$$

The first rule states that transforming an empty signal list, does not affect it. The second rule states that applying a time transform to a signal results in transforming the last tuple, and recursively transforming the rest of the signal. How is the tuple at the end of the list transformed? Remember that in the concluding remarks of section 2.2 we had said that a signal list is a relative view of a signal. By this we mean that each tuple only contains its duration, it only knows its location in time relative to all the other previous tuples. On the other hand, we said that an interpretation is an absolute view. Also recall that a time transform works on absolute timescales. Therefore, due to this fact, when we are working with signal lists, in order to transform a segment of a signal, we first have to locate its end points absolutely. We would then transform the endpoints and take the length between them to be the new transformed signal length.

How do we perform this relative-to-absolute conversion? We first generate the absolute point at the end of the original segment by applying s to the sum of the lengths of all previous segments $|\xi'|$, plus the length of the last segment (k). Once we have transformed this point, we get back the transformed length by subtracting the excess part of the result, by eliminating from it the sum of the lengths of all previous transformed segments.

Before we continue, we state a lemma showing how time transforms and the length operator on signal lists interact.

Lemma 104. *The length of a time transformed signal list is equal to transforming the length of the original signal. Let ξ be a signal list. Then the following holds.*

$$|\xi_s| = s(|\xi|)$$

With the definition of time transforms on signal lists complete, we must prove two important propositions which explain what happens when the time transform is the identity function, and when the time transform is a composition of functions. We shall state them below; the proofs can be found in the appendix.

Proposition 105. *A signal list ξ is equal to a transformed list whenever the time transform is the identity function. Let ξ be a signal list. Then,*

$$\xi_{id} = \xi$$

The following result about composition can also be proved by induction.

Proposition 106. *Let i be a signal list, and let s_1 and s_2 be time transforms. Then,*

$$\xi_{s_1 \circ s_2} = (\xi_{s1})_{s2}$$

With these two results, we have all we need to prove meta theorems analogous to the ones in section 4.2. We now pass to our next goal, that of proving the soundness of the two time transform definitions.

4.4.3 Soundness of Time Transform Definitions

Earlier we said that we should prove that the concepts of time transforms on signal lists and that of time transforms on interpretations are consistent with one another. We also said, informally, that we could prove this by showing that if we start with a signal, transform it (using the definition of time transforms on signal lists) and convert it into an interpretation, we would always get the same interpretation as if we had first converted the signal and then transformed it (using the definition of time transforms on interpretations). A similar argument can be used to show that time transforms on interpretations are sound with respect to time transforms on signal lists, by starting with the concept of an interpretation instead. Since we are required to convert from signal lists to interpretations, and from interpretations to signal lists, we shall have to employ again the constructions defined in section 2.3. Remember that due to the way we have defined our constructions, in order to convert an interpretation i to a signal ξ, we must call $c2s(i2c(i))$. On the other hand, to convert a signal ξ to an interpretation i, we must use $c2i(s2c(\xi))$.

So, if we want to prove that signal list semantics are sound with respect to signal interpretation semantics, we should prove that

$$c2i(s2c(\xi_s)) = c2i(s2c(\xi))_s$$

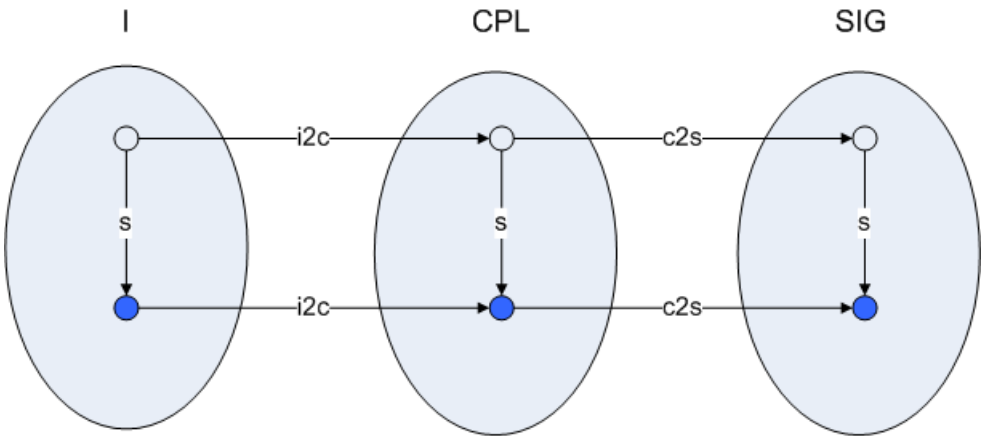
And if we want to prove that interpretation semantics are sound with respect to signal list semantics, we would show that

$$c2s(i2c(i_s)) = c2s(i2c(i))_s$$

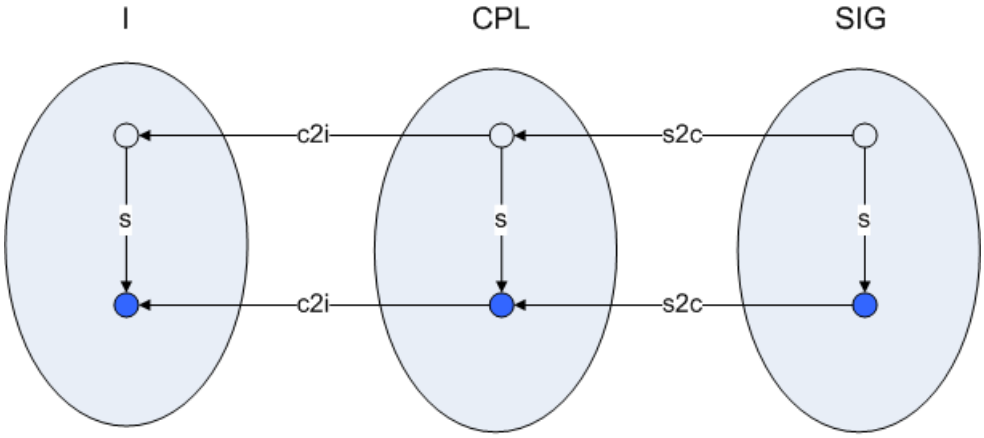
Also note that due to the way the constructions are defined, we always have to pass through the critical point list model. This, however, can prove to be an advantage. We can split the problem into two pieces. If we had a concept of slowdown on critical point lists, we could prove that the concept of slowdown on signal lists is sound with respect to the concept of slowdown on critical point lists. Then, we would prove that the concept of slowdown on critical point lists is sound with respect to the concept of slowdown on interpretations. Obviously, a similar argument holds if one starts with interpretations.

To this end, to prove that the concepts of time transforms on signal lists and that of time transforms on interpretations and are sound with respect to one another, we would need four proofs in total. This approach is simpler and more clear. This is summarised in figure 4.2.

As we shall see in the next section, there are several details that will have to be worked out before we can proceed to the actual soundness proofs.



Soundness of Time Transforms based on Interpretations with respect to Time Transforms based on Signal Lists (via CPL)



Soundness of Time Transforms based on Signal Lists with respect to Time Transforms based on Interpretations (via CPL)

Figure 4.2: Splitting soundness proofs into two pieces

4.4.4 Preliminary Definitions and Proofs

Due to the fact that we must now combine the constructions of section 2.2 with the concept of time transforms, we shall need to devise some additional mathematical machinery. We shall first prove some propositions about how the operator C() interacts with the concept of a time transform. Recall that the C() operator yields the critical points of an interpretation, and is the basis of the function $i2c$. Using this concept as inspiration, we shall have to define the concept of a time transform on critical point lists. Finally, we shall also need some additional work to reconcile time transforms with the $\overset{\text{I}}{;}$ operator. This shall be necessary since the function $c2i$ uses this operator; and we shall want to distribute time transforms through it in the final soundness proofs. But we are getting ahead of ourselves here. Let us start by considering the how time transforms interact with the critical points of an interpretation.

Propositions on Time Transforms and Critical Points

We shall now consider some propositions about an interpretation's behaviour under a time transform. Since the interesting behaviour of an interpretation happens at its critical points, it is natural to ask what happens to these points when a time transform is applied. All these propositions can also be viewed as theorems about the behaviour of the operator C().

We start by considering how a time transform affects the values between two successive critical points. What happens is that if an interpretation attains some value between two critical points, then the corresponding interval in the transformed interpretation has the same value. We state this result below.

Proposition 107. *If an interpretation attains a value for a symbol over an interval between two critical points then, that value over the transformed interval is unchanged. Let i be an interpretation. Formally, we can say the following. Consider its critical point list, obtained by ordering C(i). Take any two consecutive critical points c_i and c_{i+1}. If $i(\alpha, t) = true$ for all points in $[c_1, c_2)$, then $i_s(\alpha, s(t)) = true$ for all points in $[s(c_1), s(c2))$. The same result holds for $i(\alpha, t) = false$.*

We can also prove that transforming an interpretation using a time transform s, remaps all its critical points x to $s(x)$.

Proposition 108. *The set of critical points of a transformed interpretation consists of the set of transformed critical points (of the original interpretation). Let C(i) be the set of critical points of i, and let s be a time transform. Then,*

$$C(i_s) = \{s(x) \mid x \in C(i)\}.$$

Next we state a simple but very useful proposition about the set of critical points of an interpretation and the set of transformed critical points.

Proposition 109. *The largest (last) critical point of an interpretation i, when transformed, remains the largest (last) of i_s. We can state this fact as*

$$s(max(C(i))) = max(C(i_s))$$

This result gives the following corollary.

Corollary 110. *The length of a transformed interpretation is equal to transforming the length of the original interpretation. Let i be an interpretation. Then, we can state this as $|i_s| = s(|i|)$.*

We shall now define how time transforms operate on critical point lists.

Time Transforms on Critical Point Lists

Note that when we defined time transforms on interpretations, we said that the value of an interpretation at an untransformed point is the same as the value of the interpretation at the transformed point. Since that statement holds at all points, it also holds at the critical points. Since the concept of critical point lists is closely connected to that of interpretations, we shall use the remark we have just made as a guide to our definition.

Definition 51. *Let xs be a critical point list and s be a time transform. We define the application of s upon critical lists using the following rules.*

$$\langle (0, \bot) \rangle_s = \langle (0, \bot) \rangle$$
$$(xs : (t, \alpha))_s = (xs_s) : (s(t), \alpha)$$

We shall now state a Lemma which allows us to obtain the final value of applying a time transform on a critical point list.

Lemma 111. *Consider a critical point list $xs = \langle (c_1, \alpha), \ldots, (c_n, \bot) \rangle$. Then the effect of applying a time transform to it can be characterised by the following statement. $xs_s = \langle (s(c_1), \alpha), \ldots, (s(c_n), \bot) \rangle$.*

We now prove how time transforms and the length operator over CPL relate.

Lemma 112. *The length of a transformed critical point list is equal to transforming the length of the original critical point list. Let xs be a critical point list. Then,*

$$|(xs)_s| = s(|xs|)$$

This concludes our definition regarding time transforms on critical point lists. We now define an extension to time transforms, which we call shifted time transforms. This shall be critical to reason about how time transforms interact with the $\overset{\text{I}}{;}$ operator.

Shifted Time Transforms

We have already seen that a time transform is a function which remaps all time points with some special constraints. However, sometimes we want to apply the transform to an interpretation by beginning the time transform at some point other than 0. We call such an application of a time transform a *shifted time transform*. What do we mean by what we have said? Let us consider transforming a point t. Normally, t would be mapped to s(t) under a time transform s. Suppose however, that we would like to begin the transform at the point t = 2. What this would mean would be that t would be mapped to s(t+2).

We shall call the constant k a *shift*, and we shall write a time transform with shift k as $s \dashrightarrow k$. This means that before applying the transform to a point, we will first shift that point by k (add k to it).

This raises an issue. Consider, for illustration the singleton interpretation

$$\begin{cases} i(\alpha, t) = \text{true for } 0 \leq t < 5 \\ i(\alpha, t) = \text{false for } 5 \geq t \\ i(\beta, t) = \text{false for } t \geq 0 \end{cases}$$

What is the shape of the interpretation $i_{s \dashrightarrow 2}$? s(0) gets mapped to s(0+2), i.e. s(2), whilst 5 will get mapped to some other point greater or equal to 5+2 = 7. Essentially the problem is that this is not a correct time transform, since s(0) does not get mapped

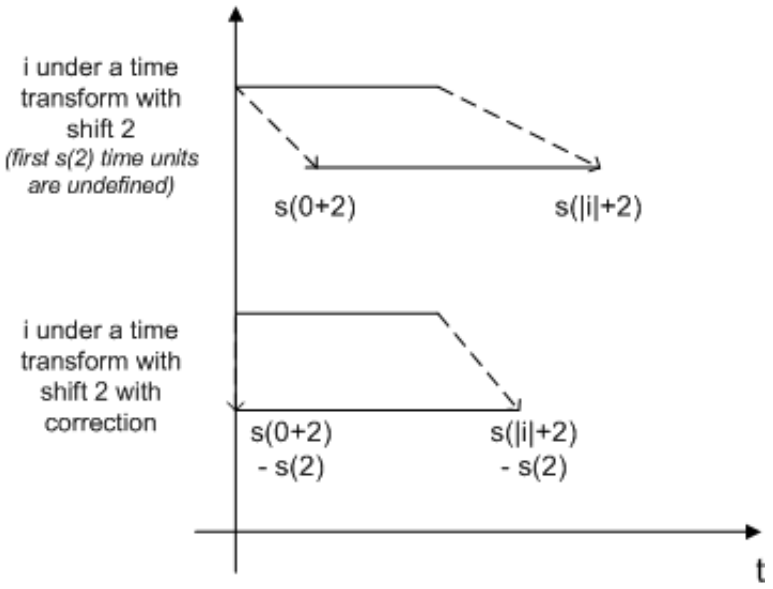


Figure 4.3: The Problem With Shifted Time Transforms

to 0. In a way we have the first s(2) time units undefined under the transform, which is clearly not our purpose. We can envisage this problem in figure 4.3.

To cater for this, we subtract these missing units, after the transformation, from all time points. By doing so we have achieved our goal of transforming our interpretation by beginning the time transform at a later point in time.

We shall now define how an interpretation relates to itself under a shifted time transform via the following logic formula.

Definition 52. *The relationship between an interpretation i and its shifted time-transformed version with shift k, $i_{s\dashrightarrow k}$ is given by the following law.*

$$\forall t : \mathbb{T}.\ i_{s\dashrightarrow k}(\alpha, s(t+k) - s(k)) = i(\alpha, t)$$

This concludes the definition for shifted time transforms. In the next section we shall be able to appreciate why we need this concept.

Time Transforms and the $\overset{\text{I}}{;}$ Operator

In this section we shall see that The $\overset{\text{I}}{;}$ operator is problematic when it comes to time transforms. However, we shall first state a Lemma which allows us to manipulate a certain feature of time transforms. This lemma states that if we manage to prove that for every time point t, and symbol α, two interpretations agree in value at time s(t), then they agree in value at all values of t.

Lemma 113. *If two interpretations agree in value at every s(t) and every symbol, then they agree in value for every time point t. Let i and j be interpretations. Then,*

$$\forall \alpha : \Sigma, t : \mathbb{T}.\ i(\alpha, s(t)) = j(\alpha, s(t)) \Rightarrow \forall \alpha : \Sigma, t : \mathbb{T}.\ i(\alpha, t) = j(\alpha, t)$$

We now return to the question of why it is not simple to combine $\overset{\text{I}}{;}$ with time transforms. Consider the interpretation

$$i \overset{\text{I}}{;} j$$

A time transform s, does not distribute through this operator; that is, we *cannot* assert that:

$$(i \overset{\text{I}}{;} j)_s = i_s \overset{\text{I}}{;} j_s$$

This occurs because of the way in which we have chose to define $i \overset{\text{I}}{;} j$. When working out the value of this expression at some particular time point and for some particular symbol, what we do is essentially check whether this value is smaller that $|i|$. If it is, we consult the value of i at those variables. If t is equal to or exceeds this value, then, we subtract $|i|$ from the time point and consult the value of j directly as if i did not exist. We have never *really* appended them to one another, although the definition works *as if* we had appended them.

This however, will create problems when $(i \overset{\text{I}}{;} j)_s$ is considered. A time transform works on an absolute time scale; it maps a specific value to all points, depending on their location. However, due to the way in which $\overset{\text{I}}{;}$ works, it does not give an absolute position to the points of j, because it never really appends the two functions together. Instead, it employs some machinery internally to decide whether the absolute input point is related to the value of i or j. Both i and j still begin at 0. As an illustration of what happens, consider the following. i has its last critical point at $|i|$, and j its last critical point at $|j|$. If we appended them together, $i \overset{\text{I}}{;} j$ would have its last critical point at $|i| + |j|$. If we applied the time transform to this point, it would clearly get mapped to $s(|i| + |j|)$. But if we applied the time transform, before applying the chop, we get the following scenario. i's last point would be mapped to $s(|i|)$ and j's last point to $s(|j|)$. When we apply $\overset{\text{I}}{;}$, the last point would be at $s(i) + s(j)$.

It is very easy to come up with a counter example showing that $s(t_1)+s(t_2) \neq s(t_1+t_2)$. So therefore we can conclude that s does not distribute over $\overset{\text{I}}{;}$.

We shall now combine our notion of shifted time transforms with the $\overset{\text{I}}{;}$ operator in order to show that we can obtain an expression in which s distributes over $\overset{\text{I}}{;}$. This is summarised by the following theorem.

Theorem 114. *A time transform distributes over $\overset{I}{;}$ into a time transform and a shifted time transform. Let s be a time transform. Then, the following law holds.*

$$(i \overset{I}{;} j)_s = i_s \overset{I}{;} j_{s \dashrightarrow |i|}$$

We now have all the mathematical machinery we need to attempt to prove that the definitions of time transforms on the structures of interpretations, signal lists and critical point lists are in fact consistent.

4.4.5 Proof of Soundness of Definitions

Let us summarise again the proof strategy we shall adopt in order to prove that the definitions are consistent. Recall that we have to show that the definition of time transforms on signal lists is consistent with the definition of time transforms on interpretations, and vice versa. For each direction we resolved that we would split each proof into two pieces, by first proving that each notion was consistent with the concept of time transforms on critical point lists. Let us thus write down the formulas that we have to prove.

If we want to prove that time transforms on interpretations are sound with respect to time transforms on signal lists, we have to prove the following two formulae, where ξ is a signal list, xs is a critical point list and i is an interpretation.

$$i2c(i_s) = i2c(i)_s$$
$$c2s(xs_s) = c2s(xs)_s$$

On the other hand, if we want to prove that time transforms on signal lists are sound with respect to time transforms on interpretations, we have to show the following two facts.

$$s2c(\xi_s) = s2c(\xi)_s$$
$$c2i(xs_s) = c2i(xs)_s$$

We note that this becomes an exercise in proving that time transforms distribute through the functions which allow us to move between models. For the actual proofs, the reader is pointed to the appendix.

Let us start with the first theorem. The following statement states that the definition of time transforms on interpretation is sound with respect to the definition of time transforms on critical point lists.

Theorem 115. *Time transforms distribute through the function $i2c$. Let i be an interpretation and s a time transform. Then, $i2c(i)_s = i2c(i_s)$*

The second theorem shows that the definition of time transforms on critical point lists is sound with respect to the definition of time transforms on signal lists.

Theorem 116. *Time transforms distribute through the function $c2s$. Let xs be a critical point list, and s a time transform. Then $c2s(xs_s) = (c2s(xs))_s$.*

We now consider the second set of proofs. The next theorem asserts the consistency of the definition of time transforms on critical point lists with that of time transforms on interpretations.

Theorem 117. *Time transforms distribute through the function $c2i$. Let xs be a critical point list and s a time transform. Then, $c2i(xs)_s = c2i(xs_s)$*

Finally, the last proof will allows us to state the consistency of time transforms on signal lists and time transforms on critical point lists.

Theorem 118. *Time transforms distribute through the function $s2c$. Let ξ be a signal in list form and s a time transform. Then $s2c(\xi_s) = (s2c(\xi))_s$.*

We have thus managed to prove the consistency of time transform definitions for signal lists and interpretations, by first showing that each of these definitions is consistent with the concept of time transforms on critical point lists. We shall summarise our achievements for this section in the next section.

4.4.6 Conclusion

In this section, we have seen both the motivation behind defining an alternative theory of slowdown and speedup, and the stages involved in the construction of such a theory. To substantiate this, we have provided the essentials for a theory based on signal lists. We concluded that the crucial things to provide for a new theory is the concept of a time transform on the new underlying model, a proof that identity transforms do not change the structures involved, a proof that composition of time transforms works as expected, and a proof that the new theory is consistent with the standard theory of slowdown and speedup. We have also seen that the proof of consistency is based on proving that time transforms distribute through the functions which convert the new structure to an interpretation and vice versa. In our case, we needed to perform two steps in each direction through the intermediate notion of critical point lists; however this is not necessary if one is able to convert from the new structures to interpretations in just a single step. Despite the relative simplicity of the approach, we also noted that sometimes time transforms might not distribute cleanly through the operators which are used within the functions which allow us to move between models. In our case this required the introduction of the new concept of a shifted time transform.

In the next section we briefly summarise the achievements in this chapter.

4.5 Conclusion

In this chapter we introduced the concepts of slowdown and speedup into the equation. We started by looking at the major elements of the theory of slowdown and speedup as defined by Colombo et al. Amongst these, we became aware of the important concepts of slowdown/speedup truth/falsity preservation. Following this, we applied this theory to derive slowdown and speedup results for timed regular expressions, and completely characterised the logic, in such a way that we know exactly what properties are satisfied by each operator. The major result was that if we attempt to set time constraints with lowerbounds, we cannot be safe from the problems of speedup, whilst if we formulate constraints containing upper bounds, we lose immunity from the problems of slowdown. If we formulate constraints with both upper bounds and lowerbounds, we cannot guarantee safety from either. To this end, we came to the conclusion that the best we can do was to attempt to formulate interesting properties using just upperbounds or just lowerbounds. Finally, we concluded by showing an alternative approach to obtaining speedup and slowdown results for a logic which does not have its semantics based on interpretations. This involved formulating a new theory of slowdown and speedup which was based on the same model as that logic. We investigated the major items that a new theory had to include, using a new theory based on signal lists as an example. We came to the conclusion that the most important parts of such a new theory would be a new definition of slowdown on

the new model and a proof of soundness between the definition of slowdown on the new model and the definition of slowdown on interpretations.

Chapter 5

Concluding Remarks

5.1 Conclusions

We shall start by considering the contributions that this work offers. Firstly, let us recall that in the beginning, we set out to prove which fragment of timed regular expressions satisfied certain speedup and slowdown properties. This objective has been achieved completely. We have a complete characterisation of which properties (slowdown/speedup truth/falsity preservation) each of the operators satisfies. What are the notable elements of these results?

Amongst other things, we noted that the operator which proved problematic was the interval restriction operator. This was to be expected since it is the only operator that introduces time constraints into the TRE formalism. What we saw was that the moment we introduce time constraints, if we specify an upperbound, we will lose immunity from the problems of slowdown, whilst if we specify a lowerbound we lose immunity from the problems of speedup. Thus, the best we can hope for when we use time constraints is to have just one of these useful properties. Moreover, we saw that it is possible not to have any immunity at all if we use time constraints involving both a lower bound and an upper bound. This seems to be a discouraging result, but it remains yet to be seen whether useful properties can be formulated using just upper bounds or just lower bounds.

In attempting to achieve these results, we considered two approaches, which we depicted as the upper and lower triangle in the introduction. Let us consider that diagram again (figure 5.1).

We had said that these two approaches could be used to prove slowdown and speedup results for logics that did not have an interpretation based semantics. The first approach, (the lower triangle) involved granting an interpretation based semantics to TRE. Once this had been done, it required one to provide soundness and completeness proofs to prove the consistency between the original semantics and the interpretation based semantics. Only then, would one be able to apply the original theory of slowdown and speedup to derive results for TRE operators. This approach was also successfully completed, and we showed in detail what would have to be done in order to apply Colombo et al.'s theory to a logic which uses a different underlying model than that of interpretations. What has been gleaned from this is that this approach can be quite involving; proving soundness can take a lot of effort and the interpretation based semantics have to be chosen with

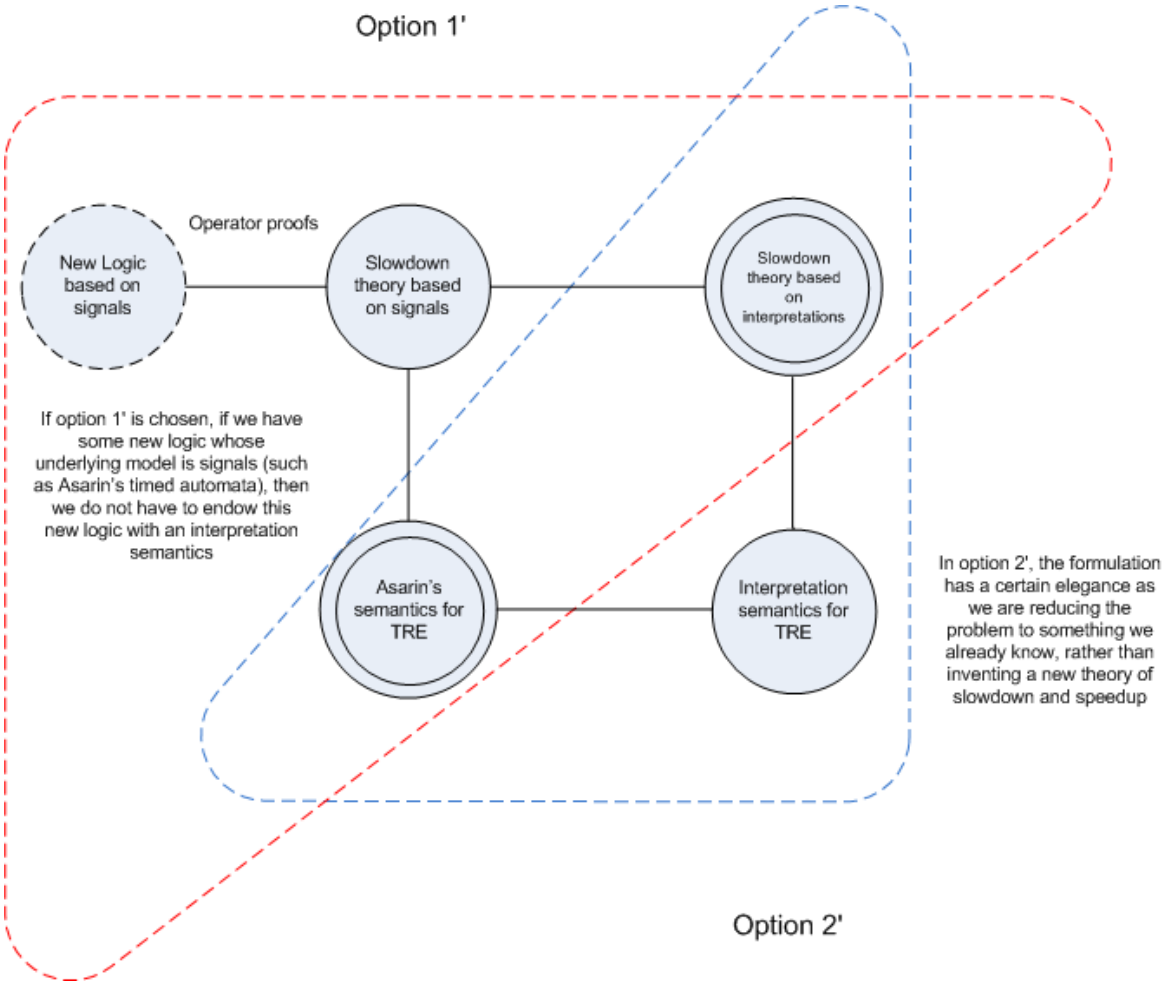


Figure 5.1: Upper and Lower Triangle

care in order for the soundness proof to work. An interesting remark is that giving a new semantics and proving soundness has to be done for each new logic; little can be reused. We shall see what we mean exactly by this statement when we consider the upper triangle.

The upper triangle, provides a different approach to deriving slowdown and speedup results for a logic which does not have its semantics defined in terms of interpretations. The approach consists of two things. The first is to define a new theory of slowdown and speedup which works directly on the new logic's underlying model. The second is to prove that the new theory is sound with respect to either the original theory, or to some other theory which has been proven to be sound to the original. In trying to define a new theory of slowdown and speedup in terms of signals, we noted that the most important notion to define was how time transforms affect the new model, which in our case, was signal lists. Since time transforms operate on absolute scales, while signals operate on a relative scale, we found that this could not be straightforward for some logics, although in our case, it was not particularly difficult. For the soundness proof, we found that what had to be done was to prove that time transforms distribute on the functions which allow us to convert from the model of the new theory (in our case signals) to the model of the

original theory (interpretations) and vice versa. This approach seems to be simpler in terms of the number proofs, especially if one can move directly from the new model to interpretations. However, as we also saw, time transforms might not distribute cleanly through the internals of these functions; in our case we had some trouble distributing them cleanly through the $\overset{\text{I}}{;}$ operator.

This approach has the advantage that it can be reused. By this we mean that once we build a new theory of slowdown and speedup, we can reuse it for all logics whose semantics are given in terms of that model. In our case, this would mean that we could apply this new theory to logics whose semantics are grounded in signals, something which cannot be done with the first approach. However, there was not enough time to apply the second approach to derive again the slowdown and speedup results for TRE. So, it is possible that other problems could be encountered whilst performing this last step.

Some other conclusions that can be drawn is the importance of having a good theory for the underlying models. In our case, we developed a rigorous theory for signal lists, interpretations and critical point lists. Many times, having a formal set of operations indicating how to manipulate the structures serves to uncover many oversights and to suggest possible ways of proving a result. Many important concepts in computer science were used in this work, such as proofs by mathematical induction, invariants, construction and case analysis. We also saw how to define functions to move between models and the need to be rigorous and not to assume that intuitive properties are preserved when moving between models. Additionally, we used the concept of a soundness proof, applied in two different settings; one to prove soundness of semantics and the other to prove the soundness of the definitions of time transforms in the different theories of slowdown and speedup.

We shall conclude this section by providing some background and comments for work related to this one. The motivation behind the development of TRE can be traced to Alur and Dill's paper on Timed Automata as a temporal logic [AD94]. Timed automata keep track of time using clocks and are useful because they are easily executable, and thus are excellent as recognisers of languages when they are used for implementing monitors. Alur and Dill's model did not use signals to specify its semantics, but used a concept of timed words. In contrast with finite signals, timed words represent infinite timed traces. Timed regular expressions were first defined in [ACM97]. The authors assert that they arose from a wish to generalise the Kleene Theorem (which asserts the equivalence of finite state automata and timed regular expressions) to the timed domain. In order to do this, the authors gave a different semantics to timed automata which were based on finite signals instead of infinite timed words. They then gave constructions for converting a TRE to a timed automaton and vice versa, thus proving that these two formalisms are equally powerful.

Timed Regular Expressions are a simple and practical logic to work with in real-time projects. To this end, they are very suitable as an introductory temporal logic. Due to their use in text processing applications, most software engineers are familiar with regular expressions, and timed regular expressions share a strong resemblance to these. The differences which arise are mainly due to the addition of operators for imposing time constraints on the signals being characterised, and this feature also works in an intuitive way. Amongst rival formalisms we find duration calculus and various flavours of timed

automata such as the logic known as DATE, found in the framework LARVA [Col08]. Of these, duration calculus, while more powerful than timed regular expressions, might not be immediately intuitive to work with to the practicing software engineer. Timed automata, on the other hand have the advantage of having a pictorial representation. However, in order to define timed automata, one needs to write down all the states, transitions, guards and other pieces of information. As the automaton's size increases, it becomes increasingly difficult to keep track of its operation, and to ensure that it does characterise the intended property. In contrast to this, we feel that timed regular expressions are compressive, in the sense that a property written using timed regular expressions is much shorter than a script expressing a timed automaton.

The theory of slowdown and speedup was first described in [Col08] and [CPS], and uses duration calculus interpretations as its underlying model. This theory was also developed in the context of work on the runtime verification framework LARVA. The DATE logic (Dynamic Automata with Events and Timers) found in LARVA is based on the concept of *symbolic timed-automata* [CPS08]. The authors claim that these automata are similar to integration automata [BLR], and that they are more powerful than Alur and Dill's timed automata [AD94], due to their ability to pause, reset and resume timers. LARVA also includes a mechanism for compiling properties into monitors and for instrumenting the monitor code into the target system's code using Aspect Oriented Programming [KIL+97] techniques. Finally, this framework also supports writing properties in other formalisms, which can then be translated into DATEs [Col08]. These formalisms include QDDC and counterexample traces (subsets of duration calculus) and Lustre. These different formalisms can provide additional guarantees. For example, Lustre, allows one to predict the effects of monitoring on the memory required [Col08]. Within this context, the theory of slowdown and speedup was developed with the aim of discovering which fragments of duration calculus were guaranteed to be speedup and slowdown truth preserving, so that additional guarantees could be given when expressing properties within the LARVA framework. In fact, LARVA is able to apply the slowdown and speedup results obtained for duration calculus in order to inform the user whether any properties he/she might be defining are safe or not with respect to the effects of slowdown and speedup [CPS]. In the next section, we shall consider potential tasks which can be performed in the future.

5.2 Future Work

This work has opened up several avenues which can be used for future work. One thing that should be done is to take the theory of slowdown and speedup (based on signals) which we have defined in this work, and attempt to use it in order to derive slowdown and speedup results for TRE with signal list semantics. The reasons for this are two fold. Firstly, deriving these results in an alternative way will act as a double check to confirm that these results are correct. More importantly it will act as a case study to determine any potential difficulties in such an approach. A superficial look at this problem will show that one will have to manipulate expressions involving interaction between the *slice* function and time transforms, which is something which is not defined in this work. If this approach proves to be feasible, then it should be tried out on other logics which use the signal model for the definition of their semantics. This would substantiate the claim that

new theories of slowdown and speedup can be effectively reused for other logics sharing the same model.

When we examined the slowdown and speedup results we obtained for timed regular expressions, we also saw the results about the interval restriction operator, and the ability of providing immunity from the problems of slowdown/speedup by using lowerbounds only or upperbounds only respectively. To this end it would be very useful for a thorough study to be performed in order to determine whether useful properties can be written under these constraints. If it turns out that a number of interesting properties are feasible, these might provide patterns for the specification of safe temporal properties (with respect to slowdown and speedup).

Also, due to the practicality of TRE for specifying properties compactly, one could develop a compiler which could translate timed regular expressions into DATEs for the LARVA framework. This would make another logic available to programmers with which to specify properties and thus, monitors, with. As an additional bonus, when using TRE for specifying monitors, the theoretical analysis in this work can be used to check whether a property written by a programmer is safe from the problems of slowdown and speedup or not.

Chapter 6

Appendix

6.1 Appendix A

In this appendix, the reader will find the proofs of theorems stated in the text.

Lemma 1. $\overset{sig}{+\!\!+}$ *is associative. Let* ξ_1, ξ_2 *and* ξ_3 *be signal lists. Then,*

$$(\xi_1 \overset{sig}{+\!\!+} \xi_2) \overset{sig}{+\!\!+} \xi_3 = \xi_1 \overset{sig}{+\!\!+} (\xi_2 \overset{sig}{+\!\!+} \xi_3)$$

Proof. By induction on ξ_3.

Base Case $\xi_3 = \langle\rangle$

$$\begin{aligned} & (\xi_1 \overset{\text{sig}}{+\!\!+} \xi_2) \overset{\text{sig}}{+\!\!+} \langle\rangle \\ = \; & \{ \text{ Definition of } \overset{\text{sig}}{+\!\!+} \} \\ & \xi_1 \overset{\text{sig}}{+\!\!+} \xi_2 \\ = \; & \{ \text{ Definition of } \overset{\text{sig}}{+\!\!+} \} \\ & \xi_1 \overset{\text{sig}}{+\!\!+} (\xi_2 \overset{\text{sig}}{+\!\!+} \langle\rangle) \end{aligned}$$

Inductive Case Assuming

$$(\xi_1 \overset{\text{sig}}{+\!\!+} \xi_2) \overset{\text{sig}}{+\!\!+} \xi_3' = \xi_1 \overset{\text{sig}}{+\!\!+} (\xi_2 \overset{\text{sig}}{+\!\!+} \xi_3')$$

then

$$(\xi_1 \overset{\text{sig}}{+\!\!+} \xi_2) \overset{\text{sig}}{+\!\!+} \xi_3' \overset{\text{sig}}{;} (a, k) = \xi_1 \overset{\text{sig}}{+\!\!+} (\xi_2 \overset{\text{sig}}{+\!\!+} \xi_3' \overset{\text{sig}}{;} (a, k))$$

$$\begin{aligned} & \xi_1 \overset{\text{sig}}{+\!\!+} (\xi_2 \overset{\text{sig}}{+\!\!+} \xi_3' \overset{\text{sig}}{;} (a, k)) \\ = \; & \{ \text{ Definition of } \overset{\text{sig}}{+\!\!+} \} \\ & \xi_1 \overset{\text{sig}}{+\!\!+} (\xi_2 \overset{\text{sig}}{+\!\!+} \xi_3') \overset{\text{sig}}{;} (a, k) \\ = \; & \{ \text{ Inductive Hypothesis } \} \\ & (\xi_1 \overset{\text{sig}}{+\!\!+} \xi_2) \overset{\text{sig}}{+\!\!+} \xi_3' \overset{\text{sig}}{;} (a, k) \end{aligned}$$

□

Lemma 2. *Let ξ be a signal list. Then, the empty signal list $\langle\rangle$ is the zero of the operation $\overset{sig}{+\!\!+}$. The laws below thus hold.*

$$\xi \overset{sig}{+\!\!+} \langle\rangle = \xi$$

and

$$\langle\rangle \overset{sig}{+\!\!+} \xi = \xi$$

Proof. The first direction is true by definition of $\overset{\text{sig}}{+\!\!+}$. We shall thus prove the second direction by induction on ξ.

Base Case: $\xi = \langle\rangle$

$$\begin{array}{ll} & \langle\rangle \overset{\text{sig}}{+\!\!+} \langle\rangle \\ = & \{ \text{ Definition of } \overset{\text{sig}}{+\!\!+} \} \\ & \langle\rangle \end{array}$$

Inductive Case: Assume

$$\langle\rangle \overset{\text{sig}}{+\!\!+} \xi' = \xi'$$

then

$$\langle\rangle \overset{\text{sig}}{+\!\!+} \xi' \overset{\text{sig}}{;} (a, k) = \xi' \overset{\text{sig}}{;} (a, k)$$

$$\begin{array}{ll} & (\langle\rangle \overset{\text{sig}}{+\!\!+} \xi' \overset{\text{sig}}{;} (a, k)) \\ = & \{ \text{ Definition of } \overset{\text{sig}}{+\!\!+} \} \\ & (\langle\rangle \overset{\text{sig}}{+\!\!+} \xi') \overset{\text{sig}}{;} (a, k) \\ = & \{ \text{ Inductive Hypothesis } \} \\ & \xi' \overset{\text{sig}}{;} (a, k) \end{array}$$

□

Lemma 3. *It is always possible to replace an application of $\overset{sig}{;}$ with an application of $\overset{sig}{+\!\!+}$ on a singleton list, and vice versa. This is represented as the following law.*

$$\xi \overset{sig}{;} (\alpha, k) = \xi \overset{sig}{+\!\!+} \langle(\alpha, k)\rangle$$

Proof.

$$
\begin{aligned}
& (\xi \overset{\text{sig}}{+\!\!+} \langle(\alpha, k)\rangle) \\
= & \quad \{ \text{ Definition of } \overset{\text{sig}}{;} \ \} \\
& (\xi \overset{\text{sig}}{+\!\!+} \langle\rangle \overset{\text{sig}}{;} (\alpha, k)) \\
= & \quad \{ \text{ Definition of } \overset{\text{sig}}{+\!\!+} \ \} \\
& (\xi \overset{\text{sig}}{+\!\!+} \langle\rangle) \overset{\text{sig}}{;} (\alpha, k) \\
= & \quad \{ \text{ Definition of } \overset{\text{sig}}{+\!\!+} \ \} \\
& \xi \overset{\text{sig}}{;} (\alpha, k)
\end{aligned}
$$

□

Lemma 4. *The length of a signal list is the sum of the lengths of the segments. If ξ is a signal list, and k_i is the length of the i^{th} segment, then the following holds.*

$$|\xi| = \sum_{k_i \in \xi} k_i$$

Proof. By induction on ξ.

Base Case: Let ξ be the empty signal list. Then there are no tuples in ξ, which means that the summation term is 0. Since the length of an empty signal list is also 0, this concludes the base case.

Inductive Case: Assume that

$$|\xi'| = \sum_{k_i \in \xi} k_i$$

We want to prove that

$$|\xi' \overset{\text{sig}}{;} (\alpha, k)| = \sum_{k_i \in \xi} k_i + k$$

$$
\begin{aligned}
& |\xi' \overset{\text{sig}}{;} (\alpha, k)| \\
= & \quad \{ \text{ Definition of Length of a Signal List } \} \\
& |\xi'| + k \\
= & \quad \{ \text{ Inductive Hypothesis } \} \\
& \textstyle\sum_{k_i \in \xi} k_i + k
\end{aligned}
$$

□

Lemma 5. *The length operator over signal lists distributes over the signal list concatenation operator $\overset{sig}{+\!\!+}$. If ξ_1 and ξ_2 are signal lists, then the following holds. $|\xi_1 \overset{sig}{+\!\!+} \xi_2| = |\xi_1| + |\xi_2|$*

Proof. By induction on ξ_2.

Base Case: $|\xi_1 \overset{\text{sig}}{+\!\!+} \langle\rangle| = |\xi_1| + |\langle\rangle|$

$$
\begin{array}{ll}
 & |\xi_1 \overset{\text{sig}}{+\!\!+} \langle\rangle| \\
= & \{ \text{ Concatenation with } \langle\rangle \text{ } \} \\
 & |\xi_1| \\
= & \{ \text{ x + 0 = x } \} \\
 & |\xi_1| + 0 \\
= & \{ \text{ Length of empty signal list is 0 } \} \\
 & |\xi_1| + |\langle\rangle|
\end{array}
$$

Inductive Case: Assuming

$$|\xi_1 \overset{\text{sig}}{+\!\!+} \xi_2'| = |\xi_1| + |\xi_2'|$$

then,

$$|\xi_1 \overset{\text{sig}}{+\!\!+} \xi_2' \overset{\text{sig}}{;} (\alpha, k)| = |\xi| + |\xi_2' \overset{\text{sig}}{;} (\alpha, k)|$$

$$
\begin{array}{ll}
 & |(\xi_1 \overset{\text{sig}}{+\!\!+} \xi_2' \overset{\text{sig}}{;} (\alpha, k))| \\
= & \{ \text{ Definition of } \overset{\text{sig}}{+\!\!+} \text{ } \} \\
 & |(\xi_1 \overset{\text{sig}}{+\!\!+} \xi_2') \overset{\text{sig}}{;} (\alpha, k)| \\
= & \{ \text{ Definition of } | \text{ } | \} \\
 & |(\xi_1 \overset{\text{sig}}{+\!\!+} \xi_2')| + k \\
= & \{ \text{ Inductive Hypothesis } \} \\
 & |\xi_1| + |\xi_2'| + k \\
= & \{ \text{ Definition of } | \text{ } | \} \\
 & |\xi_1| + |\xi_2' \overset{\text{sig}}{;} (\alpha, k)|
\end{array}
$$

□

Theorem 6. *The constructive definition of I satisfies wellformedness.*

Proof. By induction on the type definition.

Base Case: Consider the empty interpretation $False$. This interpretation is false for all times and values. Let T = 0. T satisfies the predicate since at and after this point, all symbols are false. Since no point can be smaller than 0, the interpretation does not need to satisfy the predicate talking about what happens before T, which proves the base case.

Inductive Case: Suppose that wellformedness holds for an interpretation i, and suppose we add a new wellformed singleton interpretation j to it, of length $|j|$. We want to prove that under these conditions, $i \overset{\text{I}}{;} j$ is also wellformed. How can we prove this? We have to find the point T before which only one symbol is active at a time, and at which and after which no symbol is active. Now, $i \overset{\text{I}}{;} j$ resolves to i for any point

before $|i|$. Being wellformed, i satisfies mutual exclusion for every point before $|i|$, so this will also apply to $i \overset{\text{I}}{;} j$. For all points at and after $|i|$, $i \overset{\text{I}}{;} j$ resolves to the value of j at $t-|i|$. Now, between 0 and $|j|$ (excluded), j also satisfies mutual exclusion, since it is well formed. Therefore $i \overset{\text{I}}{;} j$ satisfies mutual exclusion for all points between $|i|$ and $|i|+|j|$ (excluded). We now claim that T = $|i|+|j|$. We have already determined that all points prior to this satisfy mutual exclusion. However, this point, and all points after it are false at the corresponding points for j. Therefore they will also be false for $i \overset{\text{I}}{;} j$. This means that we have found a T satisfying the wellformedness predicate, indicating that $i \overset{\text{I}}{;} j$ is also wellformed.

□

Lemma 7. *The operator $\overset{I}{+\!+}$ is associative. If i,j and k are interpretations, then the following holds.*

$$(i \overset{I}{+\!+} j) \overset{I}{+\!+} k = i \overset{I}{+\!+} (j \overset{I}{+\!+} k)$$

Proof. By induction on k.

Base Case: $k = False$

$$\begin{array}{ll} & (i \overset{\text{I}}{+\!+} j) \overset{\text{I}}{+\!+} False \\ = & \{ \text{ Definition of } \overset{\text{I}}{+\!+} \} \\ & i \overset{\text{I}}{+\!+} j \\ = & \{ \text{ Definition of } \overset{\text{I}}{+\!+} \} \\ & i \overset{\text{I}}{+\!+} (j \overset{\text{I}}{+\!+} False) \end{array}$$

Inductive Case: Assuming

$$(i \overset{\text{I}}{+\!+} j) \overset{\text{I}}{+\!+} k' = i \overset{\text{I}}{+\!+} (j \overset{\text{I}}{+\!+} k')$$

then

$$(i \overset{\text{I}}{+\!+} j) \overset{\text{I}}{+\!+} k' \overset{\text{I}}{;} l = i \overset{\text{I}}{+\!+} (j \overset{\text{I}}{+\!+} k' \overset{\text{I}}{;} l)$$

$$\begin{array}{ll} & i \overset{\text{I}}{+\!+} (j \overset{\text{I}}{+\!+} k' \overset{\text{I}}{;} l) \\ = & \{ \text{ Definition of } \overset{\text{I}}{+\!+} \} \\ & i \overset{\text{I}}{+\!+} (j \overset{\text{I}}{+\!+} k') \overset{\text{I}}{;} l \\ = & \{ \text{ Inductive Hypothesis } \} \\ & (i \overset{\text{I}}{+\!+} j) \overset{\text{I}}{+\!+} k' \overset{\text{I}}{;} l \end{array}$$

□

Lemma 8. *Appending a singleton interpretation to the empty interpretation will yield back the singleton interpretation. Let j be a singleton interpretation, then the following holds.* $False \overset{I}{;} j = j$

Proof. Consider

$$\begin{aligned} False \overset{\mathrm{I}}{;} j(\alpha, t) &= False(\alpha, t) \text{ for } 0 \leq t < |False| \\ & \quad j(\alpha, t - |False|) \text{ for } |False| \leq t \end{aligned}$$

Since the length of the empty interpretation is 0, then we have

$$False \overset{\mathrm{I}}{;} j(\alpha, t) = j(\alpha, t) \text{ for } 0 \leq t$$

as required.

□

Lemma 9. *The empty interpretation is the zero of the operator* $\overset{I}{+\!\!+}$. *Let i and j be interpretations. Then, the following two laws hold.*

$$i \overset{I}{+\!\!+} False = i$$

and

$$False \overset{I}{+\!\!+} j = j$$

Proof. The first direction is true by definition of $\overset{\mathrm{I}}{+\!\!+}$. We shall prove the second direction by induction on j.

Base Case: $j = False$

$$\begin{aligned} & False \overset{\mathrm{I}}{+\!\!+} False \\ = \; & \{ \text{ Definition of } \overset{\mathrm{I}}{+\!\!+} \} \\ & False \end{aligned}$$

Inductive Case: Assume

$$False \overset{\mathrm{I}}{+\!\!+} j' = j'$$

then

$$False \overset{\mathrm{I}}{+\!\!+} j' \overset{\mathrm{I}}{;} l = j' \overset{\mathrm{I}}{;} l$$

$$\begin{aligned} & (False \overset{\mathrm{I}}{+\!\!+} j' \overset{\mathrm{I}}{;} l) \\ = \; & \{ \text{ Definition of } \overset{\mathrm{I}}{+\!\!+} \} \\ & (False \overset{\mathrm{I}}{+\!\!+} j') \overset{\mathrm{I}}{;} l \\ = \; & \{ \text{ Inductive Hypothesis } \} \\ & j' \overset{\mathrm{I}}{;} l \end{aligned}$$

□

Lemma 10. *An application of* $\overset{I}{;}$ *can always be replaced with an application of* $\overset{I}{+\!\!+}$*. The opposite is also true if the second parameter of* $\overset{I}{+\!\!+}$ *is a singleton interpretation. Let* i *be an interpretation, and* j *a singleton interpretation, then the following holds.*

$$i \overset{I}{;} j = i \overset{I}{+\!\!+} j$$

Proof.

$$\begin{array}{ll} & i \overset{\mathrm{I}}{+\!\!+} j \\ = & \{\ False \overset{\mathrm{I}}{;} j = j\ \} \\ & (i \overset{\mathrm{I}}{+\!\!+} False \overset{\mathrm{I}}{;} j) \\ = & \{\ \text{Definition of } \overset{\mathrm{I}}{+\!\!+}\ \} \\ & (i \overset{\mathrm{I}}{+\!\!+} False) \overset{\mathrm{I}}{;} j \\ = & \{\ \text{Definition of } \overset{\mathrm{I}}{+\!\!+}\ \} \\ & i \overset{\mathrm{I}}{;} j \end{array}$$

□

Lemma 11. *The length over an interpretation distributes over the operator* $\overset{I}{;}$*. Let* i *be an interpretation and* j *be a singleton interpretation, then the following holds.*

$$|i \overset{I}{;} j| = |i| + |j|$$

Proof. The length of i occurs at i's cutoff point T_i. Similarly, the length of j is equal to the position of its cutoff point T_j. What is the cutoff point of $i \overset{\mathrm{I}}{;} j$?

For a point to classify as the cutoff point it must have the characteristic that no symbol is ever active at it or after it, and that for every point before it one symbol must be true. Does the point $T_i + T_j$ satisfy these conditions? Clearly all times before T_i must have one symbol active since is wellformed. What about the times t between T_i(inclusive) and $T_i + T_j$(exclusive)? Each of these points is greater or equal to T_i, so the value of $i \overset{\mathrm{I}}{;} j$ at these points is the value of t at the point minus T_i. Since j must also be wellformed, then at each of these points $i \overset{\mathrm{I}}{;} j$ has one symbol active. What is the value of $i \overset{\mathrm{I}}{;} j$ at $T_i + T_j$? It is the value of j at T_j. Checking the definition of a singleton interpretation at j confirms that at this point and for all points thereafter, all symbols are false, which means that the length of $i \overset{\mathrm{I}}{;} j$ is $T_i + T_j$, or, in other words, $|i \overset{\mathrm{I}}{;} j| = |i| + |j|$

□

Theorem 12. *The constructive definition of CPL satisfies the requirements of Definition 19.*

Proof. By induction on the type definition.

Base Case: Consider the case when the CPL is $\langle(0, \bot)\rangle$. Since the list contains only one item, the first two requirements are vacuously true. The third requirement is also true since the CPL is of the required form.

Inductive Case: Suppose that a CPL $xs : (t, \bot)$ satisfies the requirements. Then, we must prove that if we apply the $\overset{\text{cp}}{;}$ operator, obtaining $xs : (t_1, \bot) \overset{\text{cp}}{;} (0, \alpha)(t_2, \bot)$ the result still satisfies the requirements. If we use the definition of $\overset{\text{cp}}{;}$, we obtain the following

$$xs : (t_1, \alpha) : (t_1 + t_2, \bot)$$

The first requirement is satisfied, since $t_2 > 0$ ($t_2 \in \mathbb{R}^+$) and therefore $t_1 + t_2$ is greater than t_1. Also, if t_1 was greater than the times of all the other tuples before the application of $\overset{\text{cp}}{;}$, this will still be the case after using $\overset{\text{cp}}{;}$. The second requirement is also satisfied since the appending rule requires that symbol of the tuple being appended (α) is different from the last symbol of the list to which we are appending (β). The third requirement is satisfied since after the application of the $\overset{\text{cp}}{;}$ operator, the critical point list still terminates with a $\bot$ symbol.

□

Lemma 13. *The $\overset{cp}{+\!\!+}$ operator is associative. Let xs, ys and zs be critical point lists, then the following holds.*

$$(xs \overset{cp}{+\!\!+} ys) \overset{cp}{+\!\!+} zs = xs \overset{cp}{+\!\!+} (ys \overset{cp}{+\!\!+} zs)$$

Proof. By induction on zs.

Base Case: $zs = \langle(0, \bot)\rangle$

$$\begin{aligned} & xs \overset{\text{cp}}{+\!\!+} (ys \overset{\text{cp}}{+\!\!+} \langle(0, \bot)\rangle) \\ = \; & \{ \text{ Definition } \overset{\text{cp}}{+\!\!+} \} \\ & xs \overset{\text{cp}}{+\!\!+} ys \\ = \; & \{ \text{ Bracketing } \} \\ & (xs \overset{\text{cp}}{+\!\!+} ys) \\ = \; & \{ \langle(0, \bot)\rangle \text{ is the zero of } \overset{\text{cp}}{+\!\!+} \} \\ & (xs \overset{\text{cp}}{+\!\!+} ys) \overset{\text{cp}}{+\!\!+} \langle(0, \bot)\rangle \end{aligned}$$

Inductive Case: Assume

$$xs \overset{\text{cp}}{+\!\!+} ys \overset{\text{cp}}{+\!\!+} zs'$$

Then

$$xs \overset{\text{cp}}{+\!\!+} (ys \overset{\text{cp}}{+\!\!+} zs' \overset{\text{cp}}{;} (0, \alpha)(t, \bot)) =$$
$$(xs \overset{\text{cp}}{+\!\!+} ys) \overset{\text{cp}}{+\!\!+} zs' \overset{\text{cp}}{;} (0, \alpha)(t, \bot)$$

$$
\begin{array}{ll}
 & xs \overset{\text{cp}}{+\!\!+} (ys \overset{\text{cp}}{+\!\!+} zs' \overset{\text{cp}}{;} (0,\alpha)(t,\bot)) \\
= & \{ \text{ Definition of } \overset{\text{cp}}{+\!\!+} \ \} \\
 & xs \overset{\text{cp}}{+\!\!+} (ys \overset{\text{cp}}{+\!\!+} zs') \overset{\text{cp}}{;} (0,\alpha)(t,\bot) \\
= & \{ \text{ Inductive Hypothesis } \} \\
 & (xs \overset{\text{cp}}{+\!\!+} ys) \overset{\text{cp}}{+\!\!+} zs' \overset{\text{cp}}{;} (0,\alpha)(t,\bot)
\end{array}
$$

□

Lemma 14. *The empty critical point list $\langle(0,\bot)\rangle$ is the zero of the operator $\overset{cp}{+\!\!+}$. Let xs be a CPL. Then the following two laws hold.*

$$xs \overset{cp}{+\!\!+} \langle(0,\bot)\rangle = xs$$

and

$$\langle(0,\bot)\rangle \overset{cp}{+\!\!+} xs = xs$$

Proof. The first statement follows directly from the definition of $\overset{\text{cp}}{+\!\!+}$. We prove the second statement by induction.

Base Case $xs = \langle(0,\bot)\rangle$

$$
\begin{array}{ll}
 & \langle(0,\bot)\rangle \overset{\text{cp}}{+\!\!+} \langle(0,\bot)\rangle \\
= & \{ \text{ Definition of } \overset{\text{cp}}{+\!\!+} \ \} \\
 & \langle(0,\bot)\rangle
\end{array}
$$

Inductive Case Assuming

$$\langle(0,\bot)\rangle \overset{\text{cp}}{+\!\!+} xs' = xs'$$

then,

$$\langle(0,\bot)\rangle \overset{\text{cp}}{+\!\!+} xs' \overset{\text{cp}}{;} (0,\alpha)(t,\bot) = xs' \overset{\text{cp}}{;} (0,\alpha)(t,\bot)$$

$$
\begin{array}{ll}
 & (\langle(0,\bot)\rangle \overset{\text{cp}}{+\!\!+} xs' \overset{\text{cp}}{;} (0,\alpha)(t,\bot)) \\
= & \{ \text{ Definition of } \overset{\text{cp}}{+\!\!+} \ \} \\
 & (\langle(0,\bot)\rangle \overset{\text{cp}}{+\!\!+} xs') \overset{\text{cp}}{;} (0,\alpha)(t,\bot) \\
= & \{ \text{ Inductive Hypothesis } \} \\
 & xs' \overset{\text{cp}}{;} (0,\alpha)(t,\bot)
\end{array}
$$

□

Lemma 15. *The $\overset{cp}{;}$ operator can always be replaced with a $\overset{cp}{+\!\!+}$ operator over a critical point list with two elements. Let xs be a critical point list. Then, the following holds.*

$$xs \overset{cp}{;} (0,\alpha)(t,\bot) = xs \overset{cp}{+\!\!+} \langle(0,\alpha),(t,\bot)\rangle$$

Proof.

$$\begin{array}{ll} & xs \overset{\text{cp}}{+\!\!+} \langle (0,\alpha),(t,\bot)\rangle \\ = & \{ \text{ Definition of } \overset{\text{cp}}{;} \ \} \\ & xs \overset{\text{cp}}{+\!\!+} (\langle (0,\bot)\rangle \overset{\text{cp}}{;} (0,\alpha)(t,\bot)) \\ = & \{ \text{ Definition of } \overset{\text{cp}}{+\!\!+} \ \} \\ & (xs \overset{\text{cp}}{+\!\!+} \langle (0,\bot)\rangle) \overset{\text{cp}}{;} (0,\alpha)(t,\bot) \\ = & \{ \text{ Definition of } \overset{\text{cp}}{+\!\!+} \ \} \\ & xs \overset{\text{cp}}{;} (0,\alpha)(t,\bot) \end{array}$$

□

Lemma 16. *The length operator over a critical point list, distributes over* $\overset{cp}{;}$*. Let* xs *be a critical point list. Then the following holds.*

$$|xs \overset{cp}{;} (0,\alpha)(k,\bot)| = |xs| + k$$

Proof.

$$\begin{array}{ll} & |xs : (t,\bot) \overset{\text{cp}}{;} (0,\alpha)(k,\bot)| \\ = & \{ \text{ Definition of } \overset{\text{cp}}{;} \ \} \\ & |xs : (t,\alpha) : (t+k,\bot)| \\ = & \{ \text{ Definition of } |\ | \text{ on a CPL } \} \\ & t + k \\ = & \{ \text{ Definition of } |\ | \text{ on a CPL } \} \\ & |xs : (t,\alpha)| + k \end{array}$$

□

Lemma 17. *The length operator over critical point lists distributes over* $\overset{cp}{+\!\!+}$*. Let* xs *and* ys *be critical point lists. Then the following holds*

$$|xs \overset{cp}{+\!\!+} ys| = |xs| + |ys|$$

Proof. By induction on ys.

Base Case: $ys = \langle (0,\bot)\rangle$

$$\begin{array}{ll} & |xs \overset{\text{cp}}{+\!\!+} \langle (0,\bot)\rangle| \\ = & \{ \text{ Definition of } \overset{\text{cp}}{+\!\!+} \ \} \\ & |xs| \\ = & \{ \text{ Length of } \langle (0,\bot)\rangle \text{ is 0 } \} \\ & |xs| + |\langle (0,\bot)\rangle| \end{array}$$

Inductive Case: Assume

$$|xs \overset{\text{cp}}{+\!\!+} ys| = |xs| + |ys|$$

then

$$|xs \overset{\text{cp}}{+\!\!+} ys \overset{\text{cp}}{;} (0,\alpha)(t,\bot)| = |xs| + |ys \overset{\text{cp}}{;} (0,\alpha)(t,\bot)|$$

$$
\begin{array}{ll}
 & |(xs \overset{\text{cp}}{+\!\!+} ys \overset{\text{cp}}{;} (0,\alpha)(k,\bot))| \\
= & \{ \text{ Definition of } \overset{\text{cp}}{+\!\!+} \ \} \\
 & |(xs \overset{\text{cp}}{+\!\!+} ys) \overset{\text{cp}}{;} (0,\alpha)(k,\bot)| \\
= & \{ \text{ Lemma 16 } \} \\
 & |(xs \overset{\text{cp}}{+\!\!+} ys)| + k \\
= & \{ \text{ Inductive Hypothesis } \} \\
 & |xs| + |ys| + k \\
= & \{ \text{ Lemma 16 } \} \\
 & |xs| + |ys \overset{\text{cp}}{;} (0,\alpha)(k,\bot)|
\end{array}
$$

□

Lemma 18. *A critical point list can be split into two parts at any tuple by using the operator* $\overset{cp}{+\!\!+}$*. Let*

$$xs = \langle (k_1, a_1), \ldots, (k_n, \bot) \rangle$$

For any tuple (k_i, a_i) *there exists a critical point list* zs *such that*

$$xs = ys : (k_i, \bot) \overset{cp}{+\!\!+} zs$$

where ys *consists of all the tuples from* (k_1, α_1) *to* (k_{i-1}, α_{i-1})*.*

Proof. By reverse induction on ys.

Base Case: $i = n$

In this case, let $zs = \langle (0, \bot) \rangle$. All we have to do is check whether the equality is satisfied with this value of zs.

$$
\begin{array}{ll}
 & ys \overset{\text{cp}}{+\!\!+} zs \\
= & \{ \text{ Replacing } ys \text{ and } zs \text{ by their values } \} \\
 & \langle (k_1, a_1), \ldots, (k_n, a_n) \rangle \overset{\text{cp}}{+\!\!+} \langle (0, \bot) \rangle \\
= & \{ \text{ Definition of } \overset{\text{cp}}{+\!\!+} \ \} \\
 & \langle (k_1, a_1), \ldots, (k_n, a_n) \rangle \\
= & \{ \text{ Value of xs } \} \\
 & xs
\end{array}
$$

Inductive Case: Assume statement holds for $i = j$

$$\exists zs : CPL.\ xs = ys : (k_{j-1}, a_j) : (k_j, \bot) \overset{\text{cp}}{+\!\!+} zs$$

then it holds for $i = j - 1$

$$\exists zs' : CPL.\ xs = ys : (k_{j-1}, \bot) \overset{\text{cp}}{+\!\!+} zs'$$

$$
\begin{array}{ll}
 & \exists zs : CPL.\ xs = ys : (k_{j-1}, a_j) : (k_j, \bot) \overset{\text{cp}}{+\!\!+} zs \\
\implies & \{ \text{ Replacing } ys : (k_{j-1}, a_j) : (k_j, \bot) \text{ with its value } \} \\
 & \exists zs : CPL.\ xs = \langle (k_1, a_1), \ldots, (k_j, \bot) \rangle \overset{\text{cp}}{+\!\!+} zs \\
\implies & \{ \text{ Definition of } \overset{\text{cp}}{;} \} \\
 & \exists zs : CPL.\ xs = \langle (k_1, a_1), \ldots, (k_{j-1}, \bot) \rangle \overset{\text{cp}}{;} (0, a_{j-1})(k_j, \bot) \overset{\text{cp}}{+\!\!+} zs \\
\implies & \{ \text{ Replacing } \overset{\text{cp}}{;} \text{ with } \overset{\text{cp}}{+\!\!+} \} \\
 & \exists zs : CPL.\ xs = \langle (k_1, a_1), \ldots, (k_{j-1}, \bot) \rangle \overset{\text{cp}}{+\!\!+} \langle (0, a_{j-1}), (k_j, \bot) \rangle \overset{\text{cp}}{+\!\!+} zs \\
\implies & \{ \ zs' = \langle (0, a_{j-1})(k_j, \bot) \rangle \overset{\text{cp}}{+\!\!+} zs \ \} \\
 & \exists zs' : CPL.\ xs = \langle (k_1, a_1), \ldots, (k_{j-1}, \bot) \rangle \overset{\text{cp}}{+\!\!+} zs' \\
\implies & \{ \text{ Replacing list value by } ys : (k_{j-1}, \bot) \ \} \\
 & \exists zs' : CPL.\ xs = ys : (k_{j-1}, \bot) \overset{\text{cp}}{+\!\!+} zs'
\end{array}
$$

□

Proposition 20. *x is a degenerate critical point of an interpretation i, if and only if it is associated with the symbol $\bot$, and vice versa.*

$$degenerate(x, i) \Leftrightarrow (x, \bot) \in \overline{C(i)}$$

Proof. Let us start with the forward direction. x is a degenerate critical point. This means that $x \in C(i)$. Consider the definition of $\overline{C(i)}$. If it is degenerate, then at all points after and including x, no symbol is true. This satisfies the second constraint in the second set comprehension of $\overline{C(i)}$. Therefore, $(x, \bot)$ is in $\overline{C(i)}$.

We now consider the reverse direction. Suppose $(x, \bot) \in \overline{C(i)}$. The only way that a point can have the symbol $\bot$ associated to it is if it satisfies the second set comprehension constraint in the definition of $\overline{C(i)}$

$$\{(x, \bot) \mid x \in C(i) \wedge \forall t : \mathbb{T}, \alpha : \Sigma.\ t \geq x \Rightarrow \neg i(\alpha, t)\}$$

Therefore x is a critical point of i, and for all points at and after it, no symbol is ever true. But this is exactly the definition of a degenerate critical point, which proves the theorem.

□

Proposition 21. *Every well formed signal interpretation i has one and only one degenerate critical point. Moreover, this point is the same point as the interpretation's cutoff point T. Consequently, the position of the degenerate critical point is equal to the length of the interpretation.*

Proof. First we prove that every well formed interpretation has one and only one degenerate critical point. Since the interpretation is well-formed, then there is a point T at which, and after which no symbol is true, which by definition is a degenerate critical point. We now prove that there is only one degenerate critical point. Before T, there cannot be any degenerate critical point, since due to the definition of wellformedness, there is always

some symbol active. Also, after T, no symbol ever changes, since if it changed, some symbol would become true, which by definition of wellformedness is not possible. Since no symbol ever changes, there can be no other degenerate critical points after T. Now, since the one degenerate critical point is attained at the cutoff value T, it must occour, by definition of the length of an interpretation, at the value $|i|$.

□

Proposition 23. *Let i be a well formed signal interpretation, and C(i) be the set of its critical points. If x is i's degenerate critical point, then x is its largest critical point. This can be expressed as*

$$degenerate(x, i) \Rightarrow x = \max\{C(i)\}$$

Proof. A degenerate critical point x ensures that no other symbol is true thereafter. Assume, for contradiction, that some other point c in $C(i)$ has a value larger than x. This point has to be non-degenerate, since by Proposition 21 we already know that an interpretation can have only one degenerate critical point. Then, by definition of a non-degenerate critical point, it changes the value of the interpretation i from one value α to some other value β. This means that there exists a β for which i is true at and after x. But this is a contradiction since i cannot be true for any value after x. Therefore, x is the maximum of $C(i)$.

□

Proposition 24. *Let i be an interpretation. Then $\overline{C(i)}$ can be ordered, and the resulting sequence is a Critical Point List.*

Proof. We first show that $\overline{C(i)}$ can be ordered. Consider the set $\overline{C(i)}$. The first element of each tuple of this set is an element of $\mathbb{R}_0^+$. It is thus possible to order the tuples on their first member under the relation $\geq$ into the sequence:

$$\langle (c_0, \alpha_0), (c_1, \alpha_1), \ldots, (c_n, \alpha_n) \rangle$$

Now we show that the result is a critical point list. First of all, the list has the required type $seq(\mathbb{R}_0^+ \times \Sigma)$. In order to be a critical point list, it must satisfy the three requirements set in definition of a critical point list (Definition 19). The first requirement is trivially satisfied since there cannot be 2 critical points with the same value (due to the fact that C(i) is a set) and that the points are in ascending order (due to being ordered under $\geq$).

The second constraint is satisfied because a critical point occurs when an interpretation changes value. Thus two successive critical points must have different symbols by definition, otherwise one of them would not be a critical point.

Consider the third constraint. The set of critical points always contains at least the element 0 by definition of C(). Suppose it contains just one element. By Proposition 21, we know that every interpretation has one and only one degenerate critical point. So in this case, the point at 0 must be degenerate. Then by 20, we know that a degenerate critical point is always associated to the symbol bottom. So in this case, the ordered sequence contains just $(0, \perp)$ which is of the allowed form.

Now suppose that $C(i)$ contains more than one element. If it contains more than one element, by Proposition 23, we know its largest critical point m (and thus its last)

is degenerate. by Proposition 20, we know that a degenerate critical point is always associated to the symbol bottom. Therefore in this case, the last element of the list is always of the form $(m, \perp)$, as required.

□

Proposition 26. *If a signal is empty, it only has one critical point at t = 0, and this is degenerate. Hence the length of the empty interpretation is 0.*

Proof. The empty signal interpretation, $False$ obtains the value of $false$ for all elements of Σ immediately and persists in this fashion for all time points. This means that there is no change in value after t = 0. Thus, by the definition of C(i), the only point in C(i) is the one at t = 0. Moreover, since after t = 0, all symbols are false, t = 0 is also degenerate, which means that the length of the empty signal interpretation is 0.

□

Proposition 28. *If an interpretation is not empty, then it has at least two critical points.*

Proof. By the definition of critical points, every interpretation has a critical point at t = 0. We now need to prove that there is another critical point. First we note that the interpretation is not empty. This means that it does not attain the value of false immediately. This implies that there is a period for which some symbol is true. Now, since every interpretation is well formed, it eventually attains the state of silence. However, due to the definition of wellformedness, this cannot be attained before every symbol is false. By proposition 21, we know that there is a degenerate critical point at the value T, the value at which silence it attained. However, T cannot be equal to zero, since there is a period during which some symbol is true, and T can only occour after this period has ended. Therefore, the interpretation has at least two critical points.

□

Proposition 29. *There is no change in the value of an interpretation between two consecutive critical points. Let i be an interpretation, and $i2c(i)$ be its critical point list. Then, for any two consecutive critical points c_i and c_{i+1} in $i2c(i)$, if $i(\alpha, t)$ is true, there is no point t in $[c_i, c_{i+1})$ at which $i(\alpha, t) = false$. Similarly, If $i(\alpha, t)$ is false, there is no point t in $[c_i, c_{i+1})$ at which $i(\alpha, t) = true$.*

Proof. By contradiction.

All the critical points of i are in the set $C(i)$. Suppose that there is a point $x \notin C(i)$ between some c_i and c_{i+1} such that the theorem is not true. This means that $i(\alpha, x) = false$, but some other $i(\beta, x) = true$, since by definition of wellformedness, before silence occurs, there must be at least one symbol which is true. Moreover, $\lim_{t \to x^-} i = \alpha \wedge i(\beta, x) = true$, where $\alpha \neq \beta$. Thus, by the definition of critical points, x would qualify as a critical point. But $x \notin C(i)$, which is a contradiction.

□

Proposition 30. *Let i be a singleton interpretation. Then, i has only two critical points.*

Proof. By proposition 28, we know that a singleton interpretation i has at least two critical points, at 0, and at $|i|$. By Proposition 21, we also know that any point at $|i|$ has to be degenerate, and by Proposition 23, that this critical point is the largest amongst all critical points. We now need to show that there cannot be more than two critical points. Given the constraints above, if there are more than two critical points, then these points must be between 0 and $|i|$. Let us assume that there is a critical point x between the two which have already been identified. Since x is a critical point, it requires a change from some previous value of the alphabet to occour at x. Since i is a singleton segment, there is no change in the value of the signal between 0 and $|i|$. This, thus, contradicts the fact that there is a critical point x between the previous two critical points.

□

Lemma 32. *The function $i2c$ has a distributive-like property over the $\overset{I}{;}$ operator. Let i be an interpretation, and j be a singleton interpretation with symbol α and length k. Then,*

$$i2c(i) \overset{cp}{;} (0, \alpha)(k, \bot) = i2c(i \overset{I}{;} j)$$

Proof. Consider $i2c(i) \overset{\text{cp}}{;} (0, \alpha)(|j|, \bot)$. Due to Corollary 25, and the fact $i2c$ yields a critical point list, we can write it as

$$xs : (|i|, \bot) \overset{\text{cp}}{;} (0, \alpha)(|j|, \bot)$$

If we use the Definition of $\overset{\text{cp}}{;}$, we obtain the following critical point list

$$xs : (|i|, \alpha) : (|i| + |j|, \bot)$$

Now, let us consider $i2c(i \overset{\text{I}}{;} j)$. By the definition of the $\overset{\text{I}}{;}$ operator, we know that this is equal to

$$i \overset{\text{I}}{;} j(\alpha, t) = \begin{cases} i(\alpha, t) \text{ for } 0 \leq t < |i| \\ j(\alpha, t - |i|) \text{ for } t \geq |i| \end{cases}$$

Remember that $i2c()$ is based on the $C()$ operator. Since $i \overset{\text{I}}{;} j$ agrees with i for the interval $[0, |i|)$, $i2c(i \overset{\text{I}}{;} j)$ will have all the critical points of i, except for the critical point $(|i|, \bot)$ since $i \overset{\text{I}}{;} j$ does not include this point from i. Also, note that $i \overset{\text{I}}{;} j$ agrees with j from $|i|$ onwards. However also note that this is done by subtracting $|i|$ from all time points after $|i|$. Thus, for any time point t after $|i|$, $i \overset{\text{I}}{;} j$ agrees with the value of j at $t - |i|$. If there is a critical point in j, it is as if there will be a critical point at $t + |j|$. Since j is singleton, by proposition 31, we know that it must have only two critical points at 0 and $|j|$. Thus, $i2c(i \overset{\text{I}}{;} j)$ will also include the critical points at $(0 + |i|, \alpha)$ and $(|j| + |i|, \bot)$. But this is equivalent to our expression for $i2c(i) \overset{\text{cp}}{;} (0, \alpha)(t, \bot)$, so the statement is proved.

□

Lemma 33. *The length of an interpretation is equal to the length of the CPL obtained when i2c is invoked on that interpretation. Let i be an interpretation, then the following holds.*

$$|i2c(i)| = |i|$$

Proof. i is either the empty interpretation or it is not. If it is the empty interpretation, $|i| = 0$, and $i2c(i) = \langle(0, \bot)\rangle$. By definition of the length of an empty critical point list, this also has length 0. Now, consider the case when i is not empty. Then by corollary 25, $i2c(i) = xs : (|i|, \bot)$ By definition of the length operator on critical point lists, $|i2c(i)| = |i|$, as required.

□

Proposition 34. *The function c2i can be expressed in terms of alpha-fold.*

$$c2i(xs) = fold_{CPL} \overset{I}{+\!\!+} xs \;\; False \;\; f$$

where

$$f :: (0 \times \Sigma) \to (\mathbb{R} \times \bot) \to I$$

$$f(0, \alpha)(t_1, \bot) = \begin{cases} i(\alpha, t) = true \; for \; 0 \leq t < t_1 \\ i(\alpha, t) = false \; for \; t \geq t_1 \\ i(\beta, t) = false \; for \; t \geq 0 \end{cases}$$

Proof. By induction on xs.

Base Case: $xs = \langle(0, \bot)\rangle$

For c2i we have

$$\begin{array}{ll} & c2i(\langle(0, \bot)\rangle) \\ = & \{ \text{ Definition of c2i } \} \\ & False \end{array}$$

Whilst for the fold version we have

$$\begin{array}{ll} & fold_{CPL} \overset{\text{I}}{+\!\!+} \langle(0, \bot)\rangle \; False \; f \\ = & \{ \text{ Definition of } fold_\alpha \; \} \\ & False \end{array}$$

Inductive Case: Assume

$$c2i(xs' : (t_1, \bot)) = fold_{CPL} \overset{\text{I}}{+\!\!+} (xs' : (t_1, \bot)) \; False \; f$$

then

$$c2i(xs' : (t_1, \bot) \overset{\text{cp}}{;} (0, \alpha)(t_2, \bot))$$

$$= fold_{CPL} \overset{\text{I}}{+\!\!+} (xs' : (t_1, \bot)) \overset{\text{cp}}{;} (0, \alpha)(t_2, \bot) \; False \; f$$

$$
\begin{aligned}
& c2i(xs' : (t_1, \bot) \overset{\text{cp}}{;} (0, \alpha)(t_2, \bot)) \\
= \quad & \{ \text{ Definition of } \overset{\text{cp}}{;} \ \} \\
& c2i(xs' : (t_1, \alpha) : (t_1 + t_2, \bot)) \\
= \quad & \{ \text{ Definition of c2i } \} \\
& c2i(xs' : (t_1, \bot)) \overset{\text{I}}{;} \begin{cases} i(\alpha, t) = \text{true for } 0 \leq t < t_2 \\ i(\alpha, t) = \text{false for } t \geq t_2 \\ i(\beta, t) = \text{false for } t \geq 0 \end{cases} \\
= \quad & \{ \text{ Replacing } \overset{\text{I}}{;} \text{ with } \overset{\text{I}}{+\!\!+} \ \} \\
& c2i(xs' : (t_1, \bot)) \overset{\text{I}}{+\!\!+} \begin{cases} i(\alpha, t) = \text{true for } 0 \leq t < t_2 \\ i(\alpha, t) = \text{false for } t \geq t_2 \\ i(\beta, t) = \text{false for } t \geq 0 \end{cases} \\
= \quad & \{ \text{ Definition of f } \} \\
& c2i(xs' : (t_1, \bot)) \overset{\text{I}}{+\!\!+} \ f \ (0, \alpha)(t_2, \bot) \\
= \quad & \{ \text{ Inductive Hypothesis } \} \\
& (fold_{CPL} \overset{\text{I}}{+\!\!+} \ (xs' : (t_1, \bot)) \ \ False \ f) \overset{\text{I}}{+\!\!+} \ f \ (0, \alpha)(t_2, \bot) \\
= \quad & \{ \text{ Definition of } fold_\alpha \ \} \\
& fold_{CPL} \overset{\text{I}}{+\!\!+} \ (xs' : (t_1, \bot) \overset{\text{cp}}{;} (0, \alpha)(t_2, \bot)) \ \ False \ f
\end{aligned}
$$

□

Lemma 35. *The length of a critical point list is equal to the length of the interpretation returned by the invocation of c2i on that critical point list. This can be expressed as*

$$|xs| = |c2i(xs)|$$

Proof. By induction on xs.

Base Case: $xs = \langle (0, \bot) \rangle$

$$
\begin{aligned}
& |c2i(\langle (0, \bot) \rangle)| \\
= \quad & \{ \text{ Definition of } c2i \ \} \\
& |False| \\
= \quad & \{ \text{ Length of Empty Interpretation } \} \\
& 0 \\
= \quad & \{ \text{ Length of Empty Critical Point List } \} \\
& |\langle (0, \bot) \rangle|
\end{aligned}
$$

Inductive Case: Assume that

$$|c2i(xs' : (t_1, \bot))| = |xs' : (t_1, \bot)|$$

It is required to prove that

$$|c2i(xs' : (t_1, \bot) \overset{\text{cp}}{;} (0, \alpha)(t_2, \bot))| = |xs' : (t_1, \bot) \overset{\text{cp}}{;} (0, \alpha)(t_2, \bot)|$$

$$
\begin{array}{ll}
 & |c2i(xs' : (t_1, \bot) \overset{\text{cp}}{;} (0, \alpha)(t_2, \bot))| \\
= & \{ \text{ Definition of } \overset{\text{cp}}{;} \ \} \\
 & |c2i(xs' : (t_1, \alpha) : (t_1 + t_2, \bot))| \\
= & \{ \text{ Definition of c2i } \} \\
 & |c2i(xs' : (t_1, \bot)) \overset{\text{I}}{;} \begin{cases} i(\alpha, t) = \text{true for } 0 \leq t < t_2 \\ i(\alpha, t) = \text{false for } t \geq t_2 \\ i(\beta, t) = \text{false for } t \geq 0 \end{cases} | \\
= & \{ \text{ Length distributes over } \overset{\text{I}}{;} \ \} \\
 & |c2i(xs' : (t_1, \bot))| + | \begin{cases} i(\alpha, t) = \text{true for } 0 \leq t < t_2 \\ i(\alpha, t) = \text{false for } t \geq t_2 \\ i(\beta, t) = \text{false for } t \geq 0 \end{cases} | \\
= & \{ \text{ Length of an interpretation } (\mathrm{T} = t_2) \ \} \\
 & |c2i(xs' : (t_1, \bot))| + t_2 \\
= & \{ \text{ Inductive Hypothesis } \} \\
 & |xs' : (t_1, \bot)| + t_2 \\
= & \{ \text{ Length of a CPL } \} \\
 & t_1 + t_2 \\
= & \{ \text{ Length of a CPL } \} \\
 & |xs : (t_1, \alpha) : (t_1 + t_2, \bot)| \\
= & \{ \text{ Definition of } \overset{\text{cp}}{;} \ \} \\
 & |xs : (t_1, \bot) \overset{\text{cp}}{;} (0, \alpha)(t_2, \bot)|
\end{array}
$$

□

Proposition 36. *Let xs be a finite critical point list. Then, $c2i(xs)$ terminates.*

Proof. By induction on xs.

Base Case: $xs = \langle (0, \bot) \rangle$

$c2i(\langle (0, \bot) \rangle)$ does not make any recursive call, so it terminates.

Inductive Case: Assuming that

$$c2i(xs' : (t_1, \bot)) \text{ terminates}$$

then

$$c2i(xs' : (t_1, \bot) \overset{\text{cp}}{;} (0, \alpha)(t_2, \bot)) \text{ terminates}$$

$$
\begin{array}{ll}
 & c2i(xs' : (t_1, \bot) \overset{\text{cp}}{;} (0, \alpha)(t_2, \bot)) \\
= & \{ \text{ Definition of } \overset{\text{cp}}{;} \ \} \\
 & c2i(xs' : (t_1, \alpha) : (t_1 + t_2, \bot)) \\
= & \{ \text{ Definition of c2i } \} \\
 & c2i(xs' : (t_1, \bot)) \overset{\text{I}}{;} \begin{cases} i(\alpha, t) = \text{true for } 0 \leq t < t_2 \\ i(\alpha, t) = \text{false for } t \geq t_2 \\ i(\beta, t) = \text{false for } t \geq 0 \end{cases}
\end{array}
$$

This means that the call has unwound into the recursive call $c2i(xs' : (t_1, \bot))$, which by the inductive hypothesis terminates. The operator $\overset{\text{I}}{;}$ does not invoke any recursive calls which means it also terminates. Thus the expression as a whole terminates.

□

Theorem 37. *Composing i2c with c2i yields the identity function. Let xs be a critical point list, then the following law holds.*

$$i2c(c2i(xs)) = xs$$

Proof. By induction on xs.

Base Case: Required to Prove: $i2c(c2i(\langle(0, \bot)\rangle)) = \langle(0, \bot)\rangle$.

$$\begin{array}{ll} & i2c(c2i(\langle(0, \bot)\rangle)) \\ = & \{ \text{ Definition of } c2i() \ \} \\ & i2c(False) \\ = & \{ \text{ Corollary 27: } i2c(False) = \langle(0, \bot)\rangle \ \} \\ & \langle(0, \bot)\rangle \end{array}$$

Inductive Case: Assume

$$i2c(c2i(xs' : (t, \bot))) = xs' : (t, \bot)$$

then

$$i2c(c2i(xs' : (t, \bot) \overset{\text{cp}}{;} (0, \alpha)(|j|, \bot))) = xs' : (t, \bot) \overset{\text{cp}}{;} (0, \alpha)(|j|, \bot)$$

$$\begin{array}{ll} & i2c(c2i(xs' : (t, \bot) \overset{\text{cp}}{;} (0, \alpha)(|j|, \bot))) \\ = & \{ \text{ Definition of } \overset{\text{cp}}{;} \ \} \\ & i2c(c2i(xs' : (t, \alpha) : (t + |j|, \bot))) \\ = & \{ \text{ Definition of c2i } \} \\ & i2c(c2i(xs' : (t, \bot)) \overset{\text{I}}{;} \begin{cases} j(\alpha, t) = \text{true for } 0 \leq t < |j| \\ j(\alpha, t) = \text{false for } t \geq |j| \\ j(\beta, t) = \text{false for } t \geq 0 \end{cases}) \\ = & \{ \text{ Lemma 32: i2c distributes over } \overset{\text{I}}{;} \ \} \\ & i2c(c2i(xs' : (t, \bot))) \overset{\text{cp}}{;} (0, \alpha)(|j|, \bot) \\ = & \{ \text{ Inductive Hypothesis } \} \\ & xs' : (t, \bot) \overset{\text{cp}}{;} (0, \alpha)(|j|, \bot) \end{array}$$

□

Theorem 38. *Composing c2i with i2c yields the identity function. Let i be an interpretation, then the following law holds*

$$c2i(i2c(i)) = i$$

Proof. By induction on i.

Base Case: $c2i(i2c(False)) = False$

$$\begin{array}{ll} & c2i(i2c(False)) \\ = & \{ \text{ Corollary 27: } i2c(False) = \langle(0, \bot)\rangle \ \} \\ & c2i(\langle(0, \bot)\rangle) \\ = & \{ \text{ Definition of c2i } \} \\ & False \end{array}$$

Inductive Case: Assume

$$c2i(i2c(i')) = i'$$

then

$$c2i(i2c(i' \overset{\text{I}}{;} j)) = i' \overset{\text{I}}{;} j$$

where the singleton interpretation j is true for symbol α between 0 and $|j|$, and false otherwise.

$$\begin{array}{ll} & c2i(i2c(i' \overset{\text{I}}{;} j)) \\ = & \{ \text{ Lemma 32: i2c distributes over } \overset{\text{I}}{;} \ \} \\ & c2i(i2c(i') \overset{\text{cp}}{;} (0, \alpha)(|j|, \bot)) \\ = & \{ \text{ Corollary 25: } i2c(i) = xs : (t, \bot) \ \} \\ & c2i(xs : (|i'|, \bot) \overset{\text{cp}}{;} (0, \alpha)(|j|, \bot)) \\ = & \{ \text{ Definition } \overset{\text{cp}}{;} \ \} \\ & c2i(xs : (|i'|, \alpha) : (|i'| + |j|, \bot)) \\ = & \{ \text{ Definition of c2i } \} \\ & c2i(xs : (|i'|, \bot)) \overset{\text{I}}{;} \begin{cases} j(\alpha, t) = \text{true for } 0 \leq t < |j| \\ j(\alpha, t) = \text{false for } t \geq |j| \\ j(\beta, t) = \text{false for } t \geq 0 \end{cases} \\ = & \{ \text{ Inductive Hypothesis } \} \\ & i' \overset{\text{I}}{;} \begin{cases} j(\alpha, t) = \text{true for } 0 \leq t < |j| \\ j(\alpha, t) = \text{false for } t \geq |j| \\ j(\beta, t) = \text{false for } t \geq 0 \end{cases} \\ = & \{ \text{ Definition of j, from the statement of the Inductive Case } \} \\ & i \overset{\text{I}}{;} j \end{array}$$

□

Proposition 39. *The function c2s can be expressed in terms of alpha-fold.*

$$c2s(xs) = fold_{CPL} \overset{sig}{+\!\!+} \ xs \ \langle\rangle \ f$$

where

$$f :: (0 \times \Sigma) \to (\mathbb{R} \times \bot) \to SIG$$
$$f(0, \alpha)(t, \bot) = \langle(\alpha, t)\rangle$$

Proof. By induction on xs.

Base Case: $xs = \langle(0, \bot)\rangle$

For c2s we have

$$\begin{array}{ll} & c2s(\langle(0, \bot)\rangle) \\ = & \{ \text{ Definition of c2s } \} \\ & \langle\rangle \end{array}$$

Whilst for the fold version we have

$$\begin{array}{ll} & fold_{cp} \overset{\text{sig}}{+\!\!+} \langle(0, \bot)\rangle \ \langle\rangle \ f \\ = & \{ \text{ Definition of } fold_\alpha \ \} \\ & \langle\rangle \end{array}$$

Inductive Case: Assume

$$c2s(xs' : (t, \bot)) = fold_{CPL} \overset{\text{sig}}{+\!\!+} (xs' : (t, \bot)) \ \langle\rangle \ f$$

then

$$c2s(xs' : (t, \bot) \overset{\text{cp}}{;} (0, \alpha)(t, \bot)) = fold_{CPL} \overset{\text{sig}}{+\!\!+} (xs' : (t, \bot) \overset{\text{cp}}{;} (0, \alpha)(t, \bot)) \ \langle\rangle \ f$$

$$\begin{array}{ll} & c2s(xs' : (t_1, \bot) \overset{\text{cp}}{;} (0, \alpha)(t_2, \bot)) \\ = & \{ \text{ Definition of } \overset{\text{cp}}{;} \ \} \\ & c2s(xs' : (t_1, \alpha) : (t_1 + t_2, \bot)) \\ = & \{ \text{ Definition of c2s } \} \\ & c2s(xs' : (t_1, \bot)) \overset{\text{sig}}{;} (\alpha, t_2) \\ = & \{ \text{ Replacing } \overset{\text{sig}}{;} \text{ by } \overset{\text{sig}}{+\!\!+} \ \} \\ & c2s(xs' : (t_1, \bot)) \overset{\text{sig}}{+\!\!+} \langle(\alpha, t_2)\rangle \\ = & \{ \text{ Definition of f } \} \\ & c2s(xs' : (t_1, \bot)) \overset{\text{sig}}{+\!\!+} f(0, \alpha)(t_2, \bot) \\ = & \{ \text{ Inductive Hypothesis } \} \\ & (fold_{CPL} \overset{\text{sig}}{+\!\!+} (xs' : (t, \bot)) \ \langle\rangle \ f) \overset{\text{sig}}{+\!\!+} f(0, \alpha)(t_2, \bot) \\ = & \{ \text{ Definition } fold_\alpha \ \} \\ & fold_{CPL} \overset{\text{sig}}{+\!\!+} (xs' : (t, \bot) \overset{\text{cp}}{;} (0, \alpha)(t_2, \bot)) \ \langle\rangle \ f \end{array}$$

□

Lemma 40. *The length of a critical point list is equal to the length of the signal obtained by invoking c2s on it. Let xs be a critical point list. Then the following holds*

$$|c2s(xs)| = |xs|$$

Proof. By induction on xs.

Base Case:

$$|c2s(\langle(0,\bot)\rangle)| = |\langle(0,\bot)\rangle|$$

$$\begin{array}{ll} & |c2s(\langle(0,\bot)\rangle)| \\ = & \{ \text{ Definition of c2s } \} \\ & |\langle\rangle| \\ = & \{ \text{ Definition: Length of Empty Signal List } \} \\ & 0 \\ = & \{ \text{ Definition: Length of Empty Critical Point List } \} \\ & |\langle(0,\bot)\rangle| \end{array}$$

Inductive Case: Assume

$$|c2s(xs : (t_1,\bot))| = |xs : (t_1,\bot)|$$

then

$$|c2s(xs : (t_1,\bot) \overset{\text{cp}}{;} (0,\alpha)(t_2,\bot))| = |xs : (t_1,\bot) \overset{\text{cp}}{;} (0,\alpha)(t_2,\bot)|$$

.

$$\begin{array}{ll} & |c2s(xs : (t_1,\bot) \overset{\text{cp}}{;} (0,\alpha)(t_2,\bot))| \\ = & \{ \text{ Definition of } \overset{\text{cp}}{;} \} \\ & |c2s(xs : (t_1,\alpha) : (t_1+t_2,\bot))| \\ = & \{ \text{ Definition of c2s } \} \\ & |c2s(xs : (t_1,\bot)) \overset{\text{sig}}{;} (t_2,\alpha)| \\ = & \{ \text{ Length of a Signal List } \} \\ & |c2s(xs : (t_1,\bot))| + t_2 \\ = & \{ \text{ Inductive Hypothesis } \} \\ & |xs : (t_1,\bot)| + t_2 \\ = & \{ \text{ Length of a CPL } \} \\ & t_1 + t_2 \\ = & \{ \text{ Length of a CPL } \} \\ & |xs : (t_1,\alpha) : (t_1+t_2,\bot)| \\ = & \{ \text{ Definition of } \overset{\text{cp}}{;} \} \\ & |xs : (t_1,\bot) \overset{\text{cp}}{;} (0,\alpha)(t_2,\bot)| \end{array}$$

□

Proposition 41. *The function s2c can be expressed in terms of alpha-fold.*

$$s2c(\xi) = fold_{SIG} \overset{cp}{+\!\!+} \xi\ \langle(0,\bot)\rangle\ f$$

where

$$f :: SIG \to CPL$$

$$f\langle(\alpha,k)\rangle = \langle(0,\alpha),(k,\bot)\rangle$$

Proof. By induction on ξ.

Base Case: $\xi = \langle\rangle$

For s2c we have

$$\begin{array}{ll} & s2c(\langle\rangle) \\ = & \{ \text{ Definition of s2c } \} \\ & \langle(0, \bot)\rangle \end{array}$$

Whilst for the fold version we have

$$\begin{array}{ll} & fold_{SIG} \overset{\text{cp}}{+\!\!+} \langle\rangle \langle(0, \bot)\rangle f \\ = & \{ \text{ Definition of } fold_\alpha \} \\ & \langle(0, \bot)\rangle \end{array}$$

Inductive Case: Assume

$$s2c(\xi') = fold_{SIG} \overset{\text{cp}}{+\!\!+} \xi' f$$

then

$$s2c(\xi' \overset{\text{sig}}{;} (\alpha, k)) = fold_{SIG} \overset{\text{cp}}{+\!\!+} (\xi' \overset{\text{sig}}{;} (\alpha, k)) \langle(0, \bot)\rangle f$$

$$\begin{array}{ll} & s2c(\xi' \overset{\text{sig}}{;} (\alpha, k)) \\ = & \{ \text{ Definition of s2c } \} \\ & s2c(\xi') \overset{\text{cp}}{;} (0, \alpha)(k, \bot) \\ = & \{ \text{ Inductive Hypothesis } \} \\ & fold_{SIG} \overset{\text{cp}}{+\!\!+} \xi' \langle(0, \bot)\rangle f \overset{\text{cp}}{;} (0, \alpha)(k, \bot) \\ = & \{ \text{ Replacing } \overset{\text{cp}}{;} \text{ with } \overset{\text{cp}}{+\!\!+} \} \\ & fold_{SIG} \overset{\text{cp}}{+\!\!+} \xi' \langle(0, \bot)\rangle f \overset{\text{cp}}{+\!\!+} \langle(0, \alpha), (k, \bot)\rangle \\ = & \{ \text{ Definition of f } \} \\ & fold_{SIG} \overset{\text{cp}}{+\!\!+} \xi' \langle(0, \bot)\rangle f \overset{\text{cp}}{+\!\!+} f(\alpha, k) \\ = & \{ \text{ Definition of } fold_\alpha \} \\ & fold_{SIG} \overset{\text{cp}}{+\!\!+} (\xi' \overset{\text{sig}}{;} (\alpha, k)) \langle(0, \bot)\rangle f \end{array}$$

□

Lemma 42. *The length of a signal is equal to the length of the critical point list obtained by invoking s2c on it. Let ξ be a signal list. Then, the following holds.*

$$|s2c(\xi)| = |\xi|$$

Proof. By induction on ξ.

Base Case:

$$|s2c(\langle\rangle)| = |\langle\rangle|$$

$$\begin{array}{ll} & |s2c(\langle\rangle)| \\ = & \{ \text{ Definition of s2c() } \} \\ & |\langle(0,\bot)\rangle| \\ = & \{ \text{ Length of a critical point list } \} \\ & 0 \\ = & \{ \text{ Length of an empty signal list } \} \\ & |\langle\rangle| \end{array}$$

Inductive Case: Assume

$$|s2c(\xi')| = |\xi'|$$

then

$$|s2c(\xi' \overset{\text{sig}}{;} (k,\alpha))| = |\xi' \overset{\text{sig}}{;} (k,\alpha)|$$

$$\begin{array}{ll} & |s2c(\xi' \overset{\text{sig}}{;} (k,\alpha))| \\ = & \{ \text{ Definition of s2c } \} \\ & |s2c(\xi') \overset{\text{cp}}{;} (0,\alpha)(k,\bot)| \\ = & \{ \text{ s2c yields a critical point list } \} \\ & |xs : (t,\bot) \overset{\text{cp}}{;} (0,\alpha)(k,\bot)| \\ = & \{ \text{ Definition of } \overset{\text{cp}}{;} \} \\ & |xs : (t,\alpha) : (t+k,\bot)| \\ = & \{ \text{ Length of critical point list } \} \\ & t+k \\ = & \{ \text{ Length of critical point list } \} \\ & |xs : (t,\alpha)| + k \\ = & \{ \text{ Substituting } s2c(\xi') \text{ back for } xs : (t,\bot) \} \\ & |s2c(\xi')| + k \\ = & \{ \text{ Inductive Hypothesis } \} \\ & |\xi'| + k \\ = & \{ \text{ Length of a Signal List } \} \\ & |\xi' \overset{\text{sig}}{;} (k,\alpha)| \end{array}$$

□

Proposition 43. *Let ξ be a finite signal list. Then $s2c(\xi)$ terminates.*

Proof. By induction on ξ.

Base Case: $\xi = \langle\rangle$
$s2c(\langle\rangle) = \langle(0,\bot)\rangle$ which makes no recursive calls and thus terminates.

Inductive Hypothesis: Assume

$$s2c(\xi') \text{ terminates}$$

then,

$$s2c(\xi' : (k, \alpha)) \text{ terminates}$$

$$\begin{array}{ll} & s2c(\xi' : (k, \alpha)) \\ = & \{ \text{ Definition of s2c() } \} \\ & s2c(\xi') \overset{\text{cp}}{;} (0, \alpha)(k, \bot) \end{array}$$

Thus the call has unwound to $s2c(\xi)$ which by the inductive hypothesis terminates. The primitive $\overset{\text{cp}}{;}$ also terminates, which means as a whole the inductive case is proven.

□

Theorem 44. *Composing s2c with c2s yields the identity function. Let xs be a critical point list. Then the following law holds.*

$$s2c(c2s(xs)) = xs$$

Proof. By induction on xs.

Base Case: $s2c(c2s(\langle(0, \bot)\rangle)) = \langle(0, \bot)\rangle$

$$\begin{array}{ll} & s2c(c2s(\langle(0, \bot)\rangle)) \\ = & \{ \text{ Definition of c2s() } \} \\ & s2c(\langle\rangle) \\ = & \{ \text{ Definition of s2c() } \} \\ & \langle(0, \bot)\rangle \end{array}$$

Inductive Case: Assume

$$s2c(c2s(xs' : (t_1, \bot))) = xs' : (t_1, \bot)$$

then

$$s2c(c2s(xs' : (t_1, \bot) \overset{\text{cp}}{;} (0, \alpha)(t_2, \bot))) = xs' : (t_1, \bot) \overset{\text{cp}}{;} (0, \alpha)(t_2, \bot)$$

$$\begin{array}{ll} & s2c(c2s(xs' : (t_1, \bot) \overset{\text{cp}}{;} (0, \alpha)(t_2, \bot))) \\ = & \{ \text{ Definition of } \overset{\text{cp}}{;} \} \\ & s2c(c2s(xs' : (t_1, \alpha) : (t_1 + t_2, \bot))) \\ = & \{ \text{ Definition of c2s } \} \\ & s2c(c2s(xs' : (t_1, \bot)) \overset{\text{sig}}{;} (\alpha, t_2)) \\ = & \{ \text{ Definition of s2c } \} \\ & s2c(c2s(xs' : (t_1, \bot))) \overset{\text{cp}}{;} (0, \alpha)(t_2, \bot) \\ = & \{ \text{ Inductive Hypothesis } \} \\ & xs' : (t_1, \bot) \overset{\text{cp}}{;} (\alpha, 0)(\bot, t_2) \end{array}$$

□

Theorem 45. *Composing c2s with s2c yields the identity function. Let ξ be a signal list. Then the following law holds*

$$c2s(s2c(\xi)) = \xi$$

Proof. By induction on ξ.

Base Case: Required to prove: $c2s(s2c(\langle\rangle)) = \langle\rangle$

$$\begin{array}{ll} & c2s(s2c(\langle\rangle)) \\ = & \{ \text{ Definition of s2c } \} \\ & c2s(\langle(0,\bot)\rangle) \\ = & \{ \text{ Definition of c2s } \} \\ & \langle\rangle \end{array}$$

Inductive Case: Assume

$$c2s(s2c(\xi')) = \xi'$$

then

$$c2s(s2c(\xi' \overset{\text{sig}}{;} (\alpha, k))) = \xi' \overset{\text{sig}}{;} (\alpha, k)$$

$$\begin{array}{ll} & c2s(s2c(\xi' : (\alpha, k))) \\ = & \{ \text{ Definition of s2c } \} \\ & c2s(s2c(\xi') \overset{\text{cp}}{;} (0,\alpha)(k,\bot) \\ = & \{ \text{ s2c returns a CPL } \} \\ & c2s((xs : (t,\bot)) \overset{\text{cp}}{;} (0,\alpha)(k,\bot)) \\ = & \{ \text{ Definition of } \overset{\text{cp}}{;} \} \\ & c2s(xs : (t,\alpha) : (t+k,\bot)) \\ = & \{ \text{ Definition of c2s } \} \\ & c2s(xs : (t,\bot)) \overset{\text{sig}}{;} (\alpha, r+k-r) \\ = & \{ \text{ Simplification } \} \\ & c2s(xs : (t,\bot)) \overset{\text{sig}}{;} (\alpha, k) \\ = & \{ \text{ Substituting } xs : (t,\bot) \text{ back for } s2c(\xi') \} \\ & c2s(s2c(\xi')) \overset{\text{sig}}{;} (\alpha, k) \\ = & \{ \text{ Inductive Hypothesis } \} \\ & \xi' \overset{\text{sig}}{;} (\alpha, k) \end{array}$$

□

Lemma 48. *The operator $\overset{sig\ glue}{+\!\!+}$ can always be replaced with $\overset{sig}{+\!\!+}$. Let ξ_1 and ξ_2 be signal lists. Then the following holds.*

$$\xi_1 \overset{sig}{;} (a, k_1) \overset{sig\ glue}{+\!\!+} \langle(\alpha, k_2)\rangle \overset{sig}{+\!\!+} \xi_2 = \xi_1 \overset{sig}{+\!\!+} \mu(a, k_1)(\alpha, k_2) \overset{sig}{+\!\!+} \xi_2$$

Proof.

$$\begin{aligned} & \xi_1 \overset{\text{sig}}{;} (a, k_1) \overset{\text{sig glue}}{+\!\!+} \langle (\alpha, k_2) \rangle \overset{\text{sig}}{+\!\!+} \xi_2 \\ = & \quad \{ \text{ Definition of } \overset{\text{sig glue}}{+\!\!+} \ \} \\ & \xi_1 \overset{\text{sig}}{;} (a, k_1) \overset{\text{sig glue}}{;} (\alpha, k_2) \overset{\text{sig}}{+\!\!+} \xi_2 \\ = & \quad \{ \text{ Definition of } \overset{\text{sig glue}}{;} \ \} \\ & \xi_1 \overset{\text{sig}}{+\!\!+} \mu(a, k_1)(\alpha, k_2) \overset{\text{sig}}{+\!\!+} \xi_2 \end{aligned}$$

□

Lemma 50. *The empty signal list is the zero of the* $\overset{\text{sig glue}}{+\!\!+}$ *operator. Let* ξ *be a signal list. Then the following two laws hold.*

$$\xi \overset{\text{sig glue}}{+\!\!+} \langle\rangle = \xi$$

and

$$\langle\rangle \overset{\text{sig glue}}{+\!\!+} \xi = \xi$$

Proof. The first direction follows directly from the definition of $\overset{\text{sig glue}}{+\!\!+}$. We thus prove the second direction by induction on ξ.

Base Case: $\xi = \langle\rangle$

$$\begin{aligned} & \langle\rangle \overset{\text{sig glue}}{+\!\!+} \langle\rangle \\ = & \quad \{ \text{ Definition of } \overset{\text{sig glue}}{+\!\!+} \ \} \\ & \langle\rangle \end{aligned}$$

Inductive Case: Assume

$$\langle\rangle \overset{\text{sig glue}}{+\!\!+} \xi' = \xi'$$

Then,

$$\langle\rangle \overset{\text{sig glue}}{+\!\!+} \xi' \overset{\text{sig}}{;} (a, k) = \xi \overset{\text{sig}}{;} (a, k)$$

$$\begin{aligned} & (\langle\rangle \overset{\text{sig glue}}{+\!\!+} \xi' \overset{\text{sig}}{;} (a, k)) \\ = & \quad \{ \text{ Definition of } \overset{\text{sig glue}}{+\!\!+} \ \} \\ & (\langle\rangle \overset{\text{sig glue}}{+\!\!+} \xi') \overset{\text{sig}}{;} (a, k) \\ = & \quad \{ \text{ Inductive Hypothesis } \} \\ & \xi' \overset{\text{sig}}{;} (a, k) \end{aligned}$$

□

Lemma 51. *The operator* $\overset{\text{sigcons}}{+\!\!+}$ *can always be replaced with the operator* $\overset{\text{sig}}{+\!\!+}$*. Let* ξ_1 *and* ξ_2 *be signal lists. Then, the following holds.*

$$\xi_1 \overset{\text{sigcons}}{+\!\!+} \xi_2 = \xi_1 \overset{\text{sig}}{+\!\!+} \xi_2$$

Proof. By induction on xs.

Base Case: $xs = \langle\rangle$

$$\begin{array}{ll} & \langle\rangle \overset{\text{sigcons}}{+\!\!+} ys \\ = & \{ \text{ Definition of } \overset{\text{sigcons}}{+\!\!+} \} \\ & ys \\ = & \{ \langle\rangle \text{ is the zero of } \overset{\text{sig}}{+\!\!+} \} \\ & \langle\rangle \overset{\text{sig}}{+\!\!+} ys \end{array}$$

Inductive Case: Assume

$$xs \overset{\text{sigcons}}{+\!\!+} ys = xs \overset{\text{sig}}{+\!\!+} ys$$

then

$$((a,k) \overset{\text{sigcons}}{;} xs) \overset{\text{sig}}{+\!\!+} ys = ((a,k) \overset{\text{sigcons}}{;} xs) \overset{\text{sig}}{+\!\!+} ys$$

$$\begin{array}{ll} & ((a,k) \overset{\text{sigcons}}{;} xs \overset{\text{sigcons}}{+\!\!+} ys) \\ = & \{ \text{ Definition of } \overset{\text{sigcons}}{+\!\!+} \} \\ & (a,k) \overset{\text{sigcons}}{;} (xs \overset{\text{sigcons}}{+\!\!+} ys) \\ = & \{ \text{ Inductive Hypothesis } \} \\ & (a,k) \overset{\text{sigcons}}{;} (xs \overset{\text{sig}}{+\!\!+} ys) \\ = & \{ \text{ Definition of } \overset{\text{sigcons}}{;} \} \\ & \langle (a,k) \rangle \overset{\text{sig}}{+\!\!+} (xs \overset{\text{sig}}{+\!\!+} ys) \\ = & \{ \text{ Associativity of } \overset{\text{sig}}{+\!\!+} \} \\ & \langle (a,k) \rangle \overset{\text{sig}}{+\!\!+} xs) \overset{\text{sig}}{+\!\!+} ys \\ = & \{ \text{ Definition of } \overset{\text{sigcons}}{;} \} \\ & ((a,k) \overset{\text{sigcons}}{;} xs) \overset{\text{sig}}{+\!\!+} ys) \end{array}$$

□

Theorem 52. *Let ξ be a (possibly infinite) signal list. Then slice$(\xi)_{[b,e]}$ terminates.*

Proof. We first note that both case one and case four terminate. Cases two and three perform a recursive call each on a list of one item smaller. We shall show that both cases 2 and 3 have to eventually invoke cases 1 or 4, causing termination. What does this imply? It implies that termination therefore depends entirely on the value of e.

How is the value of e altered by cases two and three? It is always passed to the next call as $e - k$. Now, with successive calls, the value of e will range over

$$e,\ e - k_1,\ e - k_1 - k_2, \ldots$$

where k_1, k_2 etc. are the lengths of the successive segments. Now, remember that the lengths of the segments do not exhibit Zeno-like behaviour since this is ruled out by the finite variability requirement. Therefore if the sequence of segments is infinite, the sum

of the lengths $\sum_{\infty}^{i=1} k_i$ will therefore diverge. Since e is finite, this means that there will be some k_i which will either be greater than e (resulting in a Case 4 call) or which will be equal to e which causes a Case 1 call. Therefore, the procedure terminates.

Now let us consider the case where the list is finite. In the definition of the slice function, we have insisted that e has to be smaller or equal to the length of the signal. If the signal is finite, then it has a finite number of tuples n whose lengths are $k_1, \ldots, k_n$. Since the length of the signal is the sum of these lengths, we have

$$|\xi| = \sum_{i=1}^{n} k_n$$

We know that $|\xi| \geq e$. If $|\xi| > e$, then there is some k, which when processed is greater than e. This means that a case 4 call is invoked and that the procedure terminates.

If, on the other hand, $|\xi| = e$, the sequence of subtractions means that eventually, a final call with parameter $[0, 0]$ is made, which results in a case 1 call, terminating.

□

Lemma 53. *The length of the segment that is added to the output, by any invocation of slice is equal to the difference between the lengths of the present interval and the interval of the next call. We can express this as follows. Consider* $slice((\alpha, m) \overset{sigcons}{;} \xi)_{[b_n, e_n]}$, *and suppose that it expands into a call with interval* $[b_{n+1}, e_{n+1}]$, *and adds a segment of length* k_n *to its output. Then, the equality*

$$k_n + e_{n+1} - b_{n+1} = e_n - b_n$$

holds.

Proof. We know that a call to slice can satisfy either of the four cases of the slice function. Let us thus perform a case analysis.

Case One: In case one, no new segment is added, therefore k_n is 0. Since no recursive call is made, $e_{n+1} - b_{n+1}$ is also 0. Since the interval in the first case is a point interval, $e_n - b_n$ has to be 0. Thus for this case, the equality is satisfied.

Case Two: In case two, $e_{n+1} = e_n - m$ and $b_{n+1} = b_n - m$, with k_n being 0. Therefore, $e_{n+1} - b_{n+1} = e_n - b_n$, which satisfies the equality.

Case Three: In case three, $k_n = m - b_n$, $b_{n+1} = 0$ and $e_{n+1} = e_n - m$. Adding up k_n and $e_{n+1} - b_{n+1}$, one gets $e_n - b_n$, as required.

Case Four: In the last case, $k_n = e_n - b_n$, whilst b_{n+1} and e_{n+1} are both 0 (since no recursive calls are made). This means that the equality is satisfied.

□

Theorem 54. *The length of the result of a slice call is equal to the length of the call's interval. Let* ξ *be a signal list. Then* $|slice(\xi)_{[b,e]}| = e - b$.

Proof. By Theorem 52, $slice()$ has to terminate after some finite number of calls n. If one were to show its trace, it will therefore have to call case 2 or case 3 for $n-1$ times, and terminate at either case 1 or case 4, since the latter are the only base cases.

$$\underbrace{slice()_{[b_1,b_1]} \vdash slice()_{[b_2,b_2]} \vdash \ldots \vdash slice()_{[b_{n-1},b_{n-1}]}}_{\text{Case 2 or Case 3}} \vdash \underbrace{slice()_{[b_n,b_n]}}_{\text{Case 1 or Case 4}}$$

As it executes, the interval endpoints change. By the previous Lemma, we know that

$$|k_i| + (e_{i+1} - b_{i+1}) = e_i - b_i$$

or after a small rearrangement,

$$|k_i| = e_i - b_i - (e_{i+1} - b_{i+1})$$

Now, since k_i represent the lengths of the output segments, the length of the result of applying slice to the original signal, is a signal of length

$$\sum_{i=1}^{n-1} k_i + |slice()_{[b_n,e_n]}|$$

For the summation term, we have:

$$\begin{gathered} e_1 - b_1 - (e_2 - b_2) \\ +(e_2 - b_2) - \ldots \\ + \ldots - (e_{n-1} - b_{n-1}) \\ +(e_{n-1} - b_{n-1}) - (e_n - b_n) \end{gathered}$$

which simplifies to:

$$e_1 - b_1 - (e_n - b_n)$$

Now, to this we must add back $slice()_{[b_n,e_n]}$ to obtain

$$e_1 - b_1 - (e_n - b_n) + |slice()_{[b_n,e_n]}|$$

Since the slice call is the terminating call, it is either a Case 4 call or a Case 1 call. If it is a Case 4 call, by expanding the slice() function, we obtain:

$$e_1 - b_1 - (e_n - b_n) + (e_n - b_n)$$

which is essentially

$$e_1 - b_1$$

as expected. Meanwhile, if we expand it as a Case 1 call, then $e_n - b_n$ is equal to zero, whilst the length of the output of the slice call is also 0. This leaves the result $e_1 - b_1$.

□

Lemma 55. *Calling slice with interval $[b, e]$ on a signal is equivalent to gluing the signals obtained from calling slice with interval $[b, m]$ and $[m, e]$, for any m between b and e inclusive. We can express this as follows. Let ξ be a signal list. Then for $b \leq m \leq e$,*

$$slice(\xi)_{[b,e]} = slice(\xi)_{[b,m]} \overset{sig\ glue}{+\!\!+} slice(\xi)_{[m,e]}$$

Proof. We note that all three slice functions are operating on the same signal list. Essentially, what we wish to show is that for every tuple in ξ, if when it is processed by $slice()_{[b,e]}$, a (possibly modified) tuple is added to the output, then the same modified tuple must also be added by $slice(\xi)_{[b,m]} \overset{\text{sig glue}}{+\!\!+} slice(\xi)_{[m,e]}$. Formally we want the following statement to be true.

$$slice((a,k) \overset{\text{sigcons}}{;} \xi')_{[b_1,e_1]} = (a,l) \overset{\text{sigcons}}{;} slice(\xi')_{[b_2,e_2]} \Rightarrow$$

$$slice(\xi)_{[b_1,m_1]} \overset{\text{sig glue}}{+\!\!+} slice(\xi)_{[m_1,e_1]} = (a,l) \overset{\text{sigcons}}{;} slice(\xi)_{[b_2,m_2]} \overset{\text{sig glue}}{+\!\!+} slice(\xi)_{[m_2,e_2]} \ldots (1)$$

This statement allows us to reduce the lists on which the slice functions operate by one tuple. If we can also prove that both sides of the equality will have terminated when they encounter a certain tuple (α_i, k_i), then we can use induction on ξ to prove the theorem. Therefore, for the base case, we need to prove the following.

$$Teminates(slice((a_i,k_i) \overset{\text{sigcons}}{;} \xi)_{[b_1,m_1]}) \Rightarrow$$

$$Terminates(slice((a_i,k_i) \overset{\text{sigcons}}{;} \xi)_{[b_1,m_1]} \overset{\text{sig glue}}{+\!\!+} slice((a_i,k_i) \overset{\text{sigcons}}{;} \xi)_{[m_1,e_1]}) \ldots (2)$$

Before starting the actual proof, we note that the length of the tuple being examined, k, can relate in 4 ways to b, m and e. This is shown in figure 6.1.

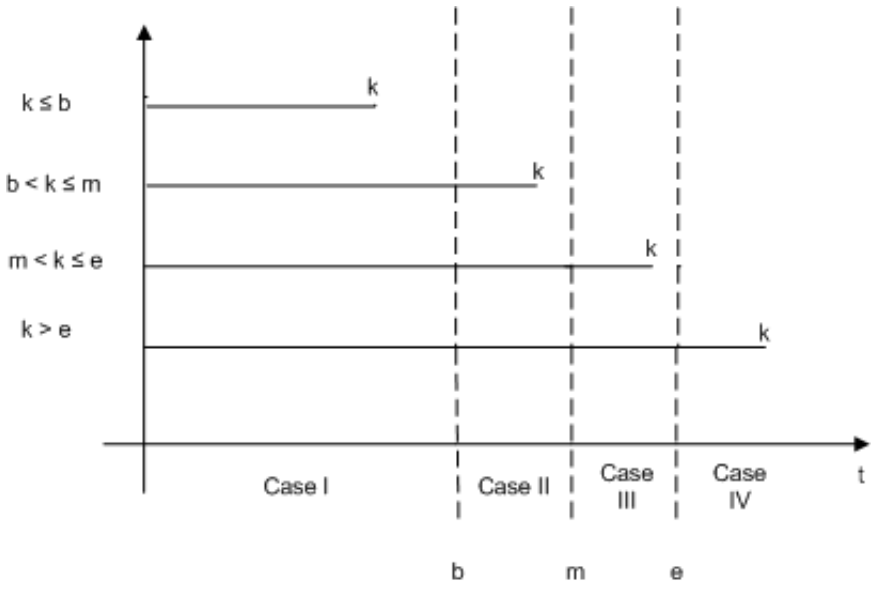


Figure 6.1: Relating k with b, e and m

For each of these cases, we tabulate the reaction of the slice function on the intervals $[b, e]$, $[b, m]$ and $[m, e]$ to tuples (a, k) in each of these four cases. We call these cases I, II, III and IV. By reaction we mean the tuple it adds to the output.

	$[b, e]$	$[b, m]$	$[m, e]$
Case I ($k \leq b$)	-	-	-
Case II ($b < k \leq m$)	(a,k-b)	(a,k-b)	-
Case III ($m < k \leq e$)	(a,k-b)	(a,m-b)	(a,k-m)
Case IV ($k > e$)	(a,e-b)	(a,m-b)	(a,e-m)

We now start with the actual proof, which we shall perform by Reverse Induction.

Reverse Inductive Case As we have seen above, when we process a tuple, there are four possible cases of what can happen. To this end, we shall have to prove the result (1) for all four cases. What we shall do is work out the value of stepping once through the slice functions for both $slice_{[b,e]}$ and $slice_{[b,m]} \overset{\text{sig glue}}{+\!\!+} slice_{[m,e]}$. We must prove that when one adds a tuple to the output, the other expression also adds the same tuple, and that the interval endpoints for the next call of one side are consistent with the interval endpoints of the next call for the other side, as per the definition in (1).

Case I. For the left hand side of the implication we know that

$$slice((a, k) \overset{\text{sigcons}}{;} \xi')_{[b_1,e_1]} = slice(\xi')_{[b_1-k,e_1-k]}$$

whilst if we expand the right hand side we get

$$slice((a, k) \overset{\text{sigcons}}{;} \xi)_{[b_1,m_1]} \overset{\text{sig glue}}{+\!\!+} slice(\xi)_{[m_1,e_1]}$$

$$= slice(\xi)_{[b_1-k,m_1-k]} \overset{\text{sig glue}}{+\!\!+} slice(\xi)_{[m_1-k,e_1-k]}$$

as required.

Case II. In this case, for the left hand side of the implication we know

$$slice((a, k) \overset{\text{sigcons}}{;} \xi')_{[b_1,e_1]} = (a, k - b) \overset{\text{sigcons}}{;} slice(\xi')_{[0,e_1-k]}$$

and we can expand the right hand side to get

$$slice((a, k) \overset{\text{sigcons}}{;} \xi)_{[b_1,m_1]} \overset{\text{sig glue}}{+\!\!+} slice(\xi)_{[m_1,e_1]}$$

$$= (a, k - b) \overset{\text{sigcons}}{;} slice(\xi)_{[0,m_1-k]} \overset{\text{sig glue}}{+\!\!+} slice(\xi)_{[m_1-k,e_1-k]}$$

as required.

Case III. In Case III, the left hand side of the implication results in

$$slice((a, k) \overset{\text{sigcons}}{;} \xi')_{[b_1,e_1]} = (a, k - b) \overset{\text{sigcons}}{;} slice(\xi')_{[0,e_1-k]}$$

and the right hand side results in

$$slice((a, k) \overset{\text{sigcons}}{;} \xi)_{[b_1,m_1]} \overset{\text{sig glue}}{+\!\!+} slice(\xi)_{[m_1,e_1]}$$

$$= \langle (a, m - b) \rangle \overset{\text{sig glue}}{+\!\!+} (a, k - m) \overset{\text{sigcons}}{;} slice(\xi)_{[0,e_1-k]}$$

We now note that in this case, the first slice call on the right hand side terminates. What about enforcing the consistency of the interval endpoints? The second call on the right hand side is consistent with the requirements. What requirements would the first call on the right hand side have had to satisfy? It would have had to operate on the interval $[0, 0]$. We shall see that we can easily fix this. However, let us simplify the expression for the right hand side of the implication first, so that it actually adds the same tuple as the left hand side of the implication .

$$\langle (a, m-b) \rangle \overset{\text{sig glue}}{+\!\!+} (a, k-m) \overset{\text{sigcons}}{;} slice(\xi)_{[0,e_1-k]}$$

Since we know that we can replace $\overset{\text{sigcons}}{;}$ with $\overset{\text{sigcons}}{+\!\!+}$, we get

$$\langle (a, m-b) \rangle \overset{\text{sig glue}}{+\!\!+} \langle (a, k-m) \rangle \overset{\text{sigcons}}{+\!\!+} slice(\xi)_{[0,e_1-k]}$$

We then expand the $\overset{\text{sig glue}}{+\!\!+}$ operation, to get

$$\langle \rangle \overset{\text{sig}}{;} (a, m-b) \overset{\text{sig glue}}{;} (a, k-m) \overset{\text{sigcons}}{+\!\!+} slice(\xi)_{[0,e_1-k]}$$

By getting rid of $\overset{\text{sig glue}}{;}$, we get

$$\langle \rangle \overset{\text{sig}}{+\!\!+} \mu(a, m-b)(a, k-m) \overset{\text{sigcons}}{+\!\!+} slice(\xi)_{[0,e_1-k]}$$

And by eliminating μ this simplifies to

$$\langle \rangle \overset{\text{sig}}{+\!\!+} \langle (k-b) \rangle \overset{\text{sigcons}}{+\!\!+} slice(\xi)_{[0,e_1-k]}$$

Since empty signal lists are the zero of $\overset{\text{sig}}{+\!\!+}$ we get

$$\langle (k-b) \rangle \overset{\text{sigcons}}{+\!\!+} slice(\xi)_{[0,e_1-k]}$$

and since $\langle (k-b) \rangle$ is a singleton list, we can simplify this to

$$(k-b) \overset{\text{sigcons}}{;} slice(\xi)_{[0,e_1-k]}$$

In this manner, we have seen that the right hand side had added exactly the same tuple as the left hand side. What about the termination problem we had earlier? By some simple manipulation, we can glue an empty signal list to $slice(\xi)_{[m_1-k,e_1-k]}$, since this is a zero under $\overset{\text{sig glue}}{+\!\!+}$.

$$(k-b) \overset{\text{sigcons}}{;} \langle \rangle \overset{\text{sig glue}}{+\!\!+} slice(\xi)_{[0,e_1-k]}$$

But an empty signal list is exactly a call of slice over a point interval, which in order to fulfill the theorem we chose to be over [0,0].

$$(k-b) \overset{\text{sigcons}}{;} slice(\xi)_{[0,0]} \overset{\text{sig glue}}{+\!\!+} slice(\xi)_{[0,e_1-k]}$$

Case IV For the final case, the left hand side of the implication is

$$slice((a,k) \overset{\text{sigcons}}{;} \xi')_{[b_1,e_1]} = \langle (a, e-b) \rangle$$

and the right hand side results in

$$slice((a,k) \overset{\text{sigcons}}{;} \xi)_{[b_1,m_1]} \overset{\text{sig glue}}{+\!\!+} slice(\xi)_{[m_1,e_1]} = \langle (a, m-b) \rangle \overset{\text{sig glue}}{+\!\!+} \langle (e-m) \rangle$$

If we simplify the right hand side of the implication

$$\langle (a, m-b) \rangle \overset{\text{sig glue}}{+\!\!+} \langle (e-m) \rangle$$

we get

$$\langle \rangle \overset{\text{sig}}{;} (a, m-b) \overset{\text{sig glue}}{;} (e-m)$$

which simplifies to

$$\langle \rangle \overset{\text{sig}}{;} \mu(a, m-b)(a, e-m)$$

By eliminating the μ operator we get

$$\langle \rangle \overset{\text{sig}}{;} (b-e)$$

which simplifies to

$$\langle (b-e) \rangle$$

which is in agreement with result of the left hand side of the implication. Since all the functions terminate in this case we do not need to show that the interval endpoints between the next calls of left hand side and right hand side of the implications agree, as there are no next calls to be performed.

Base Case: Termination We know that $slice((a_i,k_i) \overset{\text{sigcons}}{;} \xi)_{[b_1,m_1]}$ terminates. Why has it terminated? In order to terminate, it has to have satisfied the guard for case 1 or case 4. If it has satisfied the guard for case 1, then, $b_1 = e_1$. Since m_1 must lie between b_1 and e_1, then we have $b_1 = e_1 = m_1$. However, this means that the calls on the right hand side are also operating on a point interval, and hence, they must also terminate.

Now suppose that the right hand side terminates because it has satisfied the guard for case 4. Remember that termination depends on the value of the right interval endpoint of the call to slice. If $slice()_{[b_1,e_1]}$ has not terminated up to this point, $slice()_{[m_1,e_1]}$ cannot have terminated yet, since it shares the same right interval endpoint e_1. On the other hand $slice()_{[b_1,m_1]}$ might have already terminated (since $m_1 < e_1$), or it might not. If it has not already terminated up to this point, we note that since $k_i > e_1$, then $k_i > m_1$, which means that both expressions on the right hand side must terminate with a case 4 call.

□

Theorem 56. *Consider the execution of slice(). At each step it can unroll into cases 1,2,3 or 4. Then the execution trace of slice() is of one of the following forms.*

$$1$$

$$2^* \vdash 4$$

$$2^* \vdash 3^+ \vdash (1|4)$$

Proof. Consider examining a tuple (α, k). At this point, k can satisfy either of the four cases. If the execution hits case 1 or 4 it terminates. Since these are the only terminating cases, the run must end with either case 1 or 4.

Next we ask with what cases a run can potentially begin. When we consider this question, we note that it is admissible for a run to begin with either cases 1, 2 or 3 or 4.

- Beginning with a case 1 call only requires that the interval parameter be a point interval, which is admissible.
- Beginning with a case 2 call requires that the length of the first tuple k be $\leq b$, which is admissible.
- Beginning with a case 3 call requires that the length of the first tuple k to be such that $b < k \leq e$.
- Beginning with a case 4 call requires that the length of the first tuple k to be such that $k > e$

We now consider the guards that the next call can satisfy, given that the present call has satisfied a certain guard. For example, suppose that the present call $slice((\alpha, k) \overset{\text{sigcons}}{;} \xi)_{[b,e]}$, has satisfied the guard $k < b$. This means that it unwinds into the next call $slice(\xi)_{[b-k,e-k]}$. Given these constraints, what guards can the next call satisfy? We tabulate the possible successors given that the present call is of a certain type (satisfies a certain guard). The rows shall give the type of the present call and the columns will give the type of the successor call.

	1	2	3	4
1	no	no	no	no
2	no	yes	yes	yes
3	yes	no	yes	yes
4	no	no	no	no

Let us see how we arrived to this conclusion. Since Case 1 and Case 4 calls terminate, they cannot have any successors. If the present call is a case 2 call,

- *It cannot resolve into a case 1 call.* If a case 2 guard has been satisfied, then $b \neq e$, since otherwise a case 1 call would have occurred. Now, a case 2 call subtracts the length of the tuple k from both endpoints. This means that if $b \neq e$, $b - k \neq e - k$. It is also the case that k, which in this case is less than b, is not big enough to subtract both to 0.
- *Can resolve into either cases 2, 3 or 4*, since one cannot control the length of the next tuple, and both interval endpoints cannot become 0.

We now consider case 3 calls. Such calls

- *Cannot resolve into a case 2 call.* This is because the next call is made on the interval $[0, e-k]$. Recall that case 2 requires k to be less than b, and no k can be less than 0. Thus, once case three occurs, case two cannot occour again.
- *Can resolve into case 1, 3 and 4 calls.* Case 1 would occour if $e = k$, which would result in a call with interval $[0, 0]$ being made. Cases 3 and 4 can occour since the only requirement is for the next k to be either between 0 and e or to be greater than e, which is possible.

The reader can verify that the above information can be distilled into the following regular expression like structures for traces.

$$1$$

$$2^* \vdash 4$$

$$2^* \vdash 3^+ \vdash (1|4)$$

which concludes the theorem.

□

Lemma 58. *Calling slice over a signal divides it into 3 pieces. The original signal can be recovered by using gluing concatenation. We express this as follows. Let ξ is a signal list, then*

$$\forall b, e : \mathbb{R}.\ b \leq e \Rightarrow\ \exists\, \xi_1, \xi_2 : SIG.\ \xi = (\xi_1 \overset{sig\ glue}{+\!\!+} slice(\xi)_{[b,e]} \overset{sig\ glue}{+\!\!+} \xi_2)$$

Proof. We shall prove this theorem, by construction, by showing how to build ξ_1 and ξ_2.

Let us consider how *slice* works. *slice* goes through the tuples (α, k) in the signal list. For each tuple it will do some processing by invoking some particular rule, and possibly add some segment to its output. However, in certain scenarios, the processing it does will remove some tuples or parts of a tuple from the input. We shall see that these discarded parts end up in either ξ_1 or ξ_2.

We now show a procedure of how to build ξ_1 and ξ_2 depending on which guard it triggers when processing a tuple. The construction is tabulated below. Each entry of the table tells us how that particular tuple contributes to either ξ_1, to the output of slice or to ξ_2.

	ξ_1	$slice(\xi)_{[b,e]}$	ξ_2
Case 1 (terminating)	$slice(\xi)_{[0,e]}$	-	$slice(\xi)_{[e,\lvert\xi\rvert]}$
Case 2	(α, k)	-	-
Case 3	(α, b)	$(\alpha, k-b)$	-
Case 4 (terminating)	(α, b)	$(\alpha, e-b)$	(k-e) $\overset{\text{sigcons}}{;}$ xs

Some explanation is perhaps due for the definition of Case 1. Case 1 can be called with a variety of point parameters. It makes sense to assign the part to the left of the point parameter to ξ_1 and the part to the right to ξ_2. To do this, we take advantage of the slice function itself, which is a well defined and terminating computation.

We now have to prove that once one has obtained the expressions for ξ_1, ξ_2 and slice, these expressions, once concatenated, will yield back ξ. How can we prove this global condition, given that we are processing only a tuple at a time? What conditions must hold when processing a tuple such that the global constraint is met?

Essentially we want that the tuples are first added to ξ_1, then to the output of *slice* and finally to ξ_2, in that order. This ordering also implies that once we add a tuple to some category, we cannot add tuples to previous categories. We also allow tuples to be split such that they overlap categories, such that the ordering condition is still met. When such a split is made, it must be such that if we were to add back the parts, we would get the original tuple.

We shall prove that the various traces which slice can yield comply to these requirements. Now, remember that due to the Theorem 56, the execution trace of slice must be of the form:

$$1$$

$$2^* \vdash 4$$

$$2^* \vdash 3^+ \vdash (1|4)$$

Let us consult the table for our construction and check whether the traces satisfy the properties.

Trace: 1. In this trace, we know that $\xi_1 = slice(\xi)_{[0,e]}$, $\xi_2 = slice(\xi)_{[e,|\xi|]}$, and that the output of the slice function is empty. By theorem 55, we know that the gluing concatenation of ξ_1 and ξ_2, is $slice(\xi)_{[0,|\xi|]}$, which is in fact ξ.

Trace: $2^* \vdash 4$. Suppose that case two is optional. If this is the case, the expression results in a case 4 call. In this case, the tuple being processed is split into (α, b), $(\alpha, e-b)$ and $(\alpha, e-k)$, with the rest of the signal following this tuple. Adding the lengths yields k, as required, and since the rest of the signal is in ξ_2, it is clear that the concatenation will yield the input signal. If case two is not optional, the tuple being processed is appended as a whole, to ξ_1. If case two is repeated, the same will happen. Eventually, a tuple will occour which will trigger case 4, which will split the tuple amongst the three categories, and append the rest of the signal to ξ_2. If we concatenate all this together, we get back all of the original tuples in the required order.

Trace: $2^* \vdash 3^+ \vdash (1|4)$. Suppose case 2 is not optional. If this occurs, every case 2 call made, will append the tuple processed to ξ_1. When case 3 is executed for the first time, the tuple is split in such a way that it sums to the original length, with a part in ξ_1 and a part in the slice segment. There can at this point be a number of other case 3 calls. We note that case 3 usually splits the tuple into two pieces, one going to ξ_1 and one to the slice segment. However, the first case 3 call sets b to 0. This means that in al future case 3 calls, no length is in fact added to ξ_1 and all the length is added to the slice part, which is consistent with what we want. The run terminates with either case 1 or case 4 being invoked. In the case where case 1 is invoked, due

to Corollary 57, we know that this call has to be made over the interval $[0, 0]$, which yields an empty list, and therefore does not affect the conclusion of the argument. This kind of trace is therefore consistent with the rules. What if the terminating case is 4 instead of 1? In this case, since b is 0, nothing is added to ξ_1, part of the segment is added to the slice section and the rest of the segment and the signal, to ξ_2. The concatenation of all this also follows the rules.

The only thing left to consider is the case when case 2 is treated as optional. In this case, it is easy to see that the argument is not affected, since it is simply the case of beginning with a case 3 call instead of a case 2 call, which, as we have already seen, satisfies the conditions we have set out earlier.

□

Theorem 59. *The slice function divides its input into three pieces. The length of the first piece is equal to b, the original left endpoint. We express this below. Consider* $slice(\xi)_{[b,e]}$. *Recall that* $\xi = \xi_1 \overset{sig\ glue}{+\!+} slice(\xi)_{[b,e]} \overset{sig\ glue}{+\!+} \xi_2$. *Then,* $|\xi_1| = b$

Proof. In order to prove this result we will note that the initial call operates on the interval $[b, e]$. As the calls are invoked, this interval changes. We shall split our proof into two segments. In the first part, we will note that the amount subtracted from b, till it becomes 0 is always equal to the length added to ξ_1. In the second part, we shall use this result to prove the actual theorem. The basic idea is that due to the first result, when b becomes zero, the total length of ξ_1 will have to be b. Let us formulate the statement for part 1.

Lemma: *Consider* $slice(\xi)_{[b,e]}$. *Recall that* $\xi = \xi_1 \overset{sig\ glue}{+\!+} slice(\xi)_{[b,e]} \overset{sig\ glue}{+\!+} \xi_2$. *Let* b_i *be the left interval endpoint of the* i^{th} *call to slice, and* j_i *the length of the increment (in length) to* ξ_1 *during the* i^{th} *call. Then this increment is equal to the difference between the successive left interval endpoints*

$$j_i = b_i - b_{i-1}$$

Proof. By induction on the number of calls left.

Base Case, calls left = 0 If the number of calls left is 0, then the present call to slice must be either a Case 4 or a Case 1 call.

Consider Case 4. In this case, by looking at the table for the construction in theorem 58, we know that j_i is equal to b_i, with b_{i-1} not contributing any other amount since no more calls are made.

If the present call is a case 1 call, we know that it adds a tuple of length $|slice(\xi_1)_{[0,e_i]}|$ to ξ_1, which corresponds to j_i. By theorem 54, we know that the length of the slice call over $[0, e_i]$ is $e_i - 0$. j_i, the amount of length added in this case is therefore e_i, which is equal to b_i (Remember that in a case 1 call b = e). Also since the call terminates, b_{i-1} does not contribute any amount.

Inductive Case Assume that for k calls, the equality $j_k = b_k - b_{k-1}$ holds. We are required to prove that $j_{k+1} = b_{k+1} - b_k$. Since only cases 2 and 3 are recursive, we have to prove this result for when the present call corresponds to these cases.

If the present call satisfies the guard for the second case, then from the definition of the slice function, we know that the relationship between b_{k+1} and b_k can be formulated as follows.

$$b_k = b_{k+1} - j_{k+1}$$

as required.

For case 3, from the table, $j_{k+1} = b_{k+1}$, whilst $b_k = 0$, from the definition of the slice function. This means that $j_{k+1} = b_{k+1} - b_k$, as required.

□

Now that we have the above lemma, we can continue with the argument of the original theorem. We start by noting that the slice call terminates, so it will unroll itself into a finite number of recursive calls n. Let j_i be the i^{th} increment (signal segment addition) to ξ_1 as per the table in Lemma 58. Then, the length of ξ_1, is the sum of the increments at each call, which we know to be

$$|\xi_1| = \underbrace{\sum_{i=2}^{n} j_i}_{\text{Case 2 and Case 3 calls}} + \underbrace{j_1}_{\text{Case 1 or 4 call}}$$

However, by the previous lemma, we know that

$$j_i = b_i - b_{i-1}$$

where b_i is the left interval endpoint in the i^{th} call of slice. By substituting this formula into the summation, one gets,

$$b_2 - b_1$$
$$b_3 - b_2$$
$$b_4 - \ldots$$
$$\ldots - b_{n-3}$$
$$b_{n-1} - b_{n-2}$$
$$b_n - b_{n-1}$$

which is equal to $b_n - b_1$

If the terminating call is a case 4 call, the length of ξ_1 is

$$b_n - b_1 + j_1$$

What is the value of j_1? It is the increment done to ξ_1 via a case 4 call. This is clearly b_1. Therefore, it is easy to see that the length of ξ_1 is in fact b_n, which is the initial interval left endpoint, which proves the theorem in this case.

Suppose now that the terminating call is a case 1 call. In this case, b_1 is zero since by Corollary 57 we know that case 1 can be called by any other case only when the interval endpoints are both 0. This means that $|\xi_1|$ is also b_n in this case, which proves the theorem.

□

Theorem 60. *If we can divide the input to slice such that the length of the first part is less than the left interval endpoint b, then this is equivalent to calling slice on the second part with altered endpoints. We formulate this as follows. Let ξ_1 and ξ_2 be signal lists. If $|\xi_1| \leq b$, then*

$$slice(\xi_1 \overset{sigcons}{+\!\!+} \xi_2)_{[b,e]} = slice(\xi_2)_{[b-|\xi_1|,e-|\xi_1|]}$$

Proof. By induction on ξ_1.

Base Case: $\xi_1 = \langle\rangle$.

$$\begin{array}{ll} & slice(\langle\rangle \overset{\text{sigcons}}{+\!\!+} \xi_2)_{[b,e]} \\ = & \{\ \langle\rangle \text{ is the zero of } \overset{\text{sig}}{+\!\!+}\ \} \\ & slice(\xi_2)_{[b,e]} \\ = & \{\ \text{x - 0 = 0}\ \} \\ & slice(\xi_2)_{[b-0,e-0]} \\ = & \{\ \text{Length of empty list is 0}\ \} \\ & slice(\xi_2)_{[b-|\xi_1|,e-|\xi_1|]} \end{array}$$

Inductive Case: Assume

$$slice(\xi_1' \overset{\text{sigcons}}{+\!\!+} \xi_2)_{[b,e]} = slice(\xi_2)_{[b-|\xi_1'|,e-|\xi_1'|]}$$

and that

$$|(a,k) \overset{\text{sigcons}}{;} \xi_1' \overset{\text{sigcons}}{+\!\!+} \xi_2| \leq b$$

then

$$slice((a,k) \overset{\text{sigcons}}{;} \xi_1' \overset{\text{sigcons}}{+\!\!+} \xi_2)_{[b,e]} = slice(\xi_2)_{[b-|(a,k)\overset{\text{sigcons}}{;}\xi_1'|,e-|(a,k)\overset{\text{sigcons}}{;}\xi_1'|]}$$

$$\begin{array}{ll} & slice((a,k) \overset{\text{sigcons}}{;} \xi_1' \overset{\text{sigcons}}{+\!\!+} \xi_2)_{[b,e]} \\ = & \{\ |(a,k) \overset{\text{sigcons}}{;} \xi_1'| \leq b \Rightarrow k \leq b, \text{ case 2 slice call}\ \} \\ & slice(\xi_1' \overset{\text{sigcons}}{+\!\!+} \xi_2)_{[b-k,e-k]} \\ = & \{\ \text{Inductive Hypothesis}\ \} \\ & slice(\xi_2)_{[b-k-|\xi_1'|,e-k-|\xi_1'|]} \\ = & \{\ \text{Grouping terms together}\ \} \\ & slice(\xi_2)_{[b-(|\xi_1'|+k),e-(|\xi_1'|+k)]} \\ = & \{\ \text{Definition } |(a,k) \overset{\text{sigcons}}{;} \xi_1'|\ \} \\ & slice(\xi_2)_{[b-|(a,k)\overset{\text{sigcons}}{;}\xi_1'|,e-|(a,k)\overset{\text{sigcons}}{;}\xi_1'|]} \end{array}$$

□

Theorem 61. *ε's interpretation semantics are sound with respect to its signal list semantics. Let i be an interpretation. Then,*

$$i \vDash^{I}_{[b,e]} \varepsilon \Rightarrow c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \varepsilon$$

Proof.

$$\begin{array}{ll} & i \vDash^{I}_{[b,e]} \varepsilon \\ \iff & \{ \text{ Definition of } \vDash^{I}_{[b,e]} \varepsilon \ \} \\ & b = e \\ & \{ \text{ Definition of slice over point interval } \} \\ & slice(c2s(i2c(i)))_{[e,e]} = \langle\rangle \\ \iff & \{ \text{ b = e } \} \\ & slice(c2s(i2c(i)))_{[b,e]} = \langle\rangle \\ \iff & \{ \text{ Definition of } \in \text{ for singleton set } \} \\ & slice(c2s(i2c(i)))_{[b,e]} \in \{\langle\rangle\} \\ \iff & \{ \text{ Definition of } \vDash^{SIG}_{[b,e]} \varepsilon \ \} \\ & c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \varepsilon \end{array}$$

□

Theorem 62. *a's interpretation semantics are sound with respect to its signal list semantics. Let i be an interpretation. Then,*

$$i \vDash^{I}_{[b,e]} a \Rightarrow c2s(i2c(i)) \vDash^{SIG}_{[b,e]} a$$

Proof. Assuming the antecedent, we want to prove that $c2s(i2c(i)) \vDash^{SIG}_{[b,e]} a$. We shall first simplify the consequent and then attempt to prove the simplified formula.

$$\begin{array}{ll} & c2s(i2c(i)) \vDash^{SIG}_{[b,e]} a \\ \iff & \{ \text{ Definition } \vDash^{SIG}_{[b,e]} a \ \} \\ & slice(c2s(i2c(i)))_{[b,e]} \in \{a^r \mid r : \mathbb{R}^+\} \\ \iff & \{ \text{ Definition } \in \ \} \\ & \exists r : \mathbb{R}^+.\ slice(c2s(i2c(i)))_{[b,e]} = a^r \\ \iff & \{ \text{ Syntactic Sugar } \} \\ & \exists r : \mathbb{R}^+.\ slice(c2s(i2c(i)))_{[b,e]} = \langle(a, r)\rangle \end{array}$$

Now, what does the antecedent tell us?

$$\begin{array}{ll} & i \vDash^{I}_{[b,e]} a \\ \iff & \{ \text{ Definition } \vDash^{I}_{[b,e]} a \ \} \\ & \forall t : \mathbb{T}.\ b \leq t < e \Rightarrow i(a, t) \end{array}$$

Therefore, what we know about i, is that there is a period of time in the interval $[b, e]$ for which the symbol a attains the value of true. Using this knowledge, we shall now build, step by step the formula

$$slice(c2s(i2c(i)))_{[b,e]}$$

in order to see if its value really is $\langle(a, r)\rangle$ for some r. We first consider $i2c(i)$. We know that i, attains the value of true at some point, for the symbol a. It therefore follows that i cannot be empty. Thus, by Proposition 28, $i2c(i)$ will have at least two critical points. Now, in the interval $[b, e)$, $i(a, t)$ is always true. This means that there cannot be a critical point in the interval (b,e), since by Definition 25, one of the conditions required for a critical point is a change in value. Therefore, as a consequence of the above two facts, the interval (b, e) lies between some two consecutive critical points. Let these be called c_1 and c_2. Note that c_1 is thus either equal to b or smaller. Also, c_2 is either exactly equal to e or greater. By Proposition 29, we know that between these two points the value of the interpretation cannot change. Therefore, we have two critical points (c_1, a) and (c_2, α), for some α. Since $i2c(i)$ is obtained by ordering the critical points of i, $i2c(i)$ will therefore yield a critical point list of the form $xs : (c_1, a) : (c_2, \bot) \overset{\text{cp}}{+\!\!+} ys$, with the critical point lists xs and ys possibly empty.

Now, let us consider $c2s(i2c(i))$. This is equal to

$$c2s(xs : (c_1, a) : (c_2, \bot) \overset{\text{cp}}{+\!\!+} ys)$$

Now, remember that we can express c2s in terms of folds. This would yield the expression

$$fold \overset{\text{sig}}{+\!\!+} (xs : (c_1, a) : (c_2, \bot) \overset{\text{cp}}{+\!\!+} ys) \langle\rangle f$$

We also know that $\overset{\text{sig}}{+\!\!+}$ is associative. Therefore, by Theorem 19, we know that we can write the above expression as

$$fold \overset{\text{sig}}{+\!\!+} xs : (c_1, a) : (c_2, \bot) \langle\rangle f \overset{\text{sig}}{+\!\!+} fold \overset{\text{sig}}{+\!\!+} ys \langle\rangle f$$

Since the above folds will return a signal list, we introduce the signal list variable ξ_y and write

$$fold \overset{\text{sig}}{+\!\!+} xs : (c_1, a) : (c_2, \bot) \langle\rangle f \overset{\text{sig}}{+\!\!+} \xi_y$$

We now replace the c2s fold definition with a standard c2s definition to obtain

$$c2s(xs : (c_1, a) : (c_2, \bot)) \overset{\text{sig}}{+\!\!+} \xi_y$$

If we use the definition of c2s, we get

$$c2s(xs : (c_1, \bot)) \overset{\text{sig}}{;} (a, c_2 - c_1) \overset{\text{sig}}{+\!\!+} \xi_y$$

Therefore, we can conclude that the signal list arising from $i2s()$ will have the form:

$$\xi_x \overset{\text{sig}}{;} (a, c_2 - c_1) \overset{\text{sig}}{+\!\!+} \xi_y$$

where ξ_x is another signal list variable.

Moreover, we note that ξ_x arises from $c2s(xs : (c_1, \bot))$. By Lemma 40, we know that

$$|c2s(xs : (c_1, \bot))| = |xs : (c_1, \bot)|$$

We also know that due to the definition of the length of a critical point list, $|xs : (c_1, \bot)| = c_1$. Therefore $|\xi_x| = c_1$.

We now have to consider the expression $slice(c2s(i2c(i)))_{[b,e]}$. More specifically, we will consider:

$$slice(\xi_x \overset{\text{sig}}{;} (a, c_2 - c_1) \overset{\text{sig}}{+\!\!+} \xi_y)_{[b,e]}$$

Before we continue we change all the *snoc-like* operators to *cons-like* operators. We know how to exchange a $\overset{\text{sig}}{+\!\!+}$ with $\overset{\text{sigcons}}{+\!\!+}$. So the first step is to transform $\overset{\text{sig}}{;}$ into $\overset{\text{sig}}{+\!\!+}$.

$$slice(\xi_x \overset{\text{sig}}{+\!\!+} \langle (a, c_2 - c_1) \rangle \overset{\text{sig}}{+\!\!+} \xi_y)_{[b,e]}$$

We then effect the replacement of the concatenation operators.

$$slice(\xi_x \overset{\text{sigcons}}{+\!\!+} \langle (a, c_2 - c_1) \rangle \overset{\text{sigcons}}{+\!\!+} \xi_y)_{[b,e]}$$

Since the length of ξ_x is equal to c_1 which is smaller than b, we can use Theorem 60 to ignore ξ_x entirely, ending up with

$$slice(\langle (a, c_2 - c_1) \rangle \overset{\text{sigcons}}{+\!\!+} \xi_y)_{[b-|\xi_x|,e-|\xi_x|]}$$

Since the left hand side of $\overset{\text{sigcons}}{+\!\!+}$ is a singleton signal list, we can replace $\overset{\text{sigcons}}{+\!\!+}$ with $\overset{\text{sigcons}}{;}$.

$$slice((a, c_2 - c_1) \overset{\text{sigcons}}{;} \xi_y)_{[b-|\xi_x|,e-|\xi_x|]}$$

Now, we have already remarked that $|\xi_x|$ is equal to c_1, which means that our expression can be written as

$$slice((a, c_2 - c_1) \overset{\text{sigcons}}{;} \xi_y)_{[b-c_1,e-c_1]}$$

If we were to expand the slice call, what rule would be applied in this case? We know that $c_2 \geq e$. Therefore, $c_2 - c_1 \geq e - c_1$.

If $c_2 - c_1 > e - c_1$, then the result is a case 4 call which resolves to $\langle (a, e - b) \rangle$. This means that there exists an r = e-b, such that the theorem is proven.

If $c_2 - c_1 = e - c_1$, the result resolves into the case 3 call

$$(a, (c_2 - c_1) - (b - c_1)) \overset{\text{sigcons}}{;} slice(\xi_x)_{[0,(e-c1)-(c2-c1)]}$$

After some simplification we get,

$$(a, (c_2 - b)) \overset{\text{sigcons}}{;} slice(\xi_x)_{[0,e-c2]}$$

But in this special case, $c_2 = e$. So this reduces to,

$$(a, e-b) \overset{\text{sigcons}}{;} slice(\xi_x)_{[0,0]}$$

which by the slice's case 1 definition grants us

$$\langle (a, e-b) \rangle$$

as required.

□

Theorem 63. *If ψ_1 and ψ_2 under interpretation semantics are sound with respect to their signal list semantics, then $\psi_1 \wedge \psi_2$ is also sound. Let i be an interpretation. Then,*

Assuming

$$i \vDash^I_{[b,e]} \psi_1 \Rightarrow c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi_1$$

and

$$i \vDash^I_{[b,e]} \psi_2 \Rightarrow c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi_2$$

we can conclude that

$$i \vDash^I_{[b,e]} \psi_1 \wedge \psi_2 \Rightarrow c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi_1 \wedge \psi_2$$

Proof.

$$\begin{array}{ll} & i \vDash^I_{[b,e]} \psi_1 \wedge \psi_2 \\ \iff & \{ \text{ Definition } \vDash^I_{[b,e]} \psi_1 \wedge \psi_2 \ \} \\ & i \vDash^I_{[b,e]} \psi_1 \wedge i \vDash^I_{[b,e]} \psi_2 \\ \implies & \{ \text{ Applying the implication in the assumptions } \} \\ & c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi_1 \wedge c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi_2 \\ \iff & \{ \text{ Expanding } \vDash^{SIG}_{[b,e]} \ \} \\ & slice(c2s(i2c(i)))_{[b,e]} \in \psi_1 \wedge slice(c2s(i2c(i)))_{[b,e]} \in \psi_2 \\ \iff & \{ \text{ Definition } \cap \ \} \\ & slice(c2s(i2c(i)))_{[b,e]} \in \psi_1 \cap \psi_2 \\ \iff & \{ \text{ Definition } \vDash^{SIG}_{[b,e]} \psi_1 \wedge \psi_2 \ \} \\ & c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi_1 \wedge \psi_2 \end{array}$$

□

Theorem 64. *If ψ_1 and ψ_2 under interpretation semantics are sound with respect to their signal list semantics, then $\psi_1 \vee \psi_2$ is also sound. Let i be an interpretation. Then,*

Assuming

$$i \vDash^I_{[b,e]} \psi_1 \Rightarrow c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi_1$$

and

$$i \vDash^I_{[b,e]} \psi_2 \Rightarrow c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi_2$$

we can conclude that

$$i \vDash^I_{[b,e]} \psi_1 \vee \psi_2 \Rightarrow c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi_1 \vee \psi_2$$

Proof.

$$
\begin{array}{ll}
 & i \vDash^{I}_{[b,e]} \psi_1 \vee \psi_2 \\
\iff & \{ \text{ Definition } \vDash^{I}_{[b,e]} \psi_1 \vee \psi_2 \ \} \\
 & i \vDash^{I}_{[b,e]} \psi_1 \vee i \vDash^{I}_{[b,e]} \psi_2 \\
 & \{ \vee \text{ Elimination } \} \\
 & \{ \textit{ Case 1 } \} \\
 & i \vDash^{I}_{[b,e]} \psi_1 \\
\implies & \{ \text{ Applying the implication in the assumptions } \} \\
 & c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi_1 \\
\implies & \{ \vee \text{ introduction } \} \\
 & c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi_1 \vee c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi_2 \\
 & \{ \textit{ Case 2 } \} \\
 & i \vDash^{I}_{[b,e]} \psi_2 \\
\implies & \{ \text{ Applying the implication in the assumptions } \} \\
 & c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi_2 \\
\implies & \{ \vee \text{ introduction } \} \\
 & c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi_1 \vee c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi_2 \\
\implies & \{ \text{ Result of } \vee \text{ Elimination } \} \\
 & c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi_1 \vee c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi_2 \\
\iff & \{ \text{ Definition } \vDash^{SIG}_{[b,e]} \ \} \\
 & slice(c2s(i2c(i)))_{[b,e]} \in \psi_1 \vee slice(c2s(i2c(i)))_{[b,e]} \in \psi_2 \\
\iff & \{ \text{ Definition } \cup \ \} \\
 & slice(c2s(i2c(i)))_{[b,e]} \in \psi_1 \cup \psi_2 \\
\iff & \{ \text{ Definition } \vDash^{SIG}_{[b,e]} \psi_1 \vee \psi_2 \ \} \\
 & c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi_1 \vee \psi_2
\end{array}
$$

□

Theorem 65. *If ψ_1 and ψ_2 under interpretation semantics are sound with respect to their signal list semantics, then $\psi_1 \circ \psi_2$ is also sound. Let i be an interpretation. Then,*

Assuming

$$i \vDash^{I}_{[b,e]} \psi_1 \Rightarrow c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi_1$$

and

$$i \vDash^{I}_{[b,e]} \psi_2 \Rightarrow c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi_2$$

we can conclude that

$$i \vDash^{I}_{[b,e]} \psi_1 \circ \psi_2 \Rightarrow c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi_1 \circ \psi_2$$

Proof.

$$
\begin{array}{ll}
 & i \vDash^{I}_{[b,e]} \psi_1 \circ \psi_2 \\
\iff & \{ \text{ Definition } \vDash^{I}_{[b,e]} \psi_1 \circ \psi_2 \ \} \\
 & \exists m : [b,e].i \vDash^{I}_{[b,m]} \psi_1 \wedge i \vDash^{I}_{[m,e]} \psi_2 \\
\implies & \{ \text{ Applying the implication in the assumptions } \} \\
 & \exists m : [b,e].c2s(i2c(i)) \vDash^{SIG}_{[b,m]} \psi_1 \wedge c2s(i2c(i)) \vDash^{SIG}_{[m,e]} \psi_2 \\
\iff & \{ \text{ Expanding } \vDash^{SIG}_{[b,m]} \ \} \\
 & \exists m : [b,e].slice(c2s(i2c(i)))_{[b,m]} \in \psi_1 \wedge slice(c2s(i2c(i)))_{[m,e]} \in \psi_2 \\
\iff & \{ \text{ Satisfies signal list semantics } \circ\text{'s Set Comprehension } \} \\
 & slice(c2s(i2c(i)))_{[b,m]} \overset{\text{sig glue}}{+\!\!+} slice(c2s(i2c(i)))_{[m,e]} \in \psi_1 \circ \psi_2 \\
\iff & \{ \text{ Lemma 55: Slice over [b,e] can be split into 2 parts } \} \\
 & slice(c2s(i2c(i)))_{[b,e]} \in \psi_1 \circ \psi_2 \\
\iff & \{ \text{ Definition } \vDash^{SIG}_{[b,e]} \psi_1 \circ \psi_2 \ \} \\
 & c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi_1 \circ \psi_2
\end{array}
$$

□

Lemma 66. *If ψ under interpretation semantics is sound with respect to its signal list semantics, then ψ^n is also sound. Let i be an interpretation. Then,*

Assuming

$$i \vDash^{I}_{[b,e]} \psi \Rightarrow c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi$$

we can conclude that

$$i \vDash^{I}_{[b,e]} \psi^n \Rightarrow c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi^n$$

Proof. By Induction on n.

Base Case, n = 0: Let us first start from the consequent and reduce it to a simpler form.

$$
\begin{array}{ll}
 & c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi^0 \\
\iff & \{ \text{ Definition } \vDash^{SIG}_{[b,e]} \psi^0 \ \} \\
 & c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \varepsilon \\
\iff & \{ \text{ Definition } \vDash^{SIG}_{[b,e]} \ \} \\
 & slice(c2s(i2c(i)))_{[b,e]} \in \varepsilon \\
\iff & \{ \text{ Definition } |[\varepsilon]| \ \} \\
 & slice(c2s(i2c(i)))_{[b,e]} \in \{\langle\rangle\} \\
\iff & \{ \text{ Definition } \in \text{ singleton set } \} \\
 & slice(c2s(i2c(i)))_{[b,e]} = \langle\rangle
\end{array}
$$

So, one is required to arrive at the last line, starting from the antecedent. Let us now proceed from the antecedent.

$$
\begin{array}{ll}
 & i \vDash^{I}_{[b,e]} \psi^0 \\
\iff & \{ \text{ Definition } \vDash^{I}_{[b,e]} \psi^0 \ \} \\
 & i \vDash^{I}_{[b,e]} \varepsilon \\
\iff & \{ \text{ Definition } \vDash^{I}_{[b,e]} \varepsilon \} \\
 & b = e
\end{array}
$$

So, far we know that $b = e$, and we want to prove that the expression $slice(c2s(i2c(i)))_{[b,e]}$ is in fact, the empty list. This is in fact straight forward, if we combine the latter two expressions. In fact, we know that since b = e, in this case,

$$slice(c2s(i2c(i)))_{[b,e]} = slice(c2s(i2c(i)))_{[e,e]}$$

By the definition of slice on point intervals, we know that this expression is in fact the empty list, which proves the base case.

Inductive Case: We take as our inductive hypothesis

$$i \vDash^{I}_{[b,e]} \psi^k \Rightarrow c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi^k$$

and prove the case

$$i \vDash^{I}_{[b,e]} \psi^{k+1} \Rightarrow c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi^{k+1}$$

$$\begin{array}{ll}
 & i \vDash^{I}_{[b,e]} \psi^{k+1} \\
\iff & \{ \text{ Definition } \psi^n \ \} \\
 & \exists m : [b,e].i \vDash^{I}_{[b,m]} \psi^k \wedge i \vDash^{I}_{[m,e]} \psi \\
\implies & \{ \text{ Applying the inductive hypothesis, and the theorem assumption } \} \\
 & \exists m : [b,e].c2s(i2c(i)) \vDash^{SIG}_{[b,m]} \psi^k \wedge c2s(i2c(i)) \vDash^{SIG}_{[m,e]} \psi \\
\iff & \{ \text{ Expanding } \vDash^{SIG}_{[b,e]} \ \} \\
 & \exists m : [b,e].slice(c2s(i2c(i)))_{[b,m]} \in \psi^k \wedge slice(c2s(i2c(i)))_{[m,e]} \in \psi \\
\iff & \{ \text{ Definition signal list semantics } \circ\text{'s Set Comprehension } \} \\
 & \exists m : [b,e].slice(c2s(i2c(i)))_{[b,m]} \overset{\text{sig glue}}{+\!\!+} slice(c2s(i2c(i)))_{[m,e]} \in \psi^k \circ \psi \\
\iff & \{ \text{ Lemma 55: Slice over [b,e] can be split into 2 parts } \} \\
 & \exists m : [b,e].slice(c2s(i2c(i)))_{[b,e]} \in \psi^k \circ \psi \\
\iff & \{ \text{ Definition } \psi^n \ \} \\
 & \exists m : [b,e].slice(c2s(i2c(i)))_{[b,e]} \in \psi^{k+1} \\
\iff & \{ \text{ Definition } \vDash^{SIG}_{[b,e]} \ \} \\
 & c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi^{k+1}
\end{array}$$

□

Theorem 67. *If ψ under interpretation semantics is sound with respect to the signal list semantics, then ψ^* is also sound. Let i be an interpretation. Then,*

Assuming

$$i \vDash^{I}_{[b,e]} \psi \Rightarrow c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi$$

we can conclude that

$$i \vDash^{I}_{[b,e]} \psi^* \Rightarrow c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi^*$$

Proof.

$$
\begin{array}{ll}
 & i \vDash^{I}_{[b,e]} \psi^{*} \\
\iff & \{ \text{ Definition } \psi^{*} \ \} \\
 & \exists n : \mathbb{N}. i \vDash^{I}_{[b,e]} \psi^{n} \\
\implies & \{ \text{ Lemma 66 } \} \\
 & \exists n : \mathbb{N}. c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi^{n} \\
\iff & \{ \text{ Definition } \psi^{*} \ \} \\
 & c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi^{*}
\end{array}
$$

□

Theorem 68. *If ψ under interpretation semantics is sound with respect to the signal list semantics, then for any interval [c,d], $\langle\psi\rangle_{[c,d]}$ is also sound. Let i be an interpretation. Then,*

Assuming

$$i \vDash^{I}_{[b,e]} \psi \Rightarrow c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi$$

we can conclude that

$$i \vDash^{I}_{[b,e]} \langle\psi\rangle_{[c,d]} \Rightarrow c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \langle\psi\rangle_{[c,d]}$$

Proof.

$$
\begin{array}{ll}
 & i \vDash^{I}_{[b,e]} \langle\psi\rangle_{[c,d]} \\
\iff & \{ \text{ Definition } \vDash^{I}_{[b,e]} \langle\psi\rangle_{[c,d]} \ \} \\
 & i \vDash^{I}_{[b,e]} \psi \wedge (c \leq e - b \leq d) \\
\implies & \{ \text{ Applying the implication in the assumptions } \} \\
 & c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi \wedge (c \leq e - b \leq d) \\
\iff & \{ \text{ Definition } \vDash^{SIG}_{[b,e]} \ \} \\
 & slice(c2s(i2c(i)))_{[b,e]} \in |[\psi]| \wedge (c \leq e - b \leq d) \\
\iff & \{ \text{ Theorem 54, } e - b = |slice(c2s(i2c(i)))_{[b,e]}| \ \} \\
 & slice(c2s(i2c(i)))_{[b,e]} \in |[\psi]| \wedge (c \leq slice(c2s(i2c(i)))_{[b,e]} \leq d) \\
\iff & \{ \text{ Constraint satisfies set comprehension } \} \\
 & slice(c2s(i2c(i)))_{[b,e]} \in |[\psi]| \wedge slice(c2s(i2c(i)))_{[b,e]} \in \{\xi : SIG \mid c \leq |\xi| \leq d\} \\
\iff & \{ \text{ Definition } \cap \ \} \\
 & slice(c2s(i2c(i)))_{[b,e]} \in |[\psi]| \cap \{\xi : Signal \mid c \leq |\xi| \leq d\} \\
\iff & \{ \text{ Definition } \in [c,d] \ \ \} \\
 & slice(c2s(i2c(i)))_{[b,e]} \in |[\psi]| \cap \{\xi : Signal \mid |\xi| \in [c,d]\} \\
\iff & \{ \text{ Definition } \langle\psi\rangle_{[c,d]} \ \} \\
 & slice(c2s(i2c(i)))_{[b,e]} \in \langle\psi\rangle_{[c,d]} \\
\iff & \{ \text{ Definition } \vDash^{SIG}_{[b,e]} \langle\psi\rangle_{[c,d]} \ \} \\
 & c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \langle\psi\rangle_{[c,d]}
\end{array}
$$

□

Theorem 69. *ε's signal list semantics are sound with respect to its interpretation semantics. Let ξ be a signal list. Then,*

$$\xi \vDash^{SIG}_{[b,e]} \varepsilon \Rightarrow c2i(s2c(\xi)) \vDash^{I}_{[b,e]} \varepsilon$$

Proof.

$$\begin{array}{ll} & \xi \vDash^{SIG}_{[b,e]} \varepsilon \\ \iff & \{ \text{ Definition of } \vDash^{SIG}_{[b,e]} \varepsilon \ \} \\ & slice(\xi)_{[b,e]} \in \{\langle\rangle\} \\ \iff & \{ \text{ Definition } \in \text{ of singleton set } \} \\ & slice(\xi)_{[b,e]} = \langle\rangle \\ \iff & \{ \text{ Definition of slice over point interval } \} \\ & b = e \\ \iff & \{ \text{ Definition of } \vDash^{I}_{[b,e]} \varepsilon \ \} \\ & c2i(s2c(\xi)) \vDash^{I}_{[b,e]} \varepsilon \end{array}$$

□

Theorem 70. *a's signal list semantics are sound with respect to its interpretation semantics. Let ξ be a signal list. Then,*

$$\xi \vDash^{SIG}_{[b,e]} a \Rightarrow c2i(s2c(\xi)) \vDash^{I}_{[b,e]} a$$

Proof. Let us first start by simplifying the consequent, so that it becomes clearer what one has to prove.

$$\begin{array}{ll} & c2i(s2c(\xi)) \vDash^{I}_{[b,e]} a \\ \iff & \{ \text{ Definition of } \vDash^{I}_{[b,e]} a \ \} \\ & \forall t : \mathbb{T}.\ b \leq t < e \Rightarrow c2i(s2c(\xi))(a,t) = true \end{array}$$

From this one can see that we have to prove that between the times b and e, the signal interpretation resulting from the signal list attains the value of true.

What about the antecedent?

$$\begin{array}{ll} & \xi \vDash^{Sig}_{[b,e]} a \\ \iff & \{ \text{ Definition of } \vDash^{Sig}_{[b,e]} a \ \} \\ & slice(\xi)_{[b,e]} \in \{a^r | r \in \mathbb{R}\} \\ \iff & \{ \text{ Definition of } \in \ \} \\ & \exists\ r : \mathbb{R}.\ slice(\xi)_{[b,e]} = \langle(a,r)\rangle \end{array}$$

The antecedent is telling us that the value of slice, when executed on ξ is a list of one tuple, with symbol a and some length r. In fact, we can improve on the existential statement. By the Lemma which tells us the length of the output of slice, we know that in fact

$$r = e - b$$

We shall use this fact later on.

Our strategy will be to determine what the value of $c2i(s2c(\xi))(a,t)$ is, given the information which we know. The first thing we note, is, that since $slice(\xi)_{[b,e]} = \langle(a,r)\rangle$, by Lemma 58, we know that

$$\xi = \xi_1 \overset{\text{sig glue}}{+\!\!+} \langle(a,r)\rangle \overset{\text{sig glue}}{+\!\!+} \xi_2$$

with ξ_1 or ξ_2 potentially empty. More over, due to Lemma 59 we also know that $|\xi_1| = b$. Now, what happens if we apply $s2c$ to this list? First of all, we should get rid of the gluing concatenation. Let us start by reducing the first gluing operator, i.e. let us reduce

$$\xi_1 \overset{\text{sig glue}}{+\!\!+} \langle(a,r)\rangle$$

- If ξ_1 is empty, this reduces to $\langle(a,r)\rangle$.
- If it is not empty, we can say that $\xi_1 = \xi_1' \overset{\text{sig}}{;} (\alpha,k)$. In this case, the expression reduces to $\xi_1' \overset{\text{sig}}{+\!\!+} \mu(\alpha,k)(a,r) \overset{\text{sig}}{+\!\!+} \langle\rangle$. Now, depending on the value of α, μ can have two outcomes. The first is that a merge occurs, meaning that we end up with $\xi_1' \overset{\text{sig}}{+\!\!+} \langle(a,r+k)\rangle$. The second, is that no merge occurs, and we end up with $\xi \overset{\text{sig}}{+\!\!+} \langle(a,r)\rangle$.

Thus, we can generalise and say that

$$\xi_1 \overset{\text{sig glue}}{+\!\!+} \langle(a,r)\rangle = \varsigma_1 \overset{\text{sig}}{;} (a,r')$$

where ς_1 is a signal list, potentially empty. We note that by Corollary 49 r' must be either greater or equal to r. We also note that $|\varsigma_1| = b - (r' - r)$. How did we arrive at this number? b is the length of ξ_1. Now, if no merge occurred, then ς_1 is equal to ξ_1. In this case, r' and r are equal, as required. If a merge occurred, then, we have 'stolen' one of the tuples of ξ_1 to form a tuple of length r'. Therefore. in this case, $\varsigma_1 = \xi_1'$, whose length is equal to b minus the amount stolen from it to form r', which is $r' - r$.

Let us now simplify the statement

$$\varsigma_1 \overset{\text{sig}}{;} (a,r') \overset{\text{sig glue}}{+\!\!+} \xi_2$$

There are two possibilities. ξ_2 can be empty or non-empty.

- If ξ_2 is empty the statement reduces to $\varsigma_1 \overset{\text{sig}}{;} (a,r')$, which we are able to change into $\varsigma_1 \overset{\text{sig}}{+\!\!+} \langle(a,r')\rangle$.
- If it is not empty, we can say that ξ_2 has the form $\xi_2 = \langle(\alpha_2,k_2)\rangle \overset{\text{sig}}{+\!\!+} \xi_2'$. Therefore in this case, we would have the expression

$$\varsigma_1 \overset{\text{sig}}{;} (a,r') \overset{\text{sig glue}}{+\!\!+} \langle(\alpha_2,k_2)\rangle \overset{\text{sig}}{+\!\!+} \xi_2'$$

which simplifies to

$$\varsigma_1 \overset{\text{sig}}{+\!\!+} \mu(a, r')(\alpha_2, k_2) \overset{\text{sig}}{+\!\!+} \xi_2'$$

Now, μ can either merge the tuples, or it will not, depending on the value of α_2. If the tuples are merged, we end up with the expression

$$\varsigma_1 \overset{\text{sig}}{+\!\!+} \langle (a, r' + k_2) \rangle \overset{\text{sig}}{+\!\!+} \xi_2'$$

Otherwise, we end up with

$$\varsigma_1 \overset{\text{sig}}{+\!\!+} \langle (a, r'), (\alpha, k_2) \rangle \overset{\text{sig}}{+\!\!+} \xi_2'$$

which we can reduce to

$$\varsigma_1 \overset{\text{sig}}{+\!\!+} \langle (a, r') \rangle \overset{\text{sig}}{+\!\!+} \langle (\alpha, k_2) \rangle \overset{\text{sig}}{+\!\!+} \xi_2'$$

Therefore, we can generalise from these results and say that

$$\varsigma_1 \overset{\text{sig}}{;} (a, r') \overset{\text{sig glue}}{+\!\!+} \xi_2$$

is equivalent to

$$\varsigma_1 \overset{\text{sig}}{+\!\!+} \langle (a, r'') \rangle \overset{\text{sig}}{+\!\!+} \varsigma_2$$

where ς_2 is a signal list variable, and where we can observe that r" is either equal or greater than r', and therefore also greater than or equal to r.

Also, since $\langle (a, r'') \rangle$ is a singleton list, we can replace $\overset{\text{sig}}{+\!\!+}$ with $\overset{\text{sig}}{;}$, obtaining

$$\varsigma_1 \overset{\text{sig}}{;} (a, r'') \overset{\text{sig}}{+\!\!+} \varsigma_2$$

Now that we have eliminated the gluing concatenation, we can finally apply s2c, getting

$$s2c(\varsigma_1 \overset{\text{sig}}{;} (a, r'') \overset{\text{sig}}{+\!\!+} \varsigma_2)$$

We know that s2c can be written in terms of an α fold, where α is the domain SIG.

$$fold_{SIG} \overset{\text{cp}}{+\!\!+} (\varsigma_1 \overset{\text{sig}}{;} (a, r'') \overset{\text{sig}}{+\!\!+} \varsigma_2) \langle (0, \perp) \rangle f$$

Since we have already proven that $\overset{\text{cp}}{+\!\!+}$ is associative, we can use Theorem 19, which allows to write

$$(fold_{SIG} \overset{\text{cp}}{+\!\!+} (\varsigma_1 \overset{\text{sig}}{;} (a, r'')) \langle (0, \perp) \rangle f) \overset{\text{cp}}{+\!\!+} (fold_{SIG} \overset{\text{cp}}{+\!\!+} (\varsigma_2) \langle (0, \perp) \rangle f)$$

If we turn the fold definition into the standard definition of $s2c$, we obtain

$$s2c(\varsigma_1 \overset{\text{sig}}{;} (a, r'')) \overset{\text{cp}}{+\!\!+} s2c(\varsigma_2)$$

Now, since s2c returns a critical point list, we can simplify this to

$$s2c(\varsigma_1 \overset{\text{sig}}{;} (a, r'')) \overset{\text{cp}}{+\!\!+} ws$$

where ws is a critical point list. Also, if we apply the remaining $s2c()$ to its parameter we get

$$s2c(\varsigma_1) \overset{\text{cp}}{;} \langle (0,a),(r'',\bot)\rangle \overset{\text{cp}}{+\!\!+} ws$$

Recall that we said that $|\varsigma_1| = b-(r'-r)$ Now since, s2c preserves length, then $|s2c(\varsigma_1)|$ is also equal to $b-(r'-r)$. Also, since $s2c$ returns a critical point list, we can therefore write:

$$xs : (b-(r'-r),\bot) \overset{\text{cp}}{;} \langle (0,a)(r'',\bot)\rangle \overset{\text{cp}}{+\!\!+} ws$$

If we use the definition for $\overset{\text{cp}}{;}$, we get

$$xs : (b-(r'-r),a) : (r''+(b-(r'-r)),\bot) \overset{\text{cp}}{+\!\!+} ws$$

The next step is to apply c2i to the expression above.

$$c2i(xs : (b-(r'-r),a) : (r''+(b-(r'-r)),\bot) \overset{\text{cp}}{+\!\!+} ws)$$

Given that we know a definition of c2i in terms of an α fold, we can write the above expression as

$$fold_{CPL} \overset{\text{I}}{+\!\!+} (xs : (b-(r'-r),a) : (r''+b-(r'-r),\bot) \overset{\text{cp}}{+\!\!+} ws)\ False\ f$$

Since $\overset{\text{I}}{+\!\!+}$ is associative, we use theorem 19 and s2c's fold definition to transform the expression into

$$c2i(xs : (b-(r'-r),a) : (r''+(b-(r'-r)),\bot)) \overset{\text{I}}{+\!\!+} c2i(ws)$$

Again, since $c2i()$ returns an interpretation, we can write,

$$c2i(xs : (b-(r'-r),a) : (r''+(b-(r'-r)),\bot)) \overset{\text{I}}{+\!\!+} wi$$

where wi is an interpretation. Now, by expanding $c2i$, we get

$$c2i(xs : (b-(r'-r),\bot)) \overset{\text{I}}{;} \begin{cases} j(a,t) = \text{true for } 0 \le t < r'' \\ j(a,t) = \text{false for } t \ge r'' \\ j(\beta,t) = \text{false for } t \ge 0 \end{cases} \overset{\text{I}}{+\!\!+} wi$$

We now apply the $\overset{\text{I}}{;}$ operator. Remember that $c2i$ preserves length, and returns an interpretation, which we shall call i. Also recall that $|xs : (b-(r'-r),\bot)| = b-(r'-r)$. Therefore, the length of the interpretation i is also $b-(r'-r)$. Thus, when we apply the $\overset{\text{I}}{;}$ operator we get

$$\begin{cases} i(a,t) = \text{ for } 0 \le t < b-(r'-r) \\ j(a,t-(b-(r'-r))) = \text{ for } t \ge b-(r'-r) \end{cases} \overset{\text{I}}{+\!\!+} wi$$

Since j has length r'', this means that for any point between b-(r'-r) and b-(r'-r)+r"(exclusive), j determines the value of the entire expression (since any future interpretations such as

wi cannot interfere in this range) Now, $b-(r'-r)$ is either equal or less than b. We show that b-(r'-r)+r" must be equal or greater than e.

$$\begin{array}{ll} & b-(r'-r)+r'' \\ = & \{ \text{ Term Rearrangement } \} \\ & r''-r'+r+b \\ = & \{ \text{ r" } \geq \text{ r } \} \\ & x+r+b \text{ for some non-negative x} \\ = & \{ \text{ r = e-b } \} \\ & x+e-b+b \text{ for non-negative x} \\ = & \{ \text{ Simplification } \} \\ & x+e \text{ for some non-negative x} \end{array}$$

which is either equal or greater than e.

Therefore we can be sure that in the region between b and e, j controls the value of the function. Now, if we go back a few steps and look at the definition of $i \overset{\text{I}}{;} j$, we see that at the point b, it resolves to $j(\alpha, r'-r)$, which is either 0 or less than r", meaning that the value of true is attained. At the value of e, it resolves to the value of j at e - (b-(r'-r)), which reduces to the value of r'. Therefore if the input value is less than e, then the value at j is less than r'. Therefore, for every point less than e, the value at j is less than r' and therefore less than r", at which points j yields the value of true. Since, j is true at the desired points of interest, and the interpretations *wi* cannot affect values occurring earlier than e, the theorem is proven.

□

Theorem 71. *If ψ_1 and ψ_2 under signal list semantics are sound with respect to their interpretation semantics, then $\psi_1 \wedge \psi_2$ is also sound. Let ξ be a signal list. Then,*

Assuming

$$\xi \models^{SIG}_{[b,e]} \psi_1 \Rightarrow c2i(s2c(\xi)) \models^{I}_{[b,e]} \psi_1$$

and

$$\xi \models^{SIG}_{[b,e]} \psi_2 \Rightarrow c2i(s2c(\xi)) \models^{I}_{[b,e]} \psi_2$$

we can conclude that

$$\xi \models^{SIG}_{[b,e]} \psi_1 \wedge \psi_2 \Rightarrow c2i(s2c(\xi)) \models^{I}_{[b,e]} \psi_1 \wedge \psi_2$$

Proof.

$$
\begin{array}{ll}
 & \xi \models^{SIG}_{[b,e]} \psi_1 \wedge \psi_2 \\
\iff & \{ \text{ Definition } \models^{SIG}_{[b,e]} \psi_1 \wedge \psi_2 \ \} \\
 & slice(\xi)_{[b,e]} \in \psi_1 \cap \psi_2 \\
\iff & \{ \text{ Definition } \cap \ \} \\
 & slice(\xi)_{[b,e]} \in \psi_1 \wedge slice(\xi)_{[b,e]} \in \psi_2 \\
\iff & \{ \text{ Definition } \models^{SIG}_{[b,e]} \ \} \\
 & \xi \models^{SIG}_{[b,e]} \psi_1 \wedge \xi \models^{SIG}_{[b,e]} \psi_2 \\
\implies & \{ \text{ Applying Structural Induction Hypothesis } \} \\
 & c2s(s2c(i)) \models^{I}_{[b,e]} \psi_1 \wedge c2s(s2c(i)) \models^{I}_{[b,e]} \psi_2 \\
\iff & \{ \text{ Definition } \models^{I}_{[b,e]} \psi_1 \wedge \psi_2 \ \} \\
 & c2s(s2c(i)) \models^{I}_{[b,e]} \psi_1 \wedge \psi_2
\end{array}
$$

□

Theorem 72. *If ψ_1 and ψ_2 under signal list semantics are sound with respect to their interpretation semantics, then $\psi_1 \vee \psi_2$ is also sound. Let ξ be a signal list. Then,*

Assuming

$$\xi \models^{SIG}_{[b,e]} \psi_1 \Rightarrow c2i(s2c(\xi)) \models^{I}_{[b,e]} \psi_1$$

and

$$\xi \models^{SIG}_{[b,e]} \psi_2 \Rightarrow c2i(s2c(\xi)) \models^{I}_{[b,e]} \psi_2$$

we can conclude that

$$\xi \models^{SIG}_{[b,e]} \psi_1 \vee \psi_2 \Rightarrow c2i(s2c(\xi)) \models^{I}_{[b,e]} \psi_1 \vee \psi_2$$

Proof.

$$
\begin{array}{ll}
 & \xi \models^{SIG}_{[b,e]} \psi_1 \vee \psi_2 \\
\iff & \{ \text{ Definition } \models^{SIG}_{[b,e]} \psi_1 \vee \psi_2 \ \} \\
 & \xi \models^{SIG}_{[b,e]} \psi_1 \vee \xi \models^{SIG}_{[b,e]} \psi_2 \\
 & \{ \vee \textit{ Elimination } \} \\
 & \{ \textit{ Case 1 } \} \\
 & \xi \models^{SIG}_{[b,e]} \psi_1 \\
\implies & \{ \text{ Applying Structural Induction Hypothesis } \} \\
 & c2i(s2c(\xi)) \models^{I}_{[b,e]} \psi_1 \\
\implies & \{ \vee \text{ introduction } \} \\
 & c2i(s2c(\xi)) \models^{I}_{[b,e]} \psi_1 \vee c2i(s2c(\xi)) \models^{I}_{[b,e]} \psi_2 \\
 & \{ \textit{ Case 2 } \} \\
 & \xi \models^{SIG}_{[b,e]} \psi_2 \\
\implies & \{ \text{ Applying Structural Induction Hypothesis } \} \\
 & c2i(s2c(\xi)) \models^{I}_{[b,e]} \psi_2 \\
\implies & \{ \vee \text{ introduction } \} \\
 & c2i(s2c(\xi)) \models^{I}_{[b,e]} \psi_1 \vee c2i(s2c(\xi)) \models^{I}_{[b,e]} \psi_2 \\
\implies & \{ \text{ Result of } \vee \text{ elimination } \} \\
 & c2i(s2c(\xi)) \models^{I}_{[b,e]} \psi_1 \vee c2i(s2c(\xi)) \models^{I}_{[b,e]} \psi_2 \\
\iff & \{ \text{ Definition } \models^{SIG}_{[b,e]} \psi_1 \vee \psi_2 \ \} \\
 & c2i(s2c(\xi)) \models^{I}_{[b,e]} \psi_1 \vee \psi_2
\end{array}
$$

□

Theorem 73. *If ψ_1 and ψ_2 under signal list semantics are sound with respect to their interpretation semantics, then $\psi_1 \circ \psi_2$ is also sound. Let ξ be a signal list. Then,*

Assuming

$$\xi \models^{SIG}_{[b,e]} \psi_1 \Rightarrow c2i(s2c(\xi)) \models^{I}_{[b,e]} \psi_1$$

and

$$\xi \models^{SIG}_{[b,e]} \psi_2 \Rightarrow c2i(s2c(\xi)) \models^{I}_{[b,e]} \psi_2$$

we can conclude that

$$\xi \models^{SIG}_{[b,e]} \psi_1 \circ \psi_2 \Rightarrow c2i(s2c(\xi)) \models^{I}_{[b,e]} \psi_1 \circ \psi_2$$

Proof.

$$
\begin{array}{ll}
 & \xi \vDash^{SIG}_{[b,e]} \psi_1 \circ \psi_2 \\
\iff & \{ \text{ Definition } \vDash^{SIG}_{[b,e]} \psi_1 \circ \psi_2 \ \} \\
 & slice(\xi)_{[b,e]} \in \psi_1 \circ \psi_2 \\
\iff & \{ \text{ Lemma 55: Slice over [b,e] can be split into 2 parts } \} \\
 & slice(\xi)_{[b,m]} \overset{\text{sig glue}}{+\!\!+} slice(\xi)_{[m,e]} \in \psi_1 \circ \psi_2 \\
\iff & \{ \text{ Satisfies signal list semantics } \circ\text{'s set comprehension } \} \\
 & \exists m : [b,e].\ slice(\xi)_{[b,m]} \in \psi_1 \wedge slice(\xi)_{[m,e]} \in \psi_2 \\
\iff & \{ \text{ Definition of } \vDash^{SIG}_{[b,m]} \text{ and } \vDash^{SIG}_{[m,e]} \ \} \\
 & \exists m : [b,e].\ \xi \vDash^{SIG}_{[b,m]} \psi_1 \wedge \xi \vDash^{SIG}_{[m,e]} \psi_2 \\
\implies & \{ \text{ Applying Structural Induction Hypothesis } \} \\
 & \exists m : [b,e].\ c2i(s2c(\xi)) \vDash^{I}_{[b,m]} \psi_1 \wedge c2i(s2c(\xi)) \vDash^{I}_{[m,e]} \psi_2 \\
\iff & \{ \text{ Definition } \vDash^{I}_{[b,e]} \psi_1 \circ \psi_2 \ \} \\
 & i \vDash^{I}_{[b,e]} \psi_1 \circ \psi_2
\end{array}
$$

□

Lemma 74. *If ψ under signal list semantics is sound with respect to its interpretation semantics, then ψ^n is also sound. Let ξ be a signal list. Then,*

Assuming

$$\xi \vDash^{SIG}_{[b,e]} \psi \Rightarrow c2i(s2c(\xi)) \vDash^{I}_{[b,e]} \psi$$

we can conclude that

$$\xi \vDash^{SIG}_{[b,e]} \psi^n \Rightarrow c2i(s2c(\xi)) \vDash^{I}_{[b,e]} \psi^n$$

Proof. By Induction on n.

Base Case, n = 0:

$$
\begin{array}{ll}
 & \xi \vDash^{SIG}_{[b,e]} \psi^0 \\
\iff & \{ \text{ Definition } \vDash^{SIG}_{[b,e]} \psi^0 \ \} \\
 & \xi \vDash^{SIG}_{[b,e]} \varepsilon \\
\iff & \{ \text{ Definition } \vDash^{SIG}_{[b,e]} \ \} \\
 & slice(\xi)_{[b,e]} \in \varepsilon \\
\iff & \{ \text{ Definition } |[\varepsilon]| \ \} \\
 & slice(\xi)_{[b,e]} \in \{\langle\rangle\} \\
\iff & \{ \text{ Definition } \in \text{ singleton set } \} \\
 & slice(\xi)_{[b,e]} = \langle\rangle \\
\iff & \{ \text{ Definition of } slice(\xi)_{[b,e]} \ \} \\
 & slice(\xi)_{[b,e]} = slice(\xi)_{[e,e]} \\
\iff & \{ \text{ Consequence of Equality } \} \\
 & b = e \\
\iff & \{ \text{ Definition } \vDash^{I}_{[b,e]} \varepsilon \ \} \\
 & c2i(s2c(\xi)) \vDash^{I}_{[b,e]} \varepsilon \\
\iff & \{ \text{ Definition } \vDash^{I}_{[b,e]} \psi^0 \ \} \\
 & c2i(s2c(\xi)) \vDash^{I}_{[b,e]} \psi^0
\end{array}
$$

Inductive Case: We take as our inductive hypothesis

$$\xi \vDash^{SIG}_{[b,e]} \psi^k \Rightarrow c2i(s2c(\xi)) \vDash^{I}_{[b,e]} \psi^k$$

and prove the case

$$\xi \vDash^{SIG}_{[b,e]} \psi^{k+1} \Rightarrow c2i(s2c(\xi)) \vDash^{SIG}_{[b,e]} \psi^{k+1}$$

$$\begin{array}{ll}
 & \xi \vDash^{SIG}_{[b,e]} \psi^{k+1} \\
\iff & \{ \text{ Definition } \vDash^{SIG}_{[b,e]} \} \\
 & slice(\xi)_{[b,e]} \in \psi^{k+1} \\
\iff & \{ \text{ Definition } \psi^n \} \\
 & slice(\xi)_{[b,e]} \in \psi^k \circ \psi \\
\iff & \{ \text{ Lemma 55: Slice over [b,e] can be split into 2 parts } \} \\
 & \exists m : [b,e].\ slice(\xi)_{[b,m]} \overset{\text{sig glue}}{+\!\!+} slice(\xi)_{[m,e]} \in \psi^k \circ \psi \\
\iff & \{ \text{ Definition signal list semantics } \circ\text{'s set comprehension } \} \\
 & \exists m : [b,e].\ slice(\xi)_{[b,m]} \in \psi^k \wedge slice(\xi)_{[m,e]} \in \psi \\
\iff & \{ \text{ Definition of } \vDash^{SIG}_{[b,e]} \} \\
 & \exists m : [b,e].\ \xi \vDash^{SIG}_{[b,m]} \psi^k \wedge \xi \vDash^{SIG}_{[m,e]} \psi \\
\implies & \{ \text{ Applying Structural Induction Hypothesis } \} \\
 & \exists m : [b,e].\ c2i(s2c(\xi)) \vDash^{I}_{[b,m]} \psi^k \wedge c2i(s2c(\xi)) \vDash^{I}_{[m,e]} \psi \\
\iff & \{ \text{ Signal Interpretation Definition of } \circ \} \\
 & \exists m : [b,e].\ c2i(s2c(\xi)) \vDash^{I}_{[b,e]} \psi^k \circ \psi \\
\iff & \{ \text{ Definition } \psi^n \} \\
 & c2i(s2c(\xi)) \vDash^{I}_{[b,e]} \psi^{k+1}
\end{array}$$

□

Theorem 75. *If ψ under signal list semantics is sound with respect to its interpretation semantics, then ψ^* is also sound. Let ξ be a signal list. Then,*

Assuming

$$\xi \vDash^{SIG}_{[b,e]} \psi \Rightarrow c2i(s2c(\xi)) \vDash^{I}_{[b,e]} \psi$$

we can conclude that

$$\xi \vDash^{SIG}_{[b,e]} \psi^* \Rightarrow c2i(s2c(\xi)) \vDash^{I}_{[b,e]} \psi^*$$

Proof.

$$\begin{array}{ll}
 & \xi \vDash^{SIG}_{[b,e]} \psi^* \\
\iff & \{ \text{ Definition } \psi^* \} \\
 & slice(\xi)_{[b,e]} \in \bigcup_{i=0}^{\infty} |[\psi^i]| \\
\iff & \{ \text{ Definition of } \in \} \\
 & \exists n : \mathbb{N}.\ slice(\xi)_{[b,e]} \in |[\psi^n]| \\
\iff & \{ \text{ Definition of } \vDash^{SIG}_{[b,e]} \} \\
 & \exists n : \mathbb{N}.\ \xi \vDash^{SIG}_{[b,e]} \psi^n \\
\implies & \{ \text{ Lemma 74 } \} \\
 & \exists n : \mathbb{N}.\ c2i(s2c(\xi)) \vDash^{I}_{[b,e]} \psi^n \\
\iff & \{ \text{ Signal Interpretation Definition of } \psi^* \} \\
 & c2i(s2c(\xi)) \vDash^{I}_{[b,e]} \psi^*
\end{array}$$

□

Theorem 76. *If ψ under signal list semantics is sound with respect to its interpretation semantics, then $\langle\psi\rangle_{[c,d]}$ is also sound. Let ξ be a signal list. Then,*

Assuming

$$\xi \vDash^{SIG}_{[b,e]} \psi \Rightarrow c2i(s2c(\xi)) \vDash^{I}_{[b,e]} \psi$$

we can conclude that

$$\xi \vDash^{SIG}_{[b,e]} \langle\psi\rangle_{[c,d]} \Rightarrow c2i(s2c(\xi)) \vDash^{I}_{[b,e]} \langle\psi\rangle_{[c,d]}$$

Proof.

$$\begin{array}{ll}
 & \xi \vDash^{SIG}_{[b,e]} \langle\psi\rangle_{[c,d]} \\
\iff & \{ \text{ Definition } \vDash^{SIG}_{[b,e]} \langle\psi\rangle_{[c,d]} \ \} \\
 & slice(\xi)_{[b,e]} \in \langle\psi\rangle_{[c,d]} \\
\iff & \{ \text{ Definition } \langle\psi\rangle_{[c,d]} \ \} \\
 & slice(\xi)_{[b,e]} \in |[\psi]| \cap \{\xi : Signal \mid |\xi| \in [c,d]\} \\
\iff & \{ \text{ Definition } \in [c,d] \ \ \} \\
 & slice(\xi)_{[b,e]} \in |[\psi]| \cap \{\xi : Signal \mid c \leq |\xi| \leq d\} \\
\iff & \{ \text{ Definition } \cap \ \} \\
 & slice(\xi)_{[b,e]} \in |[\psi]| \wedge slice(\xi)_{[b,e]} \in \{\xi : Signal \mid c \leq |\xi| \leq d\} \\
\iff & \{ \text{ Constraint satisfies set comprehension } \} \\
 & slice(\xi)_{[b,e]} \in |[\psi]| \wedge (c \leq |slice(\xi)_{[b,e]}| \leq d) \\
\iff & \{ \text{ Theorem 54, } |slice(\xi)_{[b,e]}| = e - b \ \} \\
 & slice(\xi)_{[b,e]} \in |[\psi]| \wedge (c \leq e - b \leq d) \\
\iff & \{ \text{ Definition } \vDash^{SIG}_{[b,e]} \ \} \\
 & \xi \vDash^{SIG}_{[b,e]} \psi \wedge (c \leq e - b \leq d) \\
\implies & \{ \text{ Applying Structural Induction Hypothesis } \} \\
 & c2i(s2c(\xi)) \vDash^{I}_{[b,e]} \psi \wedge (c \leq e - b \leq d) \\
\iff & \{ \text{ Definition } \vDash^{I}_{[b,e]} \langle\psi\rangle_{[c,d]} \ \} \\
 & c2i(s2c(\xi)) \vDash^{I}_{[b,e]} \langle\psi\rangle_{[c,d]}
\end{array}$$

□

Theorem 77. *The interpretation semantics for TRE are complete with respect to the signal list semantics for TRE. Let i be an interpretation and ψ a TRE. Then the following holds.*

$$c2s(i2c(i)) \vDash^{SIG}_{[b,e]} \psi \Rightarrow i \vDash^{I}_{[b,e]} \psi$$

Proof.

$$
\begin{array}{ll}
 & \underbrace{c2s(i2c(i))}_{\text{signal}} \vDash^{SIG}_{[b,e]} \psi \\
\Longrightarrow & \{ \text{ Second Soundness Theorem, Section 3.5.2 } \} \\
 & c2i(s2c(c2s(i2c(i)))) \vDash^{I}_{[b,e]} \psi \\
\Longleftrightarrow & \{ \text{ s2c().c2s() = id, Theorem 44 } \} \\
 & c2i(id((i2c(i)))) \vDash^{I}_{[b,e]} \psi \\
\Longleftrightarrow & \{ \text{ id(x) = x } \} \\
 & c2i(i2c(i)) \vDash^{I}_{[b,e]} \psi \\
\Longleftrightarrow & \{ \text{ c2i().i2c() = id, Theorem 38 } \} \\
 & id(i) \vDash^{I}_{[b,e]} \psi \\
\Longleftrightarrow & \{ \text{ id(x) = x } \} \\
 & i \vDash^{I}_{[b,e]} \psi
\end{array}
$$

□

Theorem 78. *The signal list semantics for TRE are complete with respect to the interpretation semantics for TRE. Let ξ be an signal list and ψ a TRE. Then the following holds.*

$$c2i(s2c(\xi)) \vDash^{I}_{[b,e]} \psi \Rightarrow \xi \vDash^{SIG}_{[b,e]} \psi$$

Proof.

$$
\begin{array}{ll}
 & \underbrace{c2i(s2c(\xi))}_{\text{interpretation}} \vDash^{I}_{[b,e]} \psi \\
\Longrightarrow & \{ \text{ First Soundness Theorem, Section 3.5.3 } \} \\
 & c2s(i2c(c2i(s2c(\xi)))) \vDash^{SIG}_{[b,e]} \psi \\
\Longleftrightarrow & \{ \text{ i2c().c2i() = id, Theorem 37 } \} \\
 & c2s(id(s2c(\xi))) \vDash^{SIG}_{[b,e]} \psi \\
\Longleftrightarrow & \{ \text{ id(x) = x } \} \\
 & c2s(s2c(\xi)) \vDash^{SIG}_{[b,e]} \psi \\
\Longleftrightarrow & \{ \text{ c2s().s2c() = id, Theorem 45 } \} \\
 & id(i) \vDash^{SIG}_{[b,e]} \psi \\
\Longleftrightarrow & \{ \text{ id(i) = i } \} \\
 & i \vDash^{SIG}_{[b,e]} \psi
\end{array}
$$

□

Lemma 87. *Consider some particular time transform s. If for every time point t between b and e, a transformed interpretation is true for some symbol at the transformed point s(t), then it is also true for that symbol between s(b) and s(e). Let s be a time transform, and i and interpretation. Then,*

$$(\forall t : \mathbb{T}.\ b \leq t \leq e \Rightarrow i_s(\alpha, s(t))) \Rightarrow (\forall t' : \mathbb{T}.\ s(b) \leq t' \leq s(e) \Rightarrow i_s(\alpha, t'))$$

Proof. Recall that a time transform function will take the time domain and transform it in a continous fashion. If the time transform is $s()$, then it will map b to $s(b)$ and e to $s(e)$. Now, consider a time point t between b and e. Let us call the transformed version of this time point s(t), t'. By the monotonicity condition of a time transform, if $b \leq t \leq e$, then $s(b) \leq s(t) \leq s(e)$, or similarly, $s(b) \leq t' \leq s(e)$. Also, let $i_s(\alpha, s(t))$ be written as a boolean function of $s(t)$, $f(s(t))$. Now, $f(s(t))$ is true between b and e. For every t between b and e, $s(t)$ is equal to some t' between $s(b)$ and $s(e)$. Therefore, $f(s(t))$ can be written as a function of t', $f(t')$. If we expand the function of t', we get $i_s(\alpha, t')$. Thus, $\forall t' : \mathbb{T}.\ s(b) \leq t' \leq s(e) \Rightarrow i_s(\alpha, t')$ as required.

□

Theorem 88. *a is slowdown truth preserving. Let i be an interpretation, and s a time transform. Then,*

$$i \vDash_{[b,e]} a \Rightarrow i_s \vDash_{[s(b),s(e)]} a$$

Proof.

$$\begin{array}{ll} & i \vDash_{[b,e]} a \\ \iff & \{ \text{ Definition of } a \ \} \\ & \forall t : \mathbb{T}.\ b \leq t < e. \Rightarrow i(a, t) \\ \iff & \{ \text{ Definition of Time Transform: } \forall t : \mathbb{T}.\ i(\alpha, t) = i_s(\alpha, s(t)) \ \} \\ & \forall t : \mathbb{T}.\ b \leq t < e. \Rightarrow i_s(a, s(t)) \\ \implies & \{ \text{ Lemma 87: Moving function application to the range of input } \} \\ & \forall t : \mathbb{T}.\ s(b) \leq t' < s(e). \Rightarrow i_s(a, t') \\ \iff & \{ \text{ Definition of } a \ \} \\ & i_s \vDash_{[s(b),s(e)]} a \end{array}$$

□

Theorem 89. *If ψ_1 and ψ_2 are slowdown truth preserving, so is $\psi_1 \vee \psi_2$. Let s be a slowdown transform.*

Assuming

$$sdtp(\psi_1)$$

and

$$sdtp(\psi_2)$$

we can conclude that

$$i \vDash_{[b,e]} \psi_1 \vee \psi_2 \Rightarrow i_s \vDash_{[s(b),s(e)]} \psi_1 \vee \psi_2$$

Proof.

$$
\begin{array}{ll}
 & \{ \text{ Definition } sdtp(\psi_1) \} \\
 & i \vDash_{[b,e]} \psi_1 \Rightarrow i_s \vDash_{[s(b),s(e)]} \psi_1 \\
 & \{ \text{ Definition } sdtp(\psi_2) \} \\
 & i \vDash_{[b,e]} \psi_2 \Rightarrow i_s \vDash_{[s(b),s(e)]} \psi_2 \\
 & \{ \text{ Hypothesis } \} \\
 & i \vDash_{[b,e]} \psi_1 \vee \psi_2 \\
 & \{ \vee \textit{ elimination } \} \\
 & \{ \vee \textit{ Case 1 } \} \\
 & i \vDash_{[b,e]} \psi_1 \\
\Longrightarrow & \{ \text{ Applying } sdtp(\psi_1) \text{ implication } \} \\
 & i_s \vDash_{[s(b),s(e)]} \psi_1 \\
\Longrightarrow & \{ \vee \text{ introduction } \} \\
 & i_s \vDash_{[s(b),s(e)]} \psi_1 \vee i_s \vDash_{[s(b),s(e)]} \psi_2 \\
\Longleftrightarrow & \{ \text{ Definition } sdtp(\psi_1 \vee \psi_2) \} \\
 & i_s \vDash_{[s(b),s(e)]} \psi_1 \vee \psi_2 \\
 & \{ \vee \textit{ Case 2 } \} \\
 & i \vDash_{[b,e]} \psi_2 \\
\Longrightarrow & \{ \text{ Applying } sdtp(\psi_2) \text{ implication } \} \\
 & i_s \vDash_{[s(b),s(e)]} \psi_2 \\
\Longrightarrow & \{ \vee \text{ introduction } \} \\
 & i_s \vDash_{[s(b),s(e)]} \psi_1 \vee i_s \vDash_{[s(b),s(e)]} \psi_2 \\
\Longleftrightarrow & \{ \text{ Definition of } sdtp(\psi_1 \vee \psi_2) \} \\
 & i_s \vDash_{[s(b),s(e)]} \psi_1 \vee \psi_2 \\
\Longrightarrow & \{ \vee\text{-elimination conclusion } \} \\
 & i_s \vDash_{[s(b),s(e)]} \psi_1 \vee \psi_2
\end{array}
$$

□

Theorem 90. *If ψ_1 and ψ_2 are slowdown truth preserving, so is $\psi_1 \wedge \psi_2$. Let s be a slowdown transform.*

Assuming

$$sdtp(\psi_1)$$

and

$$sdtp(\psi_2)$$

we can conclude that

$$i \vDash_{[b,e]} \psi_1 \wedge \psi_2 \Rightarrow i_s \vDash_{[s(b),s(e)]} \psi_1 \wedge \psi_2$$

Proof.

$$
\begin{array}{ll}
 & \{ \text{ Definition } sdtp(\psi_1) \ \} \\
 & i \vDash_{[b,e]} \psi_1 \Rightarrow i_s \vDash_{[s(b),s(e)]} \psi_1 \\
 & \{ \text{ Definition } sdtp(\psi_2) \ \} \\
 & i \vDash_{[b,e]} \psi_2 \Rightarrow i_s \vDash_{[s(b),s(e)]} \psi_2 \\
 & \{ \text{ Hypothesis } \} \\
 & i \vDash_{[b,e]} \psi_1 \wedge \psi_2 \\
\Longrightarrow & \{ \wedge \text{ elimination } \} \\
 & i \vDash_{[b,e]} \psi_1 \\
\Longrightarrow & \{ \text{ Applying implication, } sdtp(\psi_1) \ \} \\
 & i_s \vDash_{[s(b),s(e)]} \psi_1 \\
\Longrightarrow & \{ \wedge \text{ elimination } \} \\
 & i \vDash_{[b,e]} \psi_2 \\
\Longrightarrow & \{ \text{ Applying implication, } sdtp(\psi_2) \ \} \\
 & i_s \vDash_{[s(b),s(e)]} \psi_2 \\
\Longrightarrow & \{ \wedge \text{ introduction } \} \\
 & i_s \vDash_{[s(b),s(e)]} \psi_1 \wedge i_s \vDash_{[s(b),s(e)]} \psi_2 \\
\Longleftrightarrow & \{ \text{ Definition } i \vDash_{[a,b]} \psi_1 \wedge \psi_2 \ \} \\
 & i_s \vDash_{[s(b),s(e)]} \psi_1 \wedge \psi_2
\end{array}
$$

□

Theorem 91. *If ψ_1 and ψ_2 are slowdown truth preserving, so is $\psi_1 \circ \psi_2$. Let s be a slowdown transform.*

Assuming

$$sdtp(\psi_1)$$

and

$$sdtp(\psi_2)$$

we can conclude that

$$i \vDash_{[b,e]} \psi_1 \circ \psi_2 \Rightarrow i_s \vDash_{[s(b),s(e)]} \psi_1 \circ \psi_2$$

Proof.

$$
\begin{array}{ll}
 & \{ \text{ Definition } sdtp(\psi_1) \ \} \\
 & i \vDash_{[b,e]} \psi_1 \Rightarrow i_s \vDash_{[s(b),s(e)]} \psi_1 \\
 & \{ \text{ Definition } sdtp(\psi_2) \ \} \\
 & i \vDash_{[b,e]} \psi_2 \Rightarrow i_s \vDash_{[s(b),s(e)]} \psi_2 \\
 & \{ \text{ Hypothesis } \} \\
 & i \vDash_{[b,e]} \psi_1 \circ \psi_2 \\
\Longleftrightarrow & \{ \text{ Definition } \circ \ \} \\
 & \exists m : [b,e].\ i \vDash_{[b,m]} \psi_1 \wedge i \vDash_{[m,e]} \psi_2 \\
\Longrightarrow & \{ \ sdtp(\psi_1) \text{ and } sdtp(\psi_2) \ \} \\
 & \exists m : [b,e].\ i_s \vDash_{[s(b),s(m)]} \psi_1 \wedge i_s \vDash_{[s(m),s(e)]} \psi_2 \\
\Longrightarrow & \{ \text{ monotonicity of s } \} \\
 & \exists m' : [s(b),s(e)].\ i_s \vDash_{[s(b),m']} \psi_1 \wedge i_s \vDash_{[m',s(e)]} \psi_2 \\
\Longleftrightarrow & \{ \text{ Definition } \circ \ \} \\
 & i_s \vDash_{[s(b),s(e)]} \psi_1 \circ \psi_2
\end{array}
$$

□

Theorem 92. *If ψ is slowdown truth preserving, so is $\langle\psi\rangle_{[c,\infty]}$. Let s be a slowdown transform.*

Assuming

$$sdtp(\psi)$$

then

$$i \vDash_{[b,e]} \langle\psi\rangle_{[c,\infty]} \Rightarrow i_s \vDash_{[s(b),s(e)]} \langle\psi\rangle_{[c,\infty]}$$

Proof.

$$\begin{array}{ll} & \{ \text{ Definition } sdtp(\psi) \ \} \\ & i \vDash_{[b,e]} \psi \Rightarrow i_s \vDash_{[s(b),s(e)]} \psi \\ & \{ \text{ Hypothesis } \} \\ & i \vDash_{[b,e]} \langle\psi\rangle_{[c,\infty]} \\ \iff & \{ \text{ Definition of } \vDash_{[b,e]} \langle\psi\rangle_{[c,\infty]} \ \} \\ & i \vDash_{[b,e]} \psi \wedge e - b \geq c \\ \implies & \{ \text{ Applying implication } sdtp(\psi) \ \} \\ & i_s \vDash_{[s(b),s(e)]} \psi \wedge e - b \geq c \\ \implies & \{ \text{ By interval monotonicity of } s,\ s(e) - s(b) \geq e - b \ \} \\ & i_s \vDash_{[s(b),s(e)]} \psi \wedge s(e) - s(b) \geq c \\ \iff & \{ \text{ Definition of } \vDash_{[b,e]} \langle\psi\rangle_{[c,\infty]} \ \} \\ & i_s \vDash_{[s(b),s(e)]} \langle\psi\rangle_{[c,\infty]} \end{array}$$

□

Theorem 93. *If ψ is slowdown truth preserving, $\psi_{\langle[0,d]\rangle}$ is* not*, in general, slowdown truth preserving.*

Proof. In order to prove this theorem, all we need to do is to find a counter example. In this case we need to find an interpretation i which satisfies the TRE over an interval, and a slowdown transform s such that i_s does not satisfy it any more over the stretched interval

Consider the TRE $\langle a\rangle_{[0,d]}$. We choose a singleton interpretation i with symbol a and length d, which obeys the TRE over the observation interval $[0, d]$. We also choose, for s, the function $s(x) = 2x$. It is easy to see that this function is a slowdown transform since it is continous, maps s(0) to 0, diverges to infinity, and is interval monotonic in the slowdown sense.

Now, when we apply the time transform, the length of the observation interval doubles to $2d$. Thus the resultant interpretation does not satisfy the TRE over the stretched interval, providing the counter example.

□

Lemma 94. *If ψ is slowdown truth preserving, so is ψ^n.*

$$\forall n : \mathbb{N}.\ sdtp(\psi) \Rightarrow sdtp(\psi^n)$$

Proof. By Induction on n.

Base Case: We first prove the result for the special case where the exponent is 0. In this case we will not need the antecedent. This means that we have to prove that

$$i \vDash_{[b,e]} \psi^0 \Rightarrow i_s \vDash_{[s(b),s(e)]} \psi^0$$

$$\begin{array}{ll} & i \vDash_{[b,e]} \psi^0 \\ \iff & \{\ \text{Definition } \psi^0\ \} \\ & b = e \\ \iff & \{\ \text{Consequence of equality, s is a function}\ \} \\ & s(b) = s(e) \\ \iff & \{\ \text{Any interpretation satisfies } \psi^0 \text{ over a point interval}\ \} \\ & i_s \vDash_{[s(b),s(e)]} \psi^0 \end{array}$$

Inductive Case: Assume the statement holds when the exponent is k,

$$sdtp(\psi) \Rightarrow sdtp(\psi^k)$$

which means that We must prove that it also holds when the exponent is $k+1$

$$sdtp(\psi) \Rightarrow sdtp(\psi^{k+1})$$

$$
\begin{array}{ll}
 & \{\ \textit{Sub Proof 1}\ \} \\
 & \{\ \text{Assumption}\ \} \\
 & \textit{sdtp}(\psi) \\
\Longrightarrow & \{\ \text{Using the implication in Inductive Hypothesis}\ \} \\
 & \textit{sdtp}(\psi^k) \\
 & \{\ \textit{Sub Proof 2}\ \} \\
 & \{\ \text{Assumption}\ \} \\
 & i \vDash_{[b,e]} \psi^{k+1} \\
\Longleftrightarrow & \{\ \text{Definition of } \vDash_{[b,e]} \psi^{k+1}\ \} \\
 & i \vDash_{[b,e]} \psi \circ \psi^k \\
\Longleftrightarrow & \{\ \text{Definition of } \vDash_{[b,e]} \psi_1 \circ \psi_2\ \} \\
 & i \vDash_{[b,e]} \psi \wedge i \vDash_{[b,e]} \psi^k \\
\Longrightarrow & \{\ \text{Applying implications of } \textit{sdtp}(\psi) \text{ and } \textit{sdtp}(\psi^k)\ \} \\
 & i_s \vDash_{[s(b),s(e)]} \psi \wedge i_s \vDash_{[s(b),s(e)]} \psi^k \\
\Longleftrightarrow & \{\ \text{Definition of } \vDash_{[b,e]} \psi_1 \circ \psi_2\ \} \\
 & i_s \vDash_{[s(b),s(e)]} \psi \circ \psi^k \\
\Longleftrightarrow & \{\ \text{Definition of } \vDash_{[b,e]} \psi^{k+1}\ \} \\
 & i_s \vDash_{[s(b),s(e)]} \psi^{k+1} \\
 & \{\ \textit{End Sub Proof 2}\ \} \\
\Longrightarrow & i \vDash_{[b,e]} \psi^{k+1} \Rightarrow i_s \vDash_{[s(b),s(e)]} \psi^{k+1} \\
\Longleftrightarrow & \{\ \text{Definition of sdtp}\ \} \\
 & \textit{sdtp}(\psi^{k+1}) \\
 & \{\ \textit{End Sub Proof 1}\ \} \\
\Longrightarrow & \textit{sdtp}(\psi) \Rightarrow \textit{sdtp}(\psi^{k+1})
\end{array}
$$

□

Theorem 95. *If ψ is slowdown truth preserving, so is ψ^*. Let s be a slowdown transform.*

Assuming

$$\textit{sdtp}(\psi)$$

then

$$i \vDash_{[b,e]} \psi^* \Rightarrow i_s \vDash_{[s(b),s(e)]} \psi^*$$

Proof.

$$
\begin{array}{ll}
 & i \vDash_{[b,e]} \psi^* \\
\Longleftrightarrow & \{\ \text{Definition } \psi^*\ \} \\
 & \exists n : \mathbb{N}.\ i \vDash_{[b,e]} \psi^n \\
\Longrightarrow & \{\ \text{Using } \textit{sdtp}(\psi) \text{ and Lemma 94}\ \} \\
 & \exists n : \mathbb{N}.\ i_s \vDash_{[s(b),s(e)]} \psi^n \\
\Longleftrightarrow & \{\ \text{Definition } \psi^*\ \} \\
 & i_s \vDash_{[s(b),s(e)]} \psi^*
\end{array}
$$

□

Theorem 96. *Suppose that ψ is slowdown truth preserving. Then, $\neg\psi$ is not necessarily slowdown truth preserving.*

Proof. By contradiction.

Suppose that if a TRE ψ is slowdown truth preserving, its negation is in general slowdown truth preserving. This means that assuming

$$i \vDash_{[b,e]} \psi \Rightarrow i_s \vDash_{[s(b),s(e)]} \psi$$

we would always be able to prove that

$$i \vDash_{[b,e]} \neg\psi \Rightarrow i_s \vDash_{[s(b),s(e)]} \neg\psi$$

or rather that

$$i \nvDash_{[b,e]} \psi \Rightarrow i_s \nvDash_{[s(b),s(e)]} \psi$$

Now, let $\psi = \langle a \rangle_{[c,\infty]}$, which we know to be slowdown truth preserving, and let us choose some i which satisfies $\neg\psi$ over some observation interval (such as from 0 to its length)

$$i \vDash_{[0,|i|]} (\neg\langle a \rangle_{[c,\infty]})$$

for example, a singleton interpretation i with symbol a and length $\frac{c}{2}$. By definition of negation, we have

$$i \nvDash_{[0,\frac{c}{2}]} \langle a \rangle_{[c,\infty]}$$

Consider now the effect of applying the slowdown transform $s(x) = 3x$ to i. The observation interval will now have a length of $\frac{3x}{2}$, which now satisfies the TRE

$$i_s \vDash_{[0,\frac{3c}{2}]} \langle a \rangle_{[c,\infty]}$$

which is a contradiction to our initial assumption.

□

Theorem 97. *If ψ is speedup truth preserving, so is $\langle \psi \rangle_{[0,d]}$. Let s be a speedup transform.*

Assuming

$$sutp(\psi)$$

we can conclude that

$$i \vDash_{[b,e]} \langle \psi \rangle_{[0,d]} \Rightarrow i_s \vDash_{[s(b),s(e)]} \langle \psi \rangle_{[0,d]}$$

Proof.

$$
\begin{array}{ll}
 & \{ \text{ Definition } sutp(\psi) \ \} \\
 & i \vDash_{[b,e]} \psi \Rightarrow i_s \vDash_{[s(b),s(e)]} \psi \\
 & \{ \text{ Hypothesis } \} \\
 & i \vDash_{[b,e]} \langle \psi \rangle_{[0,d]} \\
\iff & \{ \text{ Definition } \langle \psi \rangle \ \} \\
 & i \vDash_{[b,e]} \psi \wedge e - b \leq d \\
\implies & \{ \text{ Applying implication } sdtp(\psi) \ \} \\
 & i_s \vDash_{[s(b),s(e)]} \psi \wedge e - b \leq d \\
\implies & \{ \text{ By interval anti-monotonicity of } s,\ s(e) - s(b) \leq e - b \ \} \\
 & i_s \vDash_{[s(b),s(e)]} \psi \wedge s(e) - s(b) \leq d \\
\iff & \{ \text{ Definition of } \langle \psi \rangle \ \} \\
 & i_s \vDash_{[s(b),s(e)]} \langle \psi \rangle_{[0,d]}
\end{array}
$$

□

Theorem 98. *Assuming*

$$sdtp(\psi)$$

then

$$\psi_{\langle [c,\infty] \rangle}$$

is, in general, not speedup truth preserving.

Proof. In order to prove this theorem, all we need to do is to find a counter example, in this case an interpretation i which satisfies the TRE over some interval, and a speedup transform s such that i_s will not satisfy the TRE any more on the speeded up interval.

Consider the TRE $\langle a \rangle_{[c,\infty]}$. We choose a singleton interpretation i with symbol a and of length c, which obeys the TRE over the interval $[0, c]$. We also choose as our speedup transform the function $s(x) = \frac{x}{2}$. It is easy to see that this function is a speedup transform since it is continuous, maps s(0) to 0, diverges to infinity, and is interval monotonic in the speedup sense.

Now, if we apply the transform, the length of the interval drops to $\frac{x}{2}$. Therefore, the resultant interpretation cannot satisfy the TRE anymore, since the length of its observation interval is too short. This, in fact breaks the speedup truth preservation property, as required.

□

Theorem 99. *Suppose that ψ is speedup truth preserving. Then, $\neg\psi$ is not necessarily speedup truth preserving.*

Proof. By contradiction.

Suppose that assuming that a TRE ψ is speedup truth preserving, negation of ψ is in general speedup truth preserving. This means that assuming

$$i \vDash_{[b,e]} \psi \Rightarrow i_s \vDash_{[s(b),s(e)]} \psi$$

we would always be able to prove that

$$i \vDash_{[b,e]} \neg\psi \Rightarrow i_s \vDash_{[s(b),s(e)]} \neg\psi$$

or rather that

$$i \nvDash_{[b,e]} \psi \Rightarrow i_s \nvDash_{[s(b),s(e)]} \psi$$

Now, let $\psi = \langle a \rangle_{[0,d]}$, which due to the previous theorems is speedup truth preserving, and let us choose some i which does satisfies $\neg\psi$ over some interval (such as from 0 to its length)

$$i \vDash_{[0,|i|]} \neg(\langle a \rangle_{[0,d]})$$

An interpretation satisfying this constraint would be a singleton interpretation i with symbol a and length $2d$, which satisfies the TRE on the interval $[0, 2d]$. By definition of negation, we have

$$i \nvDash_{[0,2d]} \langle a \rangle_{[0,d]}$$

Consider now the effect of applying the speedup transform $s(x) = \frac{x}{3}$ to i. i The observation interval will now have a length of $\frac{2d}{3}$. But this now satisfies the TRE

$$i_s \vDash_{[0,\frac{2d}{3}]} \langle a \rangle_{[0,d]}$$

which is a contradiction to our initial assumption.

□

Corollary 102. *Let ψ, ψ_1 and ψ_2 be invariant under time transforms. Then all of the following are invariant.*

- $a \in \Sigma$
- $\psi_1 \vee \psi_2$
- $\psi_1 \wedge \psi_2$
- $\psi_1 \circ \psi_2$
- ψ^*

Proof. This occurs as an immediate consequence of the fact that these objects are slowdown truth preserving, and speedup truth preserving. Since this is the case, we can use the switch meta theorems to deduce that they are also speedup and slowdown false preserving. Since they possess all four traits, it is easy to see that these operators are invariant.

□

Lemma 104. *The length of a time transformed signal list is equal to transforming the length of the original signal. Let ξ be a signal list. Then the following holds.*

$$|\xi_s| = s(|\xi|)$$

Proof. By induction on ξ.

Base Case: $\xi = \langle\rangle$

$$\begin{array}{ll} & |\langle\rangle_s| \\ = & \{ \text{ Definition of time transform on signal list } \} \\ & |\langle\rangle| \\ = & \{ \text{ Definition of length of a signal list } \} \\ & 0 \\ = & \{ \text{ For any time transform, s(0) = 0 } \} \\ & s(0) \\ = & \{ \text{ Length of Empty Signal List is 0 } \} \\ & s(|\langle\rangle|) \end{array}$$

Inductive Case: Assume

$$|\xi'_s| = s(|\xi'|)$$

then

$$|(\xi' \overset{\text{sig}}{;} (a,k))_s| = s(|\xi' \overset{\text{sig}}{;} (a,k)|)$$

$$\begin{array}{ll} & |(\xi' \overset{\text{sig}}{;} (a,k))_s| \\ = & \{ \text{ Definition of time transform on signal list } \} \\ & |(\xi'_s \overset{\text{sig}}{;} (a, s(k + |\xi'|) - |\xi'_s|))| \\ = & \{ \text{ Definition of length on signal list } \} \\ & |\xi'_s| + s(k + |\xi'|) - |\xi'_s| \\ = & \{ \text{ Simplification } \} \\ & s(|\xi'| + k) \\ = & \{ \text{ Definition of length on signal list } \} \\ & s(|\xi' \overset{\text{sig}}{;} (a,k)|) \end{array}$$

□

Proposition 105. *A signal list ξ is equal to a transformed list whenever the time transform is the identity function. Let ξ be a signal list. Then,*

$$\xi_{id} = \xi$$

Proof. By induction on ξ.

Base Case: $\xi = \langle\rangle$

$$\begin{array}{ll} & \langle\rangle_{id} \\ = & \{ \text{ Definition of time transform on empty signal list } \} \\ & \langle\rangle \end{array}$$

Inductive Case: Assume

$$\xi'_{id} = \xi'$$

then

$$(\xi' \overset{\text{sig}}{;} (a,k))_{id} = \xi' \overset{\text{sig}}{;} (a,k)$$

$$
\begin{aligned}
& (\xi' \overset{\text{sig}}{;} (a,k))_{id} \\
= \quad & \{ \text{ Definition of Time Transform on Signal List } \} \\
& \xi'_{id} \overset{\text{sig}}{;} (a, id(k + |\xi'|) - |\xi'_{id}|) \\
= \quad & \{ \text{ id(x) = x } \} \\
& \xi'_{id} \overset{\text{sig}}{;} (a, k + |\xi'| - |\xi'_{id}|) \\
= \quad & \{ \text{ Lemma 104 } \} \\
& \xi'_{id} \overset{\text{sig}}{;} (a, k + |\xi'| - id(|\xi'|)) \\
= \quad & \{ \text{ id(x) = x } \} \\
& \xi'_{id} \overset{\text{sig}}{;} (a, k + |\xi'| - |\xi'|) \\
= \quad & \{ \text{ Simplification } \} \\
& \xi'_{id} \overset{\text{sig}}{;} (a, k) \\
= \quad & \{ \text{ Inductive Hypothesis } \} \\
& \xi' \overset{\text{sig}}{;} (a, k)
\end{aligned}
$$

□

Proposition 106. *Let i be a signal list, and let s_1 and s_2 be time transforms. Then,*

$$\xi_{s_1 \circ s_2} = (\xi_{s1})_{s2}$$

Proof. By induction on ξ.

Base Case: $\xi = \langle\rangle$

$$
\begin{aligned}
& (\langle\rangle_{s1})_{s2} \\
= \quad & \{ \text{ Definition of time transform on empty signal list } \} \\
& (\langle\rangle)_{s2} \\
= \quad & \{ \text{ Definition of time transform on empty signal list } \} \\
& \langle\rangle \\
= \quad & \{ \text{ Definition of time transform on empty signal list } \} \\
& \langle\rangle_{s_1 \circ s_2}
\end{aligned}
$$

Inductive Case: Assume

$$(\xi'_{s_1})_{s2} = \xi'_{s_1 \circ s_2}$$

then

$$((\xi' \overset{\text{sig}}{;} (a,k))_{s_1})_{s2} = (\xi' \overset{\text{sig}}{;} (a,k))_{s_1 \circ s_2}$$

$$
\begin{aligned}
& ((\xi' \overset{\text{sig}}{;} (a,k))_{s_1})_{s2} \\
= \quad & \{ \text{ Definition of Time Transform on Signal List } \} \\
& (\xi'_{s_1} \overset{\text{sig}}{;} (a, s_1(k + |\xi'|) - |\xi'_{s_1}|))_{s_2} \\
= \quad & \{ \text{ Definition of Time Transform on Signal List } \} \\
& (\xi'_{s_1})_{s_2} \overset{\text{sig}}{;} (a, s_2(s_1(k + |\xi'|) - |\xi'_{s_1}| + |\xi'_{s_1}|) - |(\xi'_{s_1})_{s_2}|) \\
= \quad & \{ \text{ Simplification } \} \\
& (\xi'_{s_1})_{s_2} \overset{\text{sig}}{;} (a, s_2(s_1(k + |\xi'|)) - |\xi'_{s1s2}|) \\
= \quad & \{ \text{ Lemma 104, applied twice } \} \\
& (\xi'_{s_1})_{s_2} \overset{\text{sig}}{;} (a, s_2(s_1(k + |\xi'|)) - s_2(s_1(|\xi'|)) \\
= \quad & \{ \text{ Inductive Hypothesis } \} \\
& (\xi'_{s_1 \circ s_2} \overset{\text{sig}}{;} (a, s_2(s_1(k + |\xi'|)) - s_2(s_1(|\xi'|)) \\
= \quad & \{ \text{ Definition of functional composition } \} \\
& (\xi'_{s_1 \circ s_2} \overset{\text{sig}}{;} (a, s_1 \circ s_2(k + |\xi'|) - s_1 \circ s_2(|\xi'|)) \\
= \quad & \{ \text{ Lemma 104 } \} \\
& (\xi'_{s_1 \circ s_2} \overset{\text{sig}}{;} (a, s_1 \circ s_2(k + |\xi'|) - (|\xi'_{s_1 \circ s_2}|)) \\
= \quad & \{ \text{ Definition of Time Transform on Signal List } \} \\
& (\xi' \overset{\text{sig}}{;} (a,k))_{s_1 \circ s_2}
\end{aligned}
$$

□

Proposition 107. *If an interpretation attains a value for a symbol over an interval between two critical points then, that value over the transformed interval is unchanged. Let i be an interpretation. Formally, we can say the following. Consider its critical point list, obtained by ordering C(i). Take any two consecutive critical points c_i and c_{i+1}. If $i(\alpha, t) = true$ for all points in $[c_1, c_2)$, then $i_s(\alpha, s(t)) = true$ for all points in $[s(c_1), s(c2))$. The same result holds for $i(\alpha, t) = false$.*

Proof. If a point in $[c_1, c_2)$ has a certain value under a particular interpretation, then, by Definition 39, $i_s(s(t))$ will have the same truth value. Since the property holds for all values in $[c_1, c_2)$, it will thus hold for all values in $[s(c_1), s(c_2))$ for i_s.

□

Proposition 108. *The set of critical points of a transformed interpretation consists of the set of transformed critical points (of the original interpretation). Let C(i) be the set of critical points of i, and let s be a time transform. Then,*

$$C(i_s) = \{s(x) \mid x \in C(i)\}.$$

Proof. One has to prove two things. The first is that transforming an interpretation does not introduce any additional critical points. The second is that transforming a critical point of i results in obtaining a critical point of i_s.

We start with the first requirement. Suppose, for contradiction, that transforming an interpretation introduces a new critical point at some location $s(x')$. Since a critical

point changes the value of the interpretation from one value of Σ to another, $i(\alpha, x') \neq i_s(\alpha, s(x'))$, which contradicts the definition of a transforming an interpretation.

We now pass to the second stage of the proof. Any set of critical points C(i) will contain 0 as a member. s(0) = 0 is a critical point of i_s by definition. Consider the sequence of critical points in C(i): $\langle c_1, c_2, \ldots, c_n \rangle$. We now prove that if $s(c_i)$ is a critical point of i_s, so is $s(c_{i+1})$. Since this holds for $s(c_0)$, the statement would then hold by induction. Consider $s(c_{i+1})$. We know that if $i(\alpha, c_i) = true$, then $i(\beta, c_{i+1}) = true$ for some $\beta \neq \alpha$. Now, since a time transform preserves the values of i, $i_s(\beta, s(c_{i+1}))$ is also $true$.

Now, it is the case that by proposition 29, the interval $[c_i, c_i + 1)$ possesses the property that the values of points in this interval do not change their value. By proposition, 107, this means that points in the transformed interval $[s(c_i), s(c_{i+1}))$ do not change their values either. Thus,

$$\lim_{t \to s(c_{i+1})^-} i_s(\alpha, t) \wedge \neg i_s(\alpha, (c_{i+1}))$$

which by definition 25, means that $s(c_{i+1})$ is a critical point of i_s.

□

Proposition 109. *The largest (last) critical point of an interpretation i, when transformed, remains the largest (last) of i_s. We can state this fact as*

$$s(max(C(i))) = max(C(i_s))$$

Proof. From Proposition 108, we know that

$$C(i_s) = \{s(x) \mid x \in C(i)\}$$

Now, max(C(i)) is the largest value in C(i). Since a time transform insists that if $t1 < t2$ then $s(t1) < s(t2)$, when all of the points in $x \in C(i)$ are transformed, $s(max(C(i)))$ will still be greater than all of the other points. Therefore, it will also be $max(C(i_s))$.

□

Corollary 110. *The length of a transformed interpretation is equal to transforming the length of the original interpretation. Let i be an interpretation. Then, we can state this as $|i_s| = s(|i|)$.*

Proof. This result is simple to arrive to. By Proposition 21 the length of an interpretation is the value of its largest critical point. The last critical point of i occurs at $|i|$. By Proposition 109, the last critical point of i_s is the last critical point of i, with s applied to it. Thus, the corollary is proven.

□

Lemma 111. *Consider a critical point list $xs = \langle (c_1, \alpha), \ldots, (c_n, \bot) \rangle$. Then the effect of applying a time transform to it can be characterised by the following statement. $xs_s = \langle (s(c_1), \alpha), \ldots, (s(c_n), \bot) \rangle$.*

Proof. By Induction on the length of xs.

Base Case: Required to prove $\langle (0, \bot) \rangle_s = \langle (s(0), \bot) \rangle$

$$
\begin{array}{ll}
 & \langle (0, \bot) \rangle_s \\
= & \{ \text{ Definition of Time Transform Over CPL } \} \\
 & \langle (0, \bot) \rangle \\
= & \{ \text{ s(0) = 0 } \} \\
 & \langle (s(0), \bot) \rangle
\end{array}
$$

Inductive Case: Assume

$$\langle (c_1, \alpha_1), \ldots, (c_k, \alpha_k) \rangle_s = \langle (s(c_1), \alpha_1), \ldots, (s(c_k), \alpha_k) \rangle$$

Required to Prove

$$\langle (c_1, \alpha_1), \ldots, (c_k, \alpha_k), (c_{k+1}, \alpha_{k+1}) \rangle_s = \langle (s(c_1), \alpha_1), \ldots, (s(c_k), \bot), (s(c_{k+1}), \alpha_{k+1}) \rangle$$

$$
\begin{array}{ll}
 & \langle (c_1, \alpha_1), \ldots, (c_k, \alpha_k), (c_{k+1}, \alpha_{k+1}) \rangle_s \\
= & \{ \text{ Definition of Time Transform Over CPL } \} \\
 & \langle (c_1, \alpha_1), \ldots, (c_k, \alpha_k) \rangle_s : (s(c_{k+1}), \alpha_{k+1}) \\
= & \{ \text{ Inductive Hypothesis } \} \\
 & \langle (s(c_1), \alpha_1), \ldots, (s(c_k), \alpha_k) \rangle : (s(c_{k+1}), \alpha_{k+1}) \\
= & \{ \text{ List Append } \} \\
 & \langle (s(c_1), \alpha_1), \ldots, (s(c_k), \alpha_k), (s(c_{k+1}), \alpha_{k+1}) \rangle
\end{array}
$$

□

Lemma 112. *The length of a transformed critical point list is equal to transforming the length of the original critical point list. Let xs be a critical point list. Then,*

$$|(xs)_s| = s(|xs|)$$

Proof. By definition, a critical point list always has the form $xs : (t, \bot)$. Consider $(xs : (t, \bot))_s$. By definition of time transforms over the critical point list, this expression resolves to $xs_s : (s(t), \bot)$. By the definition of length of critical point lists, the length of this entity is $s(t)$. But this is exactly equal to $s(|xs : (t, \bot)|)$, which proves the theorem.

□

Lemma 113. *If two interpretations agree in value at every s(t) and every symbol, then they agree in value for every time point t. Let i and j be interpretations. Then,*

$$\forall \alpha : \Sigma, t : \mathbb{T}.\ i(\alpha, s(t)) = j(\alpha, s(t)) \Rightarrow \forall \alpha : \Sigma, t : \mathbb{T}.\ i(\alpha, t) = j(\alpha, t)$$

Proof. Let us begin by assuming the antecedent. We know that s, since it is a time transform, is bijective. Therefore, s must also be surjective. Now, the domain of s is $\mathbb{T}$, and so is its range. Assume, for contradiction, that there exists some point t such that the consequent is not true. Now, $t \in \mathbb{T}$. Since it is in the range of s, and s is surjective, then there must exist some point t' such that t = s(t'). Since, t is now of the form s(x), by the antecedent it is clear that the property holds, which is a contradiction. Therefore the consequent must be true, given that the antecedent is true.

□

Theorem 114. *A time transform distributes over* $\overset{I}{;}$ *into a time transform and a shifted time transform. Let s be a time transform. Then, the following law holds.*

$$(i \overset{I}{;} j)_s = i_s \overset{I}{;} j_{s\dashrightarrow|i|}$$

Proof. We begin the proof by outlining what we know. We start with a reminder for the expression for $i \overset{\mathrm{I}}{;} j$

$$\begin{aligned} i \overset{\mathrm{I}}{;} j(\alpha, t) &= i(\alpha, t) \text{ for } 0 \leq t < |i| \\ & \quad\ j(\alpha, t - |i|) \text{ for } |i| \leq t \end{aligned}$$

If we replace i by i_s and j by $j_{s\dashrightarrow|i|}$, in the formula above we can obtain an expression for $i_s \overset{\mathrm{I}}{;} j_{s\dashrightarrow|i|}$

$$\begin{aligned} i_s \overset{\mathrm{I}}{;} j_{s\dashrightarrow|i|}(\alpha, t) &= i_s(\alpha, t) \text{ for } 0 \leq t < |i_s| \\ & \quad\ j_{s\dashrightarrow|i|}(\alpha, t - |i_s|) \text{ for } |i_s| \leq t \end{aligned}$$

The problem we encounter now is that we do not know an expression for the formula $(i \overset{\mathrm{I}}{;} j)_s$. How can we continue in order to prove the equality stated in the theorem? What we do have is a logical expression establishing the relationship between an interpretation and its transformed version.
We know that

$$\forall t : \mathbb{T}.\ i_s(\alpha, s(t)) = i(\alpha, t)$$

Substituting $i \overset{\mathrm{I}}{;} j$ for i, gives us an important piece of information:

$$\forall t : \mathbb{T}.\ (i \overset{\mathrm{I}}{;} j)_s(\alpha, s(t)) = i \overset{\mathrm{I}}{;} j(\alpha, t) \ldots (1)$$

This is telling us that for every time point t, the value of $i \overset{\mathrm{I}}{;} j$ at that time point is equal to that of $(i \overset{\mathrm{I}}{;} j)_s$ at the transformed point under s. What if we could prove that

$$\forall t : \mathbb{T}.\ i_s \overset{\mathrm{I}}{;} j_{s\dashrightarrow|i|}(\alpha, s(t)) = i \overset{\mathrm{I}}{;} j(\alpha, t) \ldots (2)$$

That is, that for every time point t, the value of i at that time point is equal to that of $i_s \overset{\mathrm{I}}{;} j_{s\dashrightarrow|i|}$ at the transformed point under s? By consequence we would have proved that for all time points $(i \overset{\mathrm{I}}{;} j)_s$ and $i_s \overset{\mathrm{I}}{;} j_{s\dashrightarrow|i|}$ are identical. We can write this as:

$$\forall t : \mathbb{T}.\ i_s \overset{\text{I}}{;} j_{s\dashrightarrow|i|}(\alpha, s(t)) = (i \overset{\text{I}}{;} j)_s(\alpha, s(t)) \ldots (3)$$

which by Lemma 113 implies

$$\forall t : \mathbb{T}.\ i_s \overset{\text{I}}{;} j_{s\dashrightarrow|i|}(\alpha, t) = (i \overset{\text{I}}{;} j)_s(\alpha, t) \ldots (4)$$

So, if we manage to prove (2), we shall have managed to prove (4), which would complete the theorem.

We need to prove that for all time points,

$$i_s \overset{\text{I}}{;} j_{s\dashrightarrow|i|}(\alpha, s(t)) = i \overset{\text{I}}{;} j(\alpha, t)$$

Let us choose any point t. Then by the law of the excluded middle,

$$t < |i| \vee t \geq |i|$$

We shall prove that in either of these two cases, the equality holds.

Case 1: $t < |i|$

Consider the formula $i \overset{\text{I}}{;} j(\alpha, t)$. What is the value of $i \overset{\text{I}}{;} j$ at that point? By the definition of the $\overset{\text{I}}{;}$ operator, since $t < |i|$, it is equal to $i(\alpha, t)$. Let us now consider the formula $i_s \overset{\text{I}}{;} j_{s\dashrightarrow|i|}$. What is its value at time s(t)? By monotonicity of time transforms if $t < |i|$, then $s(t) < s(|i|)$. By this fact, and the definition of the $\overset{\text{I}}{;}$ operator, it is equal to $i_s(\alpha, s(t))$. Now, by definition of the relationship between an interpretation and its transformed version, we know that

$$\forall t : \mathbb{T}.\ i_s(\alpha, s(t)) = i(\alpha, t)$$

Therefore the two expressions above are in fact equal, which proves this part of the theorem.

Case 2: $t \geq |i|$.

Consider the formula $i \overset{\text{I}}{;} j(\alpha, t)$. What is the value of $i \overset{\text{I}}{;} j$ at that point? By the definition of the $\overset{\text{I}}{;}$ operator, since $t \geq |i|$, it is equal to $j(\alpha, t - |i|)$. Let us now consider the formula $i_s \overset{\text{I}}{;} j_{s\dashrightarrow|i|}$. What is its value at time s(t)? By monotonicity of time transforms if $t > |i|$, then $s(t) > s(|i|)$. By this fact, and the definition of the $\overset{\text{I}}{;}$ operator, $i_s \overset{\text{I}}{;} j_{s\dashrightarrow|i|}$ is equal to $j_{s\dashrightarrow|i|}(\alpha, s(t) - |i_s|)$. How can we prove that these two expressions equivalent? From the definition of the relationship between an interpretation and itself under a shifted time transform, we know that

$$\forall t : \mathbb{T}.\ j_{s\dashrightarrow|i|}(\alpha, s(t + |i|) - s(|i|)) = j(\alpha, t)$$

If we substitute t for $t - |i|$ in the above formula, we get

$$\forall t : \mathbb{T}.\ j_{s\dashrightarrow|i|}(\alpha, s(t - |i| + |i|) - s(|i|)) = j(\alpha, t - |i|)$$

which simplifies to

$$\forall t : \mathbb{T}.\ j_{s\dashrightarrow|i|}(\alpha, s(t) - s(|i|)) = j(\alpha, t - |i|)$$

which asserts equality of our two expressions.

□

Theorem 115. *Time transforms distribute through the function i2c. Let i be an interpretation and s a time transform. Then,* $i2c(i)_s = i2c(i_s)$

Proof. Let us consider $i2c(i)$. Recall that this is achieved by ordering $\overline{C(i)}$. By Corollary 24 this process yields a critical point list $\langle(c_1, \alpha), \ldots, (c_n, \bot)\rangle$. By Lemma 111, we know that $i2c(i)_s$ will therefore yield $\langle(s(c_1), \alpha), \ldots, (s(c_n), \bot)\rangle$.

Now consider i_s. By Proposition 108, we know that the critical points of $i_s = \{s(x) | x \in C(i)\}$. Due to the monotonicity of time transforms if $x_1 < x_2$, then $s(x_1) < s(x_2)$. Therefore when this set is ordered by $i2c$ it will yield the sequence $\langle(s(c_1), \alpha), \ldots, (s(c_n), \bot)\rangle$. This concludes the proof of the theorem.

□

Theorem 116. *Time transforms distribute through the function c2s. Let xs be a critical point list, and s a time transform. Then* $c2s(xs_s) = (c2s(xs))_s$.

Proof. By Induction on xs.

Base Case: $xs = \langle(0, \bot)\rangle$

$$\begin{array}{ll} & c2s(\langle(0, \bot)\rangle)_s \\ = & \{\text{ Definition of c2s }\} \\ & \langle\rangle_s \\ = & \{\text{ Definition on Time Transform on Signal List }\} \\ & \langle\rangle \\ = & \{\text{ Definition of c2s }\} \\ & c2s(\langle(0, \bot)\rangle) \\ = & \{\text{ Definition of Time Transform on CPL }\} \\ & c2s(\langle(0, \bot)\rangle_s) \end{array}$$

Inductive Case: Assuming

$$c2s(xs' : (t_1, \bot))_s = c2s((xs' : (t_1, \bot))_s)$$

then,

$$(c2s(xs' : (t_1, \bot) \overset{\text{cp}}{;} (0, \alpha)(t_2, \bot)))_s = (c2s((xs' : (t_1, \bot) \overset{\text{cp}}{;} (0, \alpha)(t_2, \bot)))_s)$$

$$
\begin{array}{ll}
 & c2s((xs : (t_1, \bot) \overset{\text{cp}}{;} (0, \alpha) : (t_2, \bot))_s) \\
= & \{ \text{ Def } \overset{\text{cp}}{;} \ \} \\
 & c2s(xs : (t_1, \alpha) : (t_1 + t_2, \bot))_s \\
= & \{ \text{ Definition of time transforms on CPL (applied twice) } \} \\
 & c2s(xs_s : (s(t_1), \alpha) : (s(t_1 + t_2), \bot)) \\
= & \{ \text{ Definition of c2s } \} \\
 & c2s(xs_s : (s(t_1), \alpha)) \overset{\text{sig}}{;} (s(t_1 + t_2) - s(t_1), \alpha) \\
= & \{ \text{ Definition of time transforms on CPL } \} \\
 & c2s((xs : (t_1, \bot))_s) \overset{\text{sig}}{;} (s(t_1 + t_2) - s(t_1), \alpha) \\
= & \{ \text{ Inductive Hypothesis } \} \\
 & c2s(xs : (t_1, \bot))_s \overset{\text{sig}}{;} (s(t_1 + t_2) - s(t_1), \alpha) \\
= & \{ \text{ Lemma } |c2s(ys)| = |ys| \text{ and Length of a CPL } \} \\
 & c2s(xs : (t_1, \bot))_s \overset{\text{sig}}{;} (s(|c2s(xs : (t_1, \bot))| + t_2) - |(c2s(xs)_{t_1})_s|, \alpha) \\
= & \{ \text{ Collecting s, over a signal list, Definition 50 } \} \\
 & (c2s(xs : (t_1, \bot)) \overset{\text{sig}}{;} (t_2, \alpha))_s \\
= & \{ \text{ Adding and Subtracting } t_1 \text{ changes nothing } \} \\
 & (c2s(xs : (t_1, \bot)) \overset{\text{sig}}{;} (t_1 + t_2 - t_1, \alpha))_s \\
= & \{ \text{ Definition of c2s } \} \\
 & c2s(xs : (t_1, \alpha) : (t_1 + t_2, \bot))_s \\
= & \{ \text{ Definition } \overset{\text{cp}}{;} \ \} \\
 & c2s(xs : (t_1, \bot) \overset{\text{cp}}{;} (0, \alpha)(t_2, \bot))_s
\end{array}
$$

□

Theorem 117. *Time transforms distribute through the function c2i. Let xs be a critical point list and s a time transform. Then,* $c2i(xs)_s = c2i(xs_s)$

Proof. By Induction on xs.

Base Case: $xs = \langle (0, \bot) \rangle$

$$
\begin{array}{ll}
 & c2i(\langle (0, \bot) \rangle)_s \\
= & \{ \text{ Definition of c2i } \} \\
 & False_s \\
= & \{ \text{ Slowdown does not alter truth at any point in an interpretation } \} \\
 & False
\end{array}
$$

Inductive Case: Assuming

$$c2i(xs' : (t, \bot))_s = c2i((xs' : (t, \bot))_s)$$

then,

$$c2i(xs' : (t_1, \bot) \overset{\text{cp}}{;} (0, \alpha)(t_2, \bot))_s = c2i((xs' : (t, \bot) \overset{\text{cp}}{;} (0, \alpha)(t_2, \bot))_s)$$

$$
\begin{array}{ll}
 & c2i(xs' : (t_1, \bot) \overset{\text{cp}}{;} (0, \alpha)(t_2, \bot))_s \\
= & \{ \text{ Definition of } \overset{\text{cp}}{;} \ \} \\
 & c2i(xs' : (t_1, \alpha) : (t_1 + t_2, \bot))_s \\
= & \{ \text{ Definition of c2s } \} \\
 & c2i(xs' : (t_1, \bot) \overset{\text{I}}{;} \left\{ \begin{array}{l} i(\alpha, t) = \text{true for } 0 \leq t < t_2 \\ i(\alpha, t) = \text{false for } t \geq t_2 \\ i(\beta, t) = \text{false for } t \geq 0 \end{array} \right.)_s \\
= & \{ \text{ Lemma } |c2i(xs')| = |xs'|, \text{ and Definition of length of CPL } \} \\
 & (\overbrace{c2i(xs' : (t_1, \bot))}^{|c2i(xs' : (t_1, \bot))| = t_1} \overset{\text{I}}{;} \left\{ \begin{array}{l} i(\alpha, t) = \text{true for } 0 \leq t < t_2 \\ i(\alpha, t) = \text{false for } t \geq t_2 \\ i(\beta, t) = \text{false for } t \geq 0 \end{array} \right.)_s \\
= & \{ \text{ Theorem 114 : Distributing Time Transforms over } \overset{\text{I}}{;} \ \} \\
 & (c2i(xs' : (t_1, \bot)))_s \overset{\text{I}}{;} (\left\{ \begin{array}{l} i(\alpha, t) = \text{true for } 0 \leq t < t_2 \\ i(\alpha, t) = \text{false for } t \geq t_2 \\ i(\beta, t) = \text{false for } t \geq 0 \end{array} \right.)_{s \dashrightarrow t_1} \\
= & \{ \text{ Definition of } s \dashrightarrow t_1 \ \} \\
 & (c2i(xs' : (t_1, \bot)))_s \overset{\text{I}}{;} (\left\{ \begin{array}{l} i_{s \dashrightarrow |t_1|}(\alpha, s(t + t_1) - s(t_1)) = \text{true for } 0 \leq t < t_2 \\ i_{s \dashrightarrow |t_1|}(\alpha, s(t + t_1) - s(t_1)) = \text{false for } t \geq t_2 \\ i_{s \dashrightarrow |t_1|}(\beta, s(t + t_1) - s(t_1)) = \text{false for } t \geq 0 \end{array} \right.)
\end{array}
$$

Let us stop for a moment from the left hand side and examine the right hand side.

$$
\begin{array}{ll}
 & c2i((xs' : (t_1, \bot) \overset{\text{cp}}{;} \langle (0, \alpha), (t_2, \bot) \rangle)_s) \\
= & \{ \text{ Definition of } \overset{\text{cp}}{;} \ \} \\
 & c2i((xs' : (t_1, \alpha) : (t_1 + t_2, \bot))_s) \\
= & \{ \text{ Time transform on CPL, applied twice } \} \\
 & c2i(xs'_s : (s(t_1), \alpha) : (s(t_1 + t_2), \bot)) \\
= & \{ \text{ Definition of c2i } \} \\
 & (c2i(xs'_s : (s(t_1), \bot))) \overset{\text{I}}{;} (\left\{ \begin{array}{l} j(\alpha, t) = \text{true for } 0 \leq t < s(t_1 + t_2) - s(t_1) \\ j(\alpha, t) = \text{false for } t \geq s(t_1 + t_2) - s(t_1) \\ j(\beta, t) = \text{false for } t \geq 0 \end{array} \right.) \\
= & \{ \text{ Time transform on CPL } \} \\
 & c2i((xs' : (t_1, \bot))_s) \overset{\text{I}}{;} (\left\{ \begin{array}{l} j(\alpha, t) = \text{true for } 0 \leq t < s(t_1 + t_2) - s(t_1) \\ j(\alpha, t) = \text{false for } t \geq s(t_1 + t_2) - s(t_1) \\ j(\beta, t) = \text{false for } t \geq 0 \end{array} \right.)
\end{array}
$$

We can immediately notice that the first part of the statement we want to arrive to, $c2i((xs' : (t_1, \bot))_s)$ can be directly derived from the antecedent by the use of the inductive hypothesis. Therefore the problem reduces to proving that the two statements on the right of the $\overset{\text{I}}{;}$ operator, are in fact, the same.

We note that for the symbol β the functions are already in agreement. Therefore we only have to reconcile these functions for the value of α. We therefore note that for any singleton interpretation k with length $|k|$ the value of the interpretation at the symbol α can be written in terms of a boolean expression as follows.

$$k(t) \triangleq 0 \leq t \leq |k|$$

Following this notion, we can immediately see that we can write the interpretation expression for the left hand side as

$$i'(s(t + t_1) - s(t_1)) \triangleq 0 \leq t \leq t_2$$

and the interpretation expression of the right hand side as

$$j'(t) \triangleq 0 \leq t \leq s(t_2 + t_1) - s(t_1)$$

Let us now show that we can reduce the antecedent to the consequent.

$$\begin{array}{ll} & i'(s(t + t_1) - s(t_1)) \\ = & \{ \text{ Changing the variable. Let } x = (s(t + t_1) - s(t_1)) \ \} \\ & i'(x) \\ = & \{ \text{ Adding and subtracting } s(t_1) \ \} \\ & i'(x + s(t_1) - s(t_1)) \\ = & \{ \text{ Grouping } \} \\ & i'((x + s(t_1)) - s(t_1)) \\ = & \{ \ s.s^{-1} = id \ \} \\ & i'(s(s^-1(x + s(t_1))) - s(t_1)) \\ = & \{ \text{ Adding and Subtracting } t_1 \ \} \\ & i'(s(s^-1(x + s(t_1)) - t_1 + t_1) - s(t_1)) \\ = & \{ \text{ Grouping } \} \\ & i'(s(\{s^-1(x + s(t_1)) - t_1\} + t_1) - s(t_1)) \\ = & \{ \text{ Replacing Expression within } \{\} \text{ with } \alpha \ \} \\ & i'(s(\alpha + t_1) - s(t_1)) \\ = & \{ \text{ Expression is of the form } i'(s(t + t_1) - s(t_1)) \ \} \\ & 0 \leq s^{-1}(x + s(t_1)) - t_1 \leq t_2 \\ = & \{ \text{ Adding } t_1 \text{ to inequality } \} \\ & t_1 \leq s^{-1}(x + s(t_1)) \leq t_2 + t_1 \\ = & \{ \text{ Applying s to inequality } \} \\ & s(t_1) \leq s(s^{-1}(x + s(t_1))) \leq s(t_2 + t_1) \\ = & \{ \ s.s^{-1} = id \ \} \\ & s(t_1) \leq x + s(t_1) \leq s(t_2 + t_1) \\ = & \{ \text{ Subtracting } s(t_1) \text{ from the inequality } \} \\ & 0 \leq x \leq s(t_2 + t_1) - s(t_1) \\ = & \{ \text{ Definition } j'(x) \ \} \\ & j'(x) \end{array}$$

□

Theorem 118. *Time transforms distribute through the function s2c. Let ξ be a signal in list form and s a time transform. Then $s2c(\xi_s) = (s2c(\xi))_s$.*

Proof. By Induction on ξ.

Base Case: $\xi = \langle\rangle$

$$\begin{array}{ll} & s2c(\langle\rangle_s) \\ = & \{ \text{ Definition Time Transform on a Signal List } \} \\ & s2c(\langle\rangle) \\ = & \{ \text{ Definition of s2c } \} \\ & \langle(0, \perp)\rangle \\ = & \{ \text{ Definition of Time Transform on CPL } \} \\ & \langle(0, \perp)\rangle_s \\ = & \{ \text{ Definition of s2c } \} \\ & s2c(\langle\rangle)_s \end{array}$$

Inductive Case: Assuming

$$s2c(\xi_s) = s2c(\xi)_s$$

then

$$s2c((\xi \overset{\text{sig}}{;} (k, \alpha))_s) = s2c(\xi \overset{\text{sig}}{;} (k, \alpha))_s$$

$$
\begin{array}{ll}
 & s2c((\xi \overset{\text{sig}}{;} (k,\alpha))_s) \\
= & \{ \text{ Definition of Time Transform on Signal List } \} \\
 & s2c((\xi_s \overset{\text{sig}}{;} (s(k+|\xi|)-|\xi_s|,\alpha))) \\
= & \{ \text{ Definition of s2c } \} \\
 & s2c(\xi_s) \overset{\text{cp}}{;} (0,\alpha)(s(k+|\xi|)-|\xi_s|,\bot) \\
= & \{ \text{ Inductive Hypothesis } \} \\
 & s2c(\xi)_s \overset{\text{cp}}{;} (0,\alpha)(s(k+|\xi|)-|\xi_s|,\bot) \\
= & \{ \text{ s2c has type CPL } \} \\
 & (xs : (t,\bot))_s \overset{\text{cp}}{;} (0,\alpha)(s(k+|\xi|)-|\xi_s|,\bot) \\
= & \{ \ |s2c(\xi)| = |\xi| \text{ and } s2c(\xi) = xs : (t,\bot) \text{ and Def. Length of CPL } \} \\
 & (xs : (|\xi|,\bot))_s \overset{\text{cp}}{;} (0,\alpha)(s(k+|\xi|)-|\xi_s|,\bot) \\
= & \{ \text{ Definition of Time Transform on CPL } \} \\
 & (xs_s : (s(|\xi|),\bot)) \overset{\text{cp}}{;} (0,\alpha)(s(k+|\xi|)-|\xi_s|,\bot) \\
= & \{ \text{ If } \xi \text{ is a signal, } s(\xi) = \xi_s \ \} \\
 & (xs_s : (|\xi_s|,\bot)) \overset{\text{cp}}{;} (0,\alpha)(s(k+|\xi|)-|\xi_s|,\bot) \\
= & \{ \text{ Definition of } \overset{\text{cp}}{;} \ \} \\
 & xs_s : (|\xi_s|,\alpha) : (s(k+|\xi|)-|\xi_s|+|\xi_s|,\bot) \\
= & \{ \text{ Simplification } \} \\
 & xs_s : (|\xi_s|,\alpha) : (s(k+|\xi|),\bot) \\
= & \{ \text{ If } \xi \text{ is a signal, } s(|\xi|) = |\xi_s| \ \} \\
 & xs_s : (s(|\xi|),\alpha) : (s(k+|\xi|),\bot) \\
= & \{ \text{ Definition of Time Transform on CPL, applied twice } \} \\
 & (xs : (|\xi|,\alpha) : (k+|\xi|,\bot))_s \\
= & \{ \text{ Definition of } \overset{\text{cp}}{;} \ \} \\
 & (xs : (|\xi|,\bot) \overset{\text{cp}}{;} (0,\alpha)(k+|\xi|-|\xi|,\bot))_s \\
= & \{ \text{ Simplification } \} \\
 & (xs : (|\xi|,\bot) \overset{\text{cp}}{;} (0,\alpha)(k,\bot))_s \\
= & \{ \text{ Substituting back } xs : (|\xi|,\bot) \text{ for } s2c(\xi) \ \} \\
 & (s2c(\xi) \overset{\text{cp}}{;} (0,\alpha)(k,\bot))_s \\
= & \{ \text{ Definition of s2c } \} \\
 & (s2c(\xi : (k,\alpha)))_s
\end{array}
$$

□

Bibliography

[ACM97] E. Asarin, P. Caspi, and O. Maler. A kleene theorem for timed automata. In *LICS '97: Proceedings of the 12th Annual IEEE Symposium on Logic in Computer Science*, page 160, Washington, DC, USA, 1997. IEEE Computer Society.

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[BLR] Ahmed Bouajjani, Yassine Lakhnech, and Riadh Robbana. From duration calculus to linear hybrid automata. In *In CAV, volume 939 of LNCS*, pages 196–210. Springer-Verlag.

[CHR91] Z. ChaoChen, C.A.R. Hoare, and A.P. Ravn. A calculus of durations. *Information Processing Letters*, 40(5):269–276, 1991.

[CM04] Séverine Colin and Leonardo Mariani. Run-time verification. In *Model-Based Testing of Reactive Systems*, pages 525–555, 2004.

[Col08] Christian Colombo. Practical runtime monitoring with impact guarantees of java programs with real-time constraints. Master's thesis, University of Malta, 2008.

[CPS] C. Colombo, G. Pace, and G. Schneider. Safe runtime verification of real-time properties. *Submitted for publication.*

[CPS08] C. Colombo, G. J. Pace, and G. Schneider. Dynamic event-based runtime monitoring of real-time and contextual properties. In *FMICS'08*. To appear in LNCS, 2008.

[KIL+97] Gregor Kiczales, John Irwin, John Lamping, Jean-Marc Loingtier, Cristina Videira Lopes, Chris Maeda, and Anurag Mendhekar. Aspect-oriented programming, 1997.

[Tho99] Simon Thompson. *Haskell: The Craft of Functional Programming (2nd Edition)*. Addison Wesley, March 1999.