Slotted-Circus: A Generic UTP framework for discretely-timed *Circus*

Andrew Butterfield
Paweł Gancarski

28 July 2009
(revised October 2009)

Contents

1 Introduction **8**

2 Syntax **9**

3 Slotted-*Circus* (Formal Definition) **10**

3.1 Observational Variables . . . 10
3.2 Required Definitions . . . 10
3.2.1 Accepted and Refused Events and Equivalent Traces . . . 11
3.2.2 Null Slots . . . 12
3.2.3 Slot Prefix Relation . . . 12
3.2.4 Slot Addition (Concatenation) . . . 13
3.2.5 Slot Subtraction . . . 13
3.2.6 Relating Addition and Subtraction . . . 14
3.2.7 Hiding Slot Events . . . 15
3.2.8 Slot Synchronisation . . . 15
3.2.9 Parameter Summary . . . 16
3.3 Derived Definitions . . . 17
3.3.1 Trace Equivalence of a Slot-Sequence . . . 18
3.3.2 Extracting Refusal Sequences . . . 18
3.3.3 Slot-Sequence Prefix Ordering ($\preccurlyeq$) . . . 19
3.3.4 Slot Equivalences . . . 20
3.3.5 Slot-Sequence Addition . . . 21
3.3.6 Slot-Sequence Subtraction . . . 21
3.3.7 Relating Slot-Sequence Addition and Subtraction . . . 22
3.4 Healthiness Conditions . . . 23
3.4.1 Reactive Healthiness 1 (**R1**) . . . 23

3.4.2 Reactive Healthiness 2 (**R2**) 23
3.4.3 Reactive Healthiness 3 (**R3**) 24
3.4.4 Reactive Healthiness (**R**) 25
3.4.5 CSP Healthiness 1 (**CSP1**) 27
3.4.6 CSP Healthiness 2 (**CSP2**) 27
3.4.7 CSP Healthiness 3 (**CSP3**) 28
3.4.8 CSP Healthiness 4 (**CSP4**) 28
3.4.9 CSP Healthiness 5 (**CSP5**) 29
3.4.10 Healthy Processes 29
3.5 Slotted-*Circus* Specific Actions 30
3.5.1 Laws 31
3.6 Actions 32
3.6.1 Nondeterministic Choice 32
3.6.2 Conditional Choice 32
3.6.3 Sequential Composition 32
3.6.4 Chaos 32
3.6.5 Deadlock 32
3.6.6 Guard 32
3.6.7 Termination 32
3.6.8 Delay 33
3.6.9 Assignment 33
3.6.10 Communication (Prefix) 33
3.6.11 External Choice 33
3.6.12 Parallel Composition 34
3.6.13 Hiding 35
3.6.14 Timeout 35
3.6.15 Recursion 35
3.7 Laws 35
3.7.1 Prefix 35
3.7.2 Sequential Composition 36
3.7.3 Conditional 36
3.7.4 Guards 36
3.7.5 Non-deterministic Choice 37
3.7.6 External Choice Composition 37
3.7.7 Parallel Composition 37
3.7.8 Hiding 38

4 Slotted-*Circus*—CTA Incarnation **40**
4.1 Observational Variables 40
4.2 Required Definitions and Proofs 40

4.2.1 Defining $acc_{\mathcal{CTA}}$ . . . 40
4.2.2 Defining $EQVTRC_{\mathcal{CTA}}$ . . . 40
4.2.3 Defining $hnull_{\mathcal{CTA}}$ . . . 40
4.2.4 Defining $\preceq_{\mathcal{CTA}}$ . . . 41
4.2.5 Defining $sadd_{\mathcal{CTA}}$ . . . 43
4.2.6 Defining $ssub_{\mathcal{CTA}}$ . . . 44
4.2.7 Defining $ssync_{\mathcal{CTA}}$ . . . 47
4.2.8 Defining $shide_{\mathcal{CTA}}$ . . . 48

5 Slotted-*Circus*—MSA Incarnation 49

5.1 Observational Variables . . . 49
5.2 Required Definitions and Proofs . . . 49
5.2.1 Defining $acc_{\mathcal{MSA}}$ . . . 49
5.2.2 Defining $EQVTRC_{\mathcal{MSA}}$ . . . 49
5.2.3 Defining $hnull_{\mathcal{MSA}}$ . . . 49
5.2.4 Defining $\preceq_{\mathcal{MSA}}$ . . . 50
5.2.5 Defining $sadd_{\mathcal{MSA}}$ . . . 52
5.2.6 Defining $ssub_{\mathcal{MSA}}$ . . . 54
5.2.7 Defining $ssync_{\mathcal{MSA}}$ . . . 57
5.2.8 Defining $shide_{\mathcal{MSA}}$ . . . 57

6 Slotted-*Circus*—SA Incarnation 58

6.1 Observational Variables . . . 58
6.2 Required Definitions and Proofs . . . 58
6.2.1 Defining $acc_{\mathcal{SA}}$ . . . 58
6.2.2 Defining $EQVTRC_{\mathcal{SA}}$ . . . 58
6.2.3 Defining $hnull_{\mathcal{SA}}$ . . . 58
6.2.4 Defining $\preceq_{\mathcal{SA}}$ . . . 59
6.2.5 Defining $sadd_{\mathcal{SA}}$ . . . 61
6.2.6 Defining $ssub_{\mathcal{SA}}$ . . . 62
6.2.7 Defining $ssync_{\mathcal{SA}}$ . . . 65
6.2.8 Defining $shide_{\mathcal{SA}}$ . . . 66

A Slotted-*Circus* Foundation Proofs 67

A.1 Proofs for Derived Definitions . . . 67
A.1.1 Proof . . . 67
A.1.2 Proof . . . 68
A.1.3 Proof . . . 72
A.1.4 Proof . . . 75
A.1.5 Proof . . . 77
A.1.6 Proof . . . 78

A.1.7 Proof 79
A.1.8 Proof 82
A.1.9 Proof 83
A.1.10 Proof 83
A.1.11 Proof 84
A.1.12 Proof 86
A.1.13 Proof 87
A.1.14 Proof 88
A.1.15 Proof 89
A.1.16 Proof 90
A.2 Useful Sequence Shorthands 91
A.2.1 Proof 92
A.2.2 Proof 95
A.2.3 Proof 96
A.2.4 Proof 97
A.2.5 Proof 98
A.2.6 Proof 99
A.2.7 Proof 100
A.2.8 Proof 101
A.2.9 Proof 102
A.2.10 Proof 103
A.2.11 Proof 104
A.2.12 Proof 105
A.2.13 Proof 108
A.2.14 Proof 109
A.2.15 Proof 111
A.2.16 Proof 112
A.3 Proofs for Healthiness Conditions 113
A.3.1 Proof 113
A.3.2 Proof 114
A.3.3 Proof 115
A.3.4 Proof 115
A.3.5 Proof 116
A.3.6 Proof 117
A.3.7 Proof 118
A.3.8 Proof 119
A.3.9 Proof 120
A.3.10 Proof 121
A.3.11 Proof 121
A.3.12 Proof 122

A.3.13 Proof 123
A.3.14 Proof 124
A.3.15 Proof 125
A.3.16 Proof 127
A.3.17 Proof 129
A.3.18 Proof 132
A.3.19 Proof 133
A.3.20 Proof 134
A.3.21 Proof 135
A.3.22 Proof 136
A.3.23 Proof 137
A.3.24 Proof 138
A.3.25 Proof 139
A.3.26 Proof 139
A.3.27 Proof 140
A.3.28 Proof 141
A.3.29 Proof 142
A.3.30 Proof 143
A.3.31 Proof 144
A.3.32 Proof 144
A.3.33 Proof 145
A.3.34 Proof 145
A.3.35 Proof 146
A.3.36 Proof 147
A.3.37 Proof 148
A.3.38 Proof 150
A.3.39 Proof 153
A.3.40 Proof 154
A.3.41 Proof 155
A.3.42 Proof 156
A.3.43 Proof 158
A.3.44 Proof 159
A.3.45 Proof 159
A.3.46 Proof 160
A.3.47 Proof 160
A.3.48 Proof 161
A.3.49 Lemma 162
A.3.50 Proof 163
A.3.51 Proof 164
A.3.52 Proof 165

A.3.53 Proof 166
A.3.54 Proof 167
A.3.55 Proof 168
A.3.56 Proof 169
A.3.57 Proof 170
A.3.58 Proof 171
A.3.59 Proof 172
A.3.60 Proof 173
A.4 Slotted-*Circus* Specific Actions 173
A.4.1 Proof 173
A.4.2 Proof 174
A.4.3 Proof 175

B Slotted-*Circus* Law Proofs **176**
B.1 Prefix 176
B.1.1 Proof 176
B.1.2 Proof 177
B.1.3 Proof 178
B.1.4 Lemma 179
B.1.5 Proof 180
B.1.6 Lemma 185
B.2 Sequential Composition 185
B.2.1 Proof 185
B.2.2 Lemma 187
B.2.3 Proof 188
B.2.4 Proof 189

C Synchronous CSP is not Slotted-*Circus* **193**
C.0.5 Accepted and Refused Events and Equivalent Traces 193
C.0.6 Null Slots 193
C.0.7 Slot Prefix Relation 193
C.0.8 Slot Addition (Concatenation) 194
C.0.9 Slot Subtraction 196
C.0.10 Relating Addition and Subtraction 197
C.0.11 Hiding Slot Events 197
C.0.12 Slot Synchronisation 197

D Slotted-*Circus* Proof Principles **199**
D.1 Proof Principles 199
D.1.1 Generalised One-point 199
D.1.2 Change of Variable 201

D.1.3 Variable Change: $\preccurlyeq$ . . . 202
D.1.4 Variable Change: $\cong$ . . . 203
D.1.5 Proof of [Hsub:equal] . . . 205
D.1.6 Proof of [SSub:equal] . . . 206
D.1.7 Shifting . . . 207

E Theory of Sequences **211**
E.1 Sequence Definitions . . . 211
E.1.1 Sequence Head . . . 211
E.1.2 Sequence Tail . . . 211
E.1.3 Sequence Concatenation . . . 211
E.1.4 Sequence Length . . . 211
E.1.5 Sequence Index . . . 211
E.1.6 Prefix Relation . . . 212
E.1.7 Sequence Subtraction . . . 212
E.1.8 Strict Prefix Relation . . . 212
E.1.9 Sequence Front . . . 212
E.1.10 Sequence Last . . . 213
E.2 Sequence Properties . . . 214
E.2.1 Proof . . . 214
E.2.2 Proof . . . 214
E.2.3 Proof . . . 214
E.2.4 Proof . . . 214
E.2.5 Proof . . . 214
E.2.6 Proof . . . 214
E.2.7 Proof . . . 214
E.2.8 Proof . . . 215
E.2.9 Proof . . . 215
E.2.10 Proof . . . 215
E.2.11 Proof . . . 215
E.2.12 Proof . . . 215
E.2.13 Proof . . . 215
E.2.14 Proof . . . 215

1 Introduction

This technical report describes the development of the theory of *Slotted-Circus*, a generic, discretely timed version of *Circus*[CSW03], inspired by the work of Sherif and He on *Circus* Timed Actions (CTA) [SH03, She06].

At present this is a holding version, with details that need to be integrated from the latest work (to be reported at FM2009).

2 Syntax

We present the syntax of all three varieties below, with an informal description of the behaviour of the constructs.

$$
\begin{array}{rcl}
\mathsf{Action} & ::= & \mathit{Skip} \mid \mathit{Stop} \mid \mathit{Chaos} \\
& \mid & \mathsf{Name}^{+} := \mathsf{Expr}^{+} \mid \mathsf{Comm} \rightarrow \mathsf{Action} \mid \mathsf{b}\ \&\ \mathsf{Action} \\
& \mid & \mathsf{Action};\ \mathsf{Action} \mid \mathsf{Action} \sqcap \mathsf{Action} \mid \mathsf{Action} \Box \mathsf{Action} \\
& \mid & \mathsf{Action} \lhd \mathsf{b} \rhd \mathsf{Action} \mid \mathsf{Action} \,[\![\mathsf{CS}]\!]\, \mathsf{Action} \mid \mathsf{Action} \setminus \mathsf{CS} \\
& \mid & \mu\,\mathsf{Name} \bullet F(\mathsf{Name}) \mid \mathit{Wait}\ \mathsf{t} \mid \mathsf{Action} \overset{\mathsf{t}}{\rhd} \mathsf{Action} \\
\mathsf{Comm} & ::= & \mathsf{Name} \mid \mathsf{Name.Expr} \mid \mathsf{Name!Expr} \mid \mathsf{Name?Name} \mid \mathsf{Name} : \mathsf{T} \\
\mathsf{T} & ::= & \text{type} \\
\mathsf{Expr} & ::= & \text{expression} \\
\mathsf{t} & ::= & \text{positive integer valued expression} \\
\mathsf{b} & ::= & \text{boolean valued expression} \\
\mathsf{Name} & ::= & \text{channel or variable names} \\
\mathsf{CS} & ::= & \text{channel name sets}
\end{array}
$$

The notation X^{+} denotes a sequence of X. We assume an appropriate syntax for describing expressions and their types, subject only to the proviso that booleans and non-negative integers are included.

The basic actions *Skip*, *Stop*, *Chaos* are similar to the corresponding CSP behaviours [Hoa85, Sch00], respectively denoting actions that do nothing and terminate, do nothing and wait forever, or act unpredictably forever. The composite actions operators ; , $\sqcap$ and $\Box$ denote sequential composition, internal choice and external choice respectively. Also familiar from CSP are the conditional action ($\lhd \mathsf{b} \rhd$) and the prefix action $\mathsf{Comm} \rightarrow \mathsf{Action}$, as well as recursively defined actions ($\mu\,\mathsf{Name} \bullet F(\mathsf{Name})$). The communication prefixes range over synchronisation on a channel (Name), communicating a value on a channel (Name.Expr), sending a value on a channel (Name!Expr), receiving a value on a channel (Name?Name) and being willing to engage in one of a number of possible events (Name : T).

We refer to *Circus* behaviour as "actions", rather than "processes", because they have assignable variables. So we find (multiple) assignment (:=) as a basic action. We also have actions guarded by boolean expressions (&).

Parallel composition of actions ($[\![\mathsf{CS}]\!]$) is parameterised by a single set, composed of channel names, rather than general events. Similarly, we can hide uses of channel names ($\setminus$ CS) rather than general events.

The constructs related to time are *Wait* t and $\overset{\mathsf{t}}{\rhd}$. The first (*Wait* t) denotes an action which simply waits for t time-slots to elapse, and then terminates. The second, $A \overset{t}{\rhd} B$ denotes a timeout situation where A is interrupted by B after t time-slots, if it has not performed an observable event or terminated by then. Both these two constructs are defined for *syncCircus* and CT^{*}, but not for *Circus*.

The syntax described here differs from that presented for CT^{*} in [SJ02] by the presence of the communication form Name : T, denoting a synchronisation event with no attendant data being transferred. This is provided in order to admit a uniform semantics for all communications.

3 Slotted-*Circus* (Formal Definition)

This section presents the formal definition of the slotted-*Circus* theory framework. We shall adopt the following convention for identifiers:

observation variables	lower-case
expression functions	lower-case
un-healthy predicates	ALL CAPS
healthy predicates	First Letter Capitalised

Note that relation identifiers (as distinct from symbols) are treated as predicates in the above convention.

3.1 Observational Variables

A slotted trace is defined over *Event*s, via a type constructor $\mathcal{S}$, which captures the variety of ways in which a slot can be built from events. In effect a given slotted theory is parametric in $\mathcal{S}$, and *Event*, and a number of relations to be defined below (in 3.2).

We have the following observational variables ($obs_{\mathcal{S}}$):

Stability $ok : \mathbb{B}$

Termination $wait : \mathbb{B}$

Variable-State $state : \mathsf{Name} \rightarrow Value$

Slot-Sequences $slots : (\mathcal{S}\ Event)^{+}$

3.2 Required Definitions

In general the predicates/relations/functions are parametric over $\mathcal{S}$, so are written as $NAME_{\mathcal{S}}$. However, in the sequel, we only indicate this parameter when giving the signature, and omit it from all other uses.

We present the signature of the required relations that characterise a particular slotted-theory, as well as specifying laws that they must obey.

In particular, we expect the slot structure to be a pair of the form $\mathcal{H}\,E \times \mathbb{P}\,E$, where the first component records some form of slot history based on events, while the second gives the set of events (ref) refused in that slot after that history has occurred.

$$[\mathsf{SLOT{:}structure}] \qquad slot, (hist, ref) \in \mathcal{S}\,E \mathrel{\widehat=} \mathcal{H}\,E \times \mathbb{P}\,E$$

So, in effect, a slotted theory is fact parametric on $\mathcal{H}$, rather than $\mathcal{S}$, this latter now being defined in terms of the former.

In the sequel, many operators defined on histories can be extended to slots in an "obvious" way. Typically, we overload these operator names, occasionally using suffixes to distinguish them. These "obvious" extensions conform to the following general scheme, where q and m are arbitrary query and modify operators respectively:

[Overload:Sig:q:H]	$q_{\mathcal{H}} : \mathcal{H}\, E \to X$
[Overload:Sig:q:S]	$q_{\mathcal{S}} : \mathcal{S}\, E \to X$
[Overload:q:def]	$q_{\mathcal{S}} = Q_{\mathcal{H}} \circ \pi_1$
[Overload:Sig:m:H]	$m_{\mathcal{H}} : X \to \mathcal{H}\, E \to \mathcal{H}\, E$
[Overload:Sig:m:S]	$m_{\mathcal{S}} : X \to \mathcal{S}\, E \to \mathcal{S}\, E$
[Overload:m:def]	$m_{\mathcal{S}}(x)(hist, ref) \mathrel{\widehat=} (m_{\mathcal{H}}(x)hist,\ ref)$

We will require many history operators to satisfy key laws, which are flagged in the sequel with ✠. In most cases, corresponding laws for slot operators will be an easy consequence of the history ones.

3.2.1 Accepted and Refused Events and Equivalent Traces

With a slot we associate the set of events accepted (acc) as well as the possible trace equivalents ($EQVTRC$).

[ACC:sig]		$acc_{\mathcal{S}} : \mathcal{H}\ E \to \mathbb{P}\, E$
		$acc_{\mathcal{H}} : \mathcal{S}\ E \to \mathbb{P}\, E$
[ET:sig]		$EQVTRC_{\mathcal{H}} : E^* \leftrightarrow \mathcal{H}\ E$
		$EQVTRC_{\mathcal{S}} : E^* \leftrightarrow \mathcal{S}\ E$
[ET:elems]	✠	$EQVTRC(tr, hist) \Rightarrow elems(tr) = acc(hist)$
[HIST:exists]	✠	$\exists\, hist \bullet acc(hist) = S$

We need also to be able to associate a refusal-set with an individual slot. However, unlike traces, we do expect the relationship to be functional—every slot has a unique refusal set associated with it, which is why we require the slot to be isomorphic to a pair containing a refusal set:

[REF:sig]	$sref_{\mathcal{S}} : \mathcal{S}\ E \to \mathbb{P}\, E$
[sref:def]	$sref \mathrel{\widehat=} \pi_2$

We also expect that histories and refusals are all that is required to define a slot:

[HIST:eq] ✠ $(h_1 = h_2)$

$$\equiv\ \forall\, tr \bullet EQVTRC(tr, h_1) \equiv EQVTRC(tr, h_2)$$

A corresponding principle for slots is easy to derive:

[SLOT:eq] $(s_1 = s_2)$

$$\equiv\ sref(s_1) = sref(s_2) \wedge \forall\, tr \bullet EQVTRC(tr, s_1) \equiv EQVTRC(tr, s_2)$$

[Acc:h:eq:elems:ET] $acc(t) = S \equiv \forall\, r\, \exists\, tt \bullet EQVTRC(tt, (t, r)) \wedge elems(tt) = S$

3.2.2 Null Slots

We require the notion of a null slot with no accepted events, but capable of supporting arbitrary refusals. We are led to posit that the only aspect which varies in any null-slot is precisely those refusals. This amounts to the existence of a unique null history value

[HN:sig]		$hnull_{\mathcal{H}} : \mathcal{H}\ E$
[HN:null]	✠	$acc(hnull) = \{\}$
[SN:sig]		$snull_{\mathcal{S}} : \mathbb{P}\ E \rightarrow \mathcal{S}\ E$
[SN:def]		$snull(ref) \mathrel{\widehat=} (hnull, ref)$

Note that $snull$ is a bijection if viewed as a function from $\mathbb{P}\ E$ to $\mathcal{S}_{Null}\ E$, where $\mathcal{S}_{Null}\ E$ is the space of null-slots. The following laws are immediate:

[SN:ref]	$sref(snull(ref)) = ref$
[SN:null]	$acc(snull(ref)) = \{\}$
[SN:eq]	$snull(r) = snull(r') \equiv r = r'$

3.2.3 Slot Prefix Relation

The relation $\preceq_{\mathcal{S}}$ captures the notion of one slot being a prefix of another:

[pfx:sig]		$\preceq_{\mathcal{H}} : \mathcal{H}\ E \leftrightarrow \mathcal{H}\ E$
		$\preceq_{\mathcal{S}} : \mathcal{S}\ E \leftrightarrow \mathcal{S}\ E$
[pfx:def]		$(hist_1, ref_1) \preceq_{\mathcal{S}} (hist_2, ref_2) \mathrel{\widehat=} hist_1 \preceq_{\mathcal{H}} hist_2$
[pfx:refl]	✠	$hist \preceq hist = \textsc{True}$
		$slot \preceq slot = \textsc{True}$
[pfx:trans]	✠	$hist_1 \preceq hist_2 \land hist_2 \preceq hist_3 \Rightarrow hist_1 \preceq hist_3$
		$slot_1 \preceq slot_2 \land slot_2 \preceq slot_3 \Rightarrow slot_1 \preceq slot_3$
[pfx:anti-sym]	✠	$hist_1 \preceq hist_2 \land hist_2 \preceq hist_1 \Rightarrow hist_1 = hist_2$

It asserts that its first history argument is, in some sense, a "prefix" of its second, and must also be a pre-order. In fact over the history component it must be a partial order, but cannot be so over the whole slot structure, as it ignores the refusal component. We require that a null history is a prefix of any history, and also, if one history is a prefix of another, then there must exist corresponding equivalent traces which are in the sequence-prefix relation:

[SN:pfx]	✠	$hnull \preceq hist$
		$snull(r) \preceq slot$
[ET:pfx]	✠	$hist_1 \preceq hist_2 \Rightarrow \exists\, tr_1, tr_2 \bullet EQVTRC(tr_1, hist_1) \land EQVTRC(tr_2, hist_2) \land tr_1 \leq tr_2$
		$slot_1 \preceq slot_2 \Rightarrow \exists\, tr_1, tr_2 \bullet EQVTRC(tr_1, s_1) \land EQVTRC(tr_2, s_2) \land tr_1 \leq tr_2$

An important thing to note however, is that slots whose equivalent traces are identical can be mutual prefixes of each other, even if their associated refusals differ—in other words the question of whether or not one slot is a prefix of another does not take refusals into consideration.

[pfx:ignores:ref:1]	$slot_1 \preceq slot_2 \land slot_2 \preceq slot_1 \not\Rightarrow sref(slot_1) = sref(slot_2)$
[pfx:ignores:ref:2]	$slot_1 \preceq slot_2 \equiv \forall\, r_1, r_2 \bullet slot_1[r_1] \preceq slot_2[r_2]$

Here we use the notation $s[r]$ to denote a slot s where its refusal component has been replaced by r. These properties are an immediate consequence of the pair-structure of slots and the definition of prefixing.

3.2.4 Slot Addition (Concatenation)

We also need to have the notion of adding slots in a manner analogous to concatenation:

[sadd:sig] $sadd_{\mathcal{H}} : \mathcal{H}\ E \times \mathcal{H}\ E \to \mathcal{H}\ E$
$sadd_{\mathcal{S}} : \mathcal{S}\ E \times \mathcal{S}\ E \to \mathcal{S}\ E$

We define the slot version in terms of the history version as retaining the refusals of the latter argument:

[sadd:def] $sadd_{\mathcal{S}}((h_1, r_1), (h_2, r_2)) \mathrel{\widehat=} (sadd_{\mathcal{H}}(h_1, h_2), r_2)$

We require behaviour as below, requiring slot addition to be associative (but not generally commutative, as it is more a form of adding by concatenation):

[sadd:events] ✠ $acc(sadd(h_1, h_2)) = acc(h_1) \cup acc(h_2)$
$acc(sadd(s_1, s_2)) = acc(s_1) \cup acc(s_2)$
[sadd:ref] $sref(sadd(s_1, s_2)) = sref(s_2)$
[sadd:unit] ✠ $sadd(h_1, h_2) = h_1 \equiv h_2 = hnull$
✠ $sadd(h_1, h_2) = h_2 \equiv h_1 = Hnull$
$sadd(s_1, s_2) = s_1 \equiv s_2 = snull(sref(s_1))$
$sadd(s_1, s_2) = s_2 \equiv \exists\, r_2 \bullet s_2 = snull(r_2)$
[sadd:assoc] ✠ $sadd(h_1, sadd(h_2, h_3)) = sadd(sadd(h_1, h_2), h_3)$
$sadd(s_1, sadd(s_2, s_3)) = sadd(sadd(s_1, s_2), s_3)$
[sadd:prefix] ✠ $h \preceq sadd(h, h')$
$s \preceq sadd(s, s')$

There is a derived notion of slot equivalence ($\approx$), introduced formally later (3.3.4), and we require *sadd* to have the following property w.r.t. to such an equivalence:

[sadd:eqv:unit] $sadd(s_1, s_2) \approx s_1 \ \equiv\ \exists\, r_2 \bullet s_2 = snull(r_2)$

Note that this is weaker than the unit law [sadd:unit], but is a consequence of the history version of it.

Finally, we introduce the following binary shorthand for *sadd* :

[sadd:binop] $s_1 \sharp s_2 \mathrel{\widehat=} sadd(s_1, s_2)$

3.2.5 Slot Subtraction

The related notion of slot-subtraction is also required:

[ssub:sig] $ssub_{\mathcal{H}} : \mathcal{H}\ E \times \mathcal{H}\ E \nrightarrow \mathcal{H}\ E$
$ssub_{\mathcal{S}} : \mathcal{S}\ E \times \mathcal{S}\ E \nrightarrow \mathcal{S}\ E$

where the second argument is subtracted from the first.

As for addition, we define slot subtraction to retain the refusal of its first argument:

[ssub:def] $ssub_{\mathcal{S}}((h_1, r_1), (h_2, r_2)) \mathrel{\widehat=} (ssub_{\mathcal{H}}(h_1, h_2), r_1)$

Subtraction is partial, and needs to obey a large collection of laws:

[ssub:pre] ✠ $\text{pre } ssub(h_1, h_2) = h_2 \preceq h_1$
$\text{pre } ssub(s_1, s_2) = s_2 \preceq s_1$

[ssub:events] ✠ $h_2 \preceq h_1 \land h' = ssub(h_1, h_2) \Rightarrow acc(h_1) \setminus acc(h_2) \subseteq acc(h') \subseteq acc(h_1)$

[ssub:events]$_{\mathcal{S}}$ $s_2 \preceq s_1 \land s' = ssub(s_1, s_2) \Rightarrow acc(s_1) \setminus acc(s_2) \subseteq acc(s') \subseteq acc(s_1)$

[SSub:ref] $sref(ssub(slot', slot)) = sref(slot')$

[SSub:self] ✠ $ssub(h, h) = hnull$
$ssub(s, s) = snull(sref(s))$

[SSub:nil] ✠ $ssub(h, hnull) = h$
$ssub(s, snull(r)) = s$

[SSub:same] ✠ $hist \preceq hist'_a \land hist \preceq hist'_b \Rightarrow ssub(hist'_a, hist) = ssub(hist'_b, hist) \equiv hist'_a = hist'_b$

[SSub:same]$_{\mathcal{S}}$ $slot \preceq slot'_a \land slot \preceq slot'_b \Rightarrow ssub(slot'_a, slot) = ssub(slot'_b, slot) \equiv slot'_a = slot'_b$

[SSub:subsub] ✠ $hist_c \preceq hist_a \land hist_c \preceq hist_b \land hist_b \preceq hist_a \Rightarrow ssub(ssub(hist_a, hist_c), ssub(hist_b, hist_c)) = ssub(hist_a, hist_b)$

[SSub:subsub]$_{\mathcal{S}}$ $slot_c \preceq slot_a \land slot_c \preceq slot_b \land slot_b \preceq slot_a \Rightarrow ssub(ssub(slot_a, slot_c), ssub(slot_b, slot_c)) = ssub(slot_a, slot_b)$

The law [ssub:events] may seem a little weak, but in general subtracting s_2 from s_1 does not guarantee that the result will not mention events in s_2. As for *sadd*, we need a property linking *ssub* and slot equivalence (3.3.4):

[SSub:eqv] $s_1 \approx s_2 \equiv ssub(s_1, s_2) = snull(sref(s_1))$

This law is a consequence of the anti-symmetric of $\preceq_{\mathcal{H}}$, and the laws [SSub:self] and [ssub:def].

Also useful is a law allowing us to handle slot-subtractions that cause no change:

[SSub:equal] $s = ssub(s, sn) \equiv \exists\, rn \bullet sn = snull(rn)$

For proof see [SSub:equal]:p206

Finally, we introduce the following binary shorthand for *ssub* :

[sadd:binop] $s_1 \setminus s_2 \mathrel{\widehat=} ssub(s_1, s_2)$

3.2.6 Relating Addition and Subtraction

We also require addition and subtraction to satisfy the following laws, the first of which can be considered a defining feature of subtraction, and the second being required to ensure that **R2** (see

[R2:def]:p23) is idempotent:

$$
\begin{array}{lll}
[\mathsf{sadd:ssub}] & \maltese & hist \preceq hist' \Rightarrow sadd(hist, ssub(hist', hist)) = hist' \\
 & & slot \preceq slot' \Rightarrow sadd(slot, ssub(slot', slot)) = slot' \\
[\mathsf{ssub:sadd}] & \maltese & ssub(sadd(h_1, h_2), h_1) = h_2 \\
 & & ssub(sadd(s_1, s_2), s_1) = s_2
\end{array}
$$

We will allow certain variants of slotted-*Circus* that fail to satisfy [ssub:sadd], provided a different form of the **R2** healthiness condition is used. An example of this is that case where we model the event occurrences as a set, and *sadd* and *ssub* correspond to set union and set difference respectively. In this case it is generally the case that:

$$(S_1 \cup S_2) \setminus S_1 \neq S_2 \qquad \text{e.g.: } (\{a\} \cup \{a\}) \setminus \{a\} = \emptyset \neq \{a\}.$$

3.2.7 Hiding Slot Events

We need to specify how to hide events in a slot:

$$
\begin{array}{ll}
[\mathsf{SHid:sig}] & shide_{\mathcal{H}} : \mathbb{P}\, E \rightarrow \mathcal{H}\ E \rightarrow \mathcal{H}\ E \\
 & shide_{\mathcal{S}} : \mathbb{P}\, E \rightarrow \mathcal{S}\ E \rightarrow \mathcal{S}\ E
\end{array}
$$

Hiding shrinks the event-set:

$$
\begin{array}{rll}
[\mathsf{SHid:def}] & & shide_{\mathcal{S}}(hid)(hist, ref) \mathrel{\widehat{=}} (shide_{\mathcal{H}}(hid)hist, ref) \\
[\mathsf{SHid:evts}] & \maltese & acc(shide(hid)(h)) = acc(h) \setminus hid \\
 & & acc(shide(hid)(s)) = acc(s) \setminus hid \\
[\mathsf{SHid:refs}] & & sref(shide(hid)(s)) = sref(s) \\
[\mathsf{hide:it:is:null}] & & (\exists\, t \bullet acc(t) = \{c\} \wedge tt = SHide_H\{c\}(t)) \equiv acc(tt) = \emptyset
\end{array}
$$

3.2.8 Slot Synchronisation

Finally, we need a function that captures the way in which two slots can synchronise on a given channel-set:

$$
\begin{array}{ll}
[\mathsf{SNC:sig}] & ssync_{\mathcal{H}} : \mathbb{P}\, E \rightarrow \mathcal{H}\ E \times \mathcal{H}\ E \rightarrow \mathbb{P}(\mathcal{H}\ E) \\
 & ssync_{\mathcal{S}} : \mathbb{P}\, E \rightarrow \mathcal{S}\ E \times \mathcal{S}\ E \rightarrow \mathbb{P}(\mathcal{S}\ E)
\end{array}
$$

We define the slot-version in terms of the history one as follows:

$$
\begin{array}{rl}
[\mathsf{SNC:def}] & ssync_{\mathcal{S}}(cs)((hist_1, ref_1), (hist_2, ref_2)) \\
 & \mathrel{\widehat{=}} ssync_{\mathcal{H}}(cs)(hist_1, hist_2) \times \{rsync(cs)(ref_1, ref_2)\} \\
[\mathsf{RSYN:sig}] & rsync : \mathbb{P}\, E \rightarrow \mathbb{P}\, E \times \mathbb{P}\, E \rightarrow \mathbb{P}\, E \\
[\mathsf{RSYN:def}] & rsync(cs)(r_1, r_2) \mathrel{\widehat{=}} ((r_1 \cup r_2) \cap cs) \cup ((r_1 \cap r_2) \setminus cs) \\
[\mathsf{RSYN:sym}] & rsync(cs)(r_1, r_2) = rsync(cs)(r_2, r_1) \\
[\mathsf{RSYN:assoc}] & rsync(cs)(r_1, rsync(cs)(r_2, r_3) = rsync(cs)(rsync(cs)(r_1, r_2), r_3)
\end{array}
$$

Synchronisation needs to satisfy the following:

$$
\begin{array}{rcl}
\text{[SNC:sym]} & \maltese & ssync(cs)(h_1, h_2) = ssync(cs)(h_2, h_1) \\
 & & ssync(cs)(s_1, s_2) = ssync(cs)(s_2, s_1) \\
\text{[SNC:null]} & & ssync(cs)(snull(r_1), snull(r_2)) = \{snull(rsync(r_1, r_2))\} \\
\text{[SNC:one]} & \maltese & \forall h' \in ssync(cs)(h_1, hnull) \bullet acc(h') \subseteq acc(h_1) \setminus cs \\
 & & \forall r_2 \bullet \forall s' \in ssync(cs)(s_1, snull(r_2)) \bullet acc(s') \subseteq acc(s_1) \setminus cs \\
\text{[SNC:only]} & \maltese & h' \in acc(ssync(cs)(h_1, h_2)) \Rightarrow acc(h') \subseteq acc(h_1) \cup acc(h_2) \\
 & & s' \in acc(ssync(cs)(s_1, s_2)) \Rightarrow acc(s') \subseteq acc(s_1) \cup acc(s_2) \\
\text{[SNC:sync]} & \maltese & h' \in acc(ssync(cs)(h_1, h_2)) \Rightarrow cs \cap acc(h') \subseteq cs \cap (acc(h_1) \cap acc(h_2)) \\
 & & s' \in acc(ssync(cs)(s_1, s_2)) \Rightarrow cs \cap acc(s') \subseteq cs \cap (acc(s_1) \cap acc(s_2))
\end{array}
$$

Note that these laws are weaker than might be expected—in particular, they do not specify the difference between what happens to events common to both slots, *vis-a-vis* their membership of the synchronisation set. This aspect of behaviour depends on the specifics of a given slotted theory.

We would like an associativity principle, but in order to do that we need to handle sychronisation of one history against a set of same:

$$
\begin{array}{rcl}
\text{[SNCS:sig]} & & syncset : \mathbb{P}\, E \rightarrow \mathcal{H}\, E \rightarrow \mathbb{P}(\mathcal{H}\, E) \rightarrow \mathbb{P}(\mathcal{H}\, E) \\
\text{[SNCS:def]} & & syncset(cs)(h)(H) \mathrel{\widehat{=}} \bigcup\{ssync(cs)(h, h') \mid h' \in H\} \\
\text{[SNC:assoc]} & \maltese & syncset(cs)(h_1)(ssync(cs)(h_2, h_3)) = syncset(cs)(h_3)(ssync(cs)(h_1, h_2)) \\
 & & syncset(cs)(s_1)(ssync(cs)(s_2, s_3)) = syncset(cs)(s_3)(ssync(cs)(s_1, s_2))
\end{array}
$$

3.2.9 Parameter Summary

We recap: the parameters required to define a slotted-*Circus* theory, itself parametric on *Events*, are:

$$
\begin{array}{rcl}
\mathcal{H} & : & E \rightarrow \mathcal{H}\ E \\
acc_{\mathcal{H}} & : & \mathcal{H}\ E \rightarrow \mathbb{P}\, E \\
EQVTRC_{\mathcal{H}} & : & E^* \leftrightarrow \mathcal{H}\ E \\
hnull_{\mathcal{H}} & : & \mathcal{H}\ E \\
\preceq_{\mathcal{H}} & : & \mathcal{H}\ E \leftrightarrow \mathcal{H}\ E \\
ssub_{\mathcal{H}} & : & \mathcal{H}\ E \times \mathcal{H}\ E \rightarrow\!\!\!\!\!+ \mathcal{H}\ E \\
sadd_{\mathcal{H}} & : & \mathcal{H}\ E \times \mathcal{H}\ E \rightarrow \mathcal{H}\ E \\
shide_{\mathcal{H}} & : & \mathbb{P}\, E \rightarrow \mathcal{H}\ E \rightarrow \mathcal{H}\ E \\
ssync_{\mathcal{H}} & : & \mathbb{P}\, E \rightarrow \mathcal{H}\ E \times \mathcal{H}\ E \rightarrow \mathbb{P}(\mathcal{H}\ E)
\end{array}
$$

These need to satisfy the following laws:

[ET:elems] ✠ $EQVTRC(tr, hist) \Rightarrow elems(tr) = acc(hist)$
[HIST:exists] ✠ $\exists\, hist \bullet acc(hist) = S$
[HIST:eq] ✠ $(h_1 = h_2) \;\equiv\; \forall\, tr \bullet EQVTRC(tr, h_1) \equiv EQVTRC(tr, h_2)$
[HN:null] ✠ $acc(hnull) = \{\}$
[pfx:refl] ✠ $hist \preceq hist = \text{True}$
[pfx:trans] ✠ $hist_1 \preceq hist_2 \land hist_2 \preceq hist_3 \Rightarrow hist_1 \preceq hist_3$
[pfx:anti-sym] ✠ $hist_1 \preceq hist_2 \land hist_2 \preceq hist_1 \Rightarrow hist_1 = hist_2$
[SN:pfx] ✠ $hnull \preceq hist$
[ET:pfx] ✠ $hist_1 \preceq hist_2 \Rightarrow \exists\, tr_1, tr_2 \bullet EQVTRC(tr_1, hist_1) \land EQVTRC(tr_2, hist_2) \land tr_1 \leq tr_2$
(continued overleaf)

[sadd:events] ✠ $acc(sadd(h_1, h_2)) = acc(h_1) \cup acc(h_2)$
[sadd:unit:r] ✠ $sadd(h_1, h_2) = h_1 \equiv h_2 = hnull$
[sadd:unit:l] ✠ $sadd(h_1, h_2) = h_2 \equiv h_1 = hnull$
[sadd:assoc] ✠ $sadd(h_1, sadd(h_2, h_3)) = sadd(sadd(h_1, h_2), h_3)$
[sadd:prefix] ✠ $h \preceq sadd(h, h')$
[ssub:pre] ✠ pre $ssub(h_1, h_2) = h_2 \preceq h_1$
[ssub:events] ✠ $h_2 \preceq h_1 \land h' = ssub(h_1, h_2) \Rightarrow$
$acc(h_1) \setminus acc(h_2) \subseteq acc(h') \subseteq acc(h_1)$
[SSub:self] ✠ $ssub(h, h) = hnull$
[SSub:nil] ✠ $ssub(h, hnull) = h$
[SSub:same] ✠ $hist \preceq hist'_a \land hist \preceq hist'_b \Rightarrow$
$ssub(hist'_a, hist) = ssub(hist'_b, hist) \;\equiv\; hist'_a = hist'_b$
[SSub:subsub] ✠ $hist_c \preceq hist_a \land hist_c \preceq hist_b \land hist_b \preceq hist_a$
$\Rightarrow ssub(ssub(hist_a, hist_c), ssub(hist_b, hist_c)) = ssub(hist_a, hist_b)$
[sadd:ssub] ✠ $hist \preceq hist' \Rightarrow sadd(hist, ssub(hist', hist)) = hist'$
[ssub:sadd] ✠ $ssub(sadd(h_1, h_2), h_1) = h_2$
[SHid:evts] ✠ $acc(shide(hid)(h)) = acc(h) \setminus hid$
[SNC:sym] ✠ $ssync(cs)(h_1, h_2) = ssync(cs)(h_2, h_1)$
[SNC:one] ✠ $\forall\, h' \in ssync(cs)(h_1, hnull) \bullet acc(h') \subseteq acc(h_1) \setminus cs$
[SNC:only] ✠ $h' \in acc(ssync(cs)(h_1, h_2)) \Rightarrow acc(h') \subseteq acc(h_1) \cup acc(h_2)$
[SNC:sync] ✠ $h' \in acc(ssync(cs)(h_1, h_2)) \Rightarrow cs \cap acc(h') \subseteq cs \cap (acc(h_1) \cap acc(h_2))$
[SNC:assoc] ✠ $syncset(cs)(h_1)(ssync(cs)(h_2, h_3)) = syncset(cs)(h_3)(ssync(cs)(h_1, h_2))$

3.3 Derived Definitions

The following relations have general use throughout the theory:

3.3.1 Trace Equivalence of a Slot-Sequence

First, we define the notion of a trace equivalent to a slot-sequence:

[ETs:sig] $EQVTRACE_{\mathcal{S}} : E^* \leftrightarrow (\mathcal{S}\ E)^*$

[ETs:def:nil] $EQVTRACE(tr, \langle\rangle) \mathrel{\widehat=} tr = \langle\rangle$

[ETs:def:cons] $EQVTRACE(tr, slot \mathbin{⨾} slots)$
$\mathrel{\widehat=} \exists\, tr_0 \bullet tr_0 \leq tr \land EQVTRC(tr_0, slot) \land EQVTRACE(tr - tr_0, slots)$

This relationship links slotted-sequences back to the tr observation of *Circus*. We find that $EQVTRACE$ obeys the following laws:

[ETs:sngl] $EQVTRACE(tr, \langle slot\rangle) \mathrel{\widehat=} EQVTRC(tr, slot)$

[ETs:cat] $EQVTRACE(tr_a, slots_a) \land EQVTRACE(tr_b, slots_b)$
$\Rightarrow EQVTRACE(tr_a \frown tr_b, slots_a \frown slots_b)$

[ETs:elems] $EQVTRACE(tr, slots) \Rightarrow elems(tr) = \bigcup_{i \in 1}^{\#slots} \{acc(slots(i))\}$

[ETs:null] $EQVTRACE(\langle\rangle, slots) \equiv \forall\, i \in 1 \ldots \#slots \bullet EQVTRC(\langle\rangle, slots(i))$

Proofs: [ETs:sngl]:p67 , [ETs:cat]:p68 , [ETs:elems]:p72 , [ETs:null]:p75 .

Some slotted theories, define an auxiliary variable $trace'$ that captures the notion of which events have occurred since the current action began. This only makes sense if the $EQVTRACE$ relation is functional. If not, the best we can come up with is a predicate that defines possible values for $trace'$:

[PEV:sig] $POSSEVTS : E^* \leftrightarrow (\mathcal{S}\ E)^+ \times (\mathcal{S}\ E)^+$

[PEV:def] $POSSEVTS(trace', (slots, slots'))$
$\mathrel{\widehat=}\ \exists\, tr, tr' \bullet EQVTRACE(tr, slots) \land EQVTRACE(tr', slots') \land trace' = tr' - tr$

In fact, we make no further use of this in the sequel, instead relying on more specific relations to capture specific cases regarding the nature of $trace'$ (null, non-null, starting with a particular element, etc..).

3.3.2 Extracting Refusal Sequences

We can extract sequence of refusals from a slot-sequence:

[RFS:sig] $srefs_{\mathcal{S}} : (\mathcal{S}\ E)^+ \to (\mathbb{P}\, E)^+$

[RFS:def] $srefs(slots) = map(sref)(slots)$

We then specialise this to a relationship between a single refusal set and a slot-sequence, where the refusal corresponds to the last slot:

[ER:sig] $eqvrefs_{\mathcal{S}} : (\mathcal{S}\ E)^+ \to \mathbb{P}\, E$

[ER:def] $eqvref(slots) \mathrel{\widehat=} sref(last(slots))$

This is in fact the relationship used to link a slotted theory back to the ref observation of *Circus*.

3.3.3 Slot-Sequence Prefix Ordering ($\preccurlyeq$)

We define a prefix ordering ($\preccurlyeq$) on slot-sequences by noting that all but the last slot of *slots* is a prefix of *slots*′, and that the last slot of *slots* is a slot-prefix to the corresponding slot of *slots*′:

[EX:sig] $\preccurlyeq_{\mathcal{S}} : (\mathcal{S}\ E)^+ \leftrightarrow (\mathcal{S}\ E)^+$

[EX:def] $slots \preccurlyeq slots' \mathrel{\widehat=} front(slots) < slots' \wedge last(slots) \preceq slots'(\#slots)$

Note that we insist on strict prefixing between $front(slots)$ and *slots*, because the definition above is undefined if $front(slots) = slots'$, as then *slots* is in fact longer than *slots*′ by one element, so $slots'(\#slots)$ is undefined. This corrects a common error found in the literature of slotted-like theories (see **??**).

We also give an alternative definition in terms of a formulation that makes the last slot explicit (an exercise in sequence properties):

[EX:prior-hist-ref:def]
$$p \frown \langle (h,r) \rangle \preccurlyeq p' \frown \langle (h',r') \rangle$$
$$= \quad p \le p' \wedge (h,r) \preceq (p' \frown \langle (h',r') \rangle)(\#p+1)$$

The sequence-ordering relation ($\le$) on slot-sequences is a sub-relation of $\preccurlyeq$. We also note that $\preccurlyeq_{\mathcal{S}}$ is only a pre-order, inheriting this property from slot-prefixing. However, if $\preceq$ in a particular theory is anti-symmetric, then so is $\preccurlyeq$.

[EX:subseq] $slots_a \le slots_b \Rightarrow slots_a \preccurlyeq slots_b$

[EX:refl] $slots \preccurlyeq slots$

[EX:trans] $slots_a \preccurlyeq slots_b \wedge slots_b \preccurlyeq slots_c \Rightarrow slots_a \preccurlyeq slots_c$

[EX:anti] $(\forall slot_a, slot_b \bullet slot_a \preceq slot_b \wedge slot_b \preceq slot_a \Rightarrow slot_a = slot_b)$
$\Rightarrow$
$(slots_a \preccurlyeq slots_b \wedge slots_b \preccurlyeq slots_a \Rightarrow slots_a = slots_b)$

Proofs: [EX:subseq]:p77 , [EX:refl]:p78 , [EX:trans]:p79 , [EX:anti]:p82 . We also expect the (almost) empty slot-sequence to be a prefix of any other sequence:

[EX:null] $\langle snull(r) \rangle \preccurlyeq slots$

Proof: [EX:null]:p83 .

At this point we note that when $slots \preccurlyeq slots'$ holds, we can write *slots* and *slots*′ as follows:

$$slots = pfx \frown \langle slot \rangle$$
$$slots' = pfx \frown \langle slot' \rangle \frown sfx$$

[EX:pfx]

here *pfx* denotes their common prefix, while *slot* and *slot*′ are the first slot at which they differ, this always being the last slot of *slots*. Then *sfx* is the subsequent behaviour of *slots*′ after *slot*′. We note that both *pfx* and *sfx* can be empty lists, and that $pfx = front(slots)$.

In addition to the above, there are a few properties regarding slot-sequences with common sub-parts worth noting:

[EX:prefix] $ss_1 \frown ss_2 \preccurlyeq ss_1 \frown ss_3 \equiv ss_2 \preccurlyeq ss_3$

[EX:sngl] $\langle s_1 \rangle \preccurlyeq s_2 \mathbin{⨾} ss \equiv s_1 \preceq s_2$

Proofs: [EX:prefix]:p87 , [EX:sngl]:p88 . Finally, it is worth noting the following relationships between slot-prefixing and sequential composition[1]:

[EX;EX] $(slots \preccurlyeq slots');\ (slots \preccurlyeq slots') = (slots \preccurlyeq slots')$

Proof: [EX;EX]:p89.

3.3.4 Slot Equivalences

We define an equivalence on slots based on mutual prefixing:

[SEQV:sig] $\approx_{\mathcal{S}}\colon \mathcal{S}\ E \leftrightarrow \mathcal{S}\ E$

[SEQV:def] $slot_1 \approx slot_2 \mathrel{\widehat{=}} slot_1 \preceq slot_2 \land slot_2 \preceq slot_1$

A key property is that it amounts to equality of the history component:

[SEQV:equal-h] $(h_1, \_) \approx (h_2, \_) \equiv h_1 = h_2$

Proof is an elementary use of definitions and the property [pfx:anti-sym]:p17.
We then extend this slot-equivalence to slot-sequences:

[SSEQV:sig] $\cong_{\mathcal{S}}\colon (\mathcal{S}\ E)^+ \leftrightarrow (\mathcal{S}\ E)^+$

[SSEQV:def] $slots_1 \cong slots_2 \mathrel{\widehat{=}} slots_1 \preccurlyeq slots_2 \land slots_2 \preccurlyeq slots_1$

That $\approx$ and $\cong$ are equivalence relations is an immediate consequence of $\preceq$, and hence $\preccurlyeq$, being a pre-order (exercise):

[SEQV:refl] $slot \approx slot$

[SEQV:symm] $slot_1 \approx slot_2 \equiv slot_2 \approx slot_1$

[SEQV:trans] $slot_1 \approx slot_2 \land slot_2 \approx slot_3 \Rightarrow slot_1 \approx slot_3$

[SSEQV:refl] $slots \cong slots$

[SSEQV:symm] $slots_1 \cong slots_2 \equiv slots_2 \cong slots_1$

[SSEQV:trans] $slots_1 \cong slots_2 \land slots_2 \cong slots_3 \Rightarrow slots_1 \cong slots_3$

It is useful to have a direct expansion of $\cong$:

[SSEQV:expand]
$$slots_1 \cong slots_2 \\ = front(slots_1) = front(slots_2) \land last(slots_1) \approx last(slots_2)$$

Proof: [SSEQV:expand]:p90

We get the following relationships between $\preccurlyeq$, $\cong$ and sequential composition:

[EX;SSEQV] $(slots \preccurlyeq slots');\ (slots \cong slots') \equiv slots \preccurlyeq slots'$

[SSEQV;EX] $(slots \cong slots');\ (slots \preccurlyeq slots') \equiv slots \preccurlyeq slots'$

[SSEQV;SSEQV] $(slots \cong slots');\ (slots \cong slots') \equiv slots \cong slots'$

[1] This is an instance of the general law $GROW;\ GROW \equiv GROW$ introduced in the algebraic approach to reactive theories (work in progress)

Proofs—similar to that for [EX;EX]:p89.

We also expect the following interactions between $\approx$, *sadd* and *ssub*::

$$
\begin{array}{rl}
[\mathsf{sadd:eqv:unit}] : p13 & sadd(s_1, s_2) \approx s_1 \;\; \equiv? \; \exists\, r \bullet s_2 = snull(r) \\
[\mathsf{SSub:eqv}] : p14 & (\exists\, r \bullet ssub(s_1, s_2) = snull(r)) \;\; \equiv? \; s_1 \approx s_2
\end{array}
$$

These are listed as required properties of *sadd* and *ssub*.

3.3.5 Slot-Sequence Addition

We need to introduce the slot-analogue of sequence addition ($\sharp\!\sharp$) here, which merges the last slot of its first argument and the first slot of its second argument using $\sharp$:

$$
\begin{array}{ll}
[\mathsf{CAT:sig}] & \sharp\!\sharp_{\mathcal{S}} : ((\mathcal{S}\ E)^+ \times (\mathcal{S}\ E)^+) \rightarrow (\mathcal{S}\ E)^+ \\
[\mathsf{CAT:def}] & slots_1 \sharp\!\sharp\, slots_2 \mathrel{\widehat{=}} front(slots_1) \frown \langle last(slots_1) \sharp\, head(slots_2) \rangle \frown tail(slots_2)
\end{array}
$$

We find that $\sharp\!\sharp$ satisfies the following laws:

$$
\begin{array}{rl}
[\mathsf{CAT:assoc}] & sl_1 \sharp\!\sharp\, (sl_2 \sharp\!\sharp\, sl_3) = (sl_1 \sharp\!\sharp\, sl_2) \sharp\!\sharp\, sl_3 \\
[\mathsf{CAT:PFX}] & ss \preccurlyeq ss \sharp\!\sharp\, tt \\
[\mathsf{CAT:ER:last}] & eqvref(sl_1 \sharp\!\sharp\, sl_2) = eqvref(sl_2) \\
[\mathsf{CAT:eqv}] & sl_1 \cong sl_1 \sharp\!\sharp\, sl_2 \equiv \exists\, r_2 \bullet sl_2 = \langle snull(r_2) \rangle \\
[\mathsf{CAT:equal}] & sl_1 = sl_1 \sharp\!\sharp\, sl_2 \equiv sl_2 = \langle snull(eqvref(sl_1)) \rangle \\
[\mathsf{CAT:len}] & \#(sl_1 \sharp\!\sharp\, sl_2) = \#(sl_1) + \#(sl_2) - 1
\end{array}
$$

Proof: [CAT:assoc]:p92, [CAT:PFX]:p95, [CAT:ER:last]:p96, [CAT:eqv]:p97, [CAT:equal]:p98, [CAT:len]:p99.

3.3.6 Slot-Sequence Subtraction

We need to introduce the slot-analogue of sequence subtraction here:

$$
\begin{array}{rl}
[\mathsf{DF:sig}] & dif_{\mathcal{S}} : ((\mathcal{S}\ E)^+ \times (\mathcal{S}\ E)^+) \nrightarrow (\mathcal{S}\ E)^+ \\
[\mathsf{DF:pre}] & \text{pre } dif(slots', slots) = slots \preccurlyeq slots' \\
[\mathsf{DF:def}] & dif(slots', slots) \mathrel{\widehat{=}} ssub(slot', slot) \mathbin{⨾} sfx \\
\textbf{where} & slot = last(slots) \\
 & (slot' \mathbin{⨾} sfx) = slots' - front(slots) \\
[\mathsf{DF:pfx}] & dif(pfx \frown \langle slot' \rangle \frown sfx, pfx \frown \langle slot \rangle) = ssub(slot', slot) \mathbin{⨾} sfx
\end{array}
$$

We find that dif satisfies the following laws:

[DF:equal]	$slots = dif(slots, sln) \equiv \exists\, rn \bullet sln = \langle snull(rn) \rangle$
[DF:self]	$dif(slots, slots) = \langle snull(eqvref(slots)) \rangle$
[DF:nil]	$dif(slots, \langle snull(r) \rangle) = slots$
[DF:Null:equal]	$slots' \mathbin{\setminus\!\!\setminus} slots = \langle snull(r') \rangle \equiv slots' \cong slots \wedge eqvref(slots') = r'$
[DF:Null:eqv]	$(\exists\, r' \bullet slots' \mathbin{\setminus\!\!\setminus} slots \cong \langle snull(r') \rangle) \equiv slots' \cong slots$
[DF:same]	$slots_b \preccurlyeq slots_a \wedge slots_b \preccurlyeq slots_c \Rightarrow$ $dif(slots_a, slots_b) = dif(slots_c, slots_b) \equiv slots_a = slots_c$
[DF:all-null]	$EQVTRACE(\langle\rangle, dif(slots_a, slots_b)) \equiv$ $\forall\, i \in 1 \ldots \# dif(slots_a, slots_b) \bullet EQVTRC(\langle\rangle, (dif(slots_a, slots_b))(i))$
[DF:ref]	$srefs(dif(slots_a, slots_b)) = srefs(slots_a - front(slots_b))$
[DF:hd-Evt]	$EQVTRACE(\langle\rangle, dif(slots', slots)) \Rightarrow EQVTRC(\langle\rangle, head(dif(slots', slots)))$
[DF:subsub]	$slots_c \preccurlyeq slots_a \wedge slots_c \preccurlyeq slots_b \wedge slots_b \preccurlyeq slots_a$ $\Rightarrow dif(dif(slots_a, slots_c), dif(slots_b, slots_c)) = dif(slots_a, slots_b)$
[EX:dif]	$slots \preccurlyeq slots' \equiv \langle snull(r) \rangle \preccurlyeq dif(slots', slots)$
[DF:ER:first]	$eqvref(sl_1 \mathbin{\setminus\!\!\setminus} sl_2) = eqvref(sl_1)$
[DF:len]	$\#(sl_1 \mathbin{\setminus\!\!\setminus} sl_2) = (\#(sl_1) - \#(sl_2)) + 1$

Proofs: [DF:equal]:p100, [DF:self]:p101, [DF:nil]:p102, [DF:Null:equal]:p103, [DF:Null:eqv]:p104, [DF:same]:p105, [DF:all-null](see [ETs:null]:p18), [DF:ref]:p108, [DF:hd-Evt](see [ETs:null]:p18, instantiated with $i = 1$), [DF:subsub]:p109, [EX:dif]:p83, [DF:ER:first]:p84, [DF:len]:p86.

We introduce a binary notation:

[DF:binop] $\quad slots_1 \mathbin{\setminus\!\!\setminus} slots_2 \mathrel{\widehat=} dif(slots_1, slots_2)$

$$= \textbf{let}\ \ rest = slots_1 - front(slots_2)$$
$$\textbf{in}\ \ (head(rest) \setminus last(slots_2)) \mathbin{⨾} tail(rest)$$

3.3.7 Relating Slot-Sequence Addition and Subtraction

We expect slot-sequence subtraction and addition to obey the following laws:

[CAT:DF:id]	$(ss \sharp\, tt) \mathbin{\setminus\!\!\setminus} ss = tt$
[CAT:DF:pfx]	$(ss \sharp\, tt) \mathbin{\setminus\!\!\setminus} (ss \sharp\, uu) = tt \mathbin{\setminus\!\!\setminus} uu, \quad \textbf{if}\ uu \preccurlyeq tt$

Proof: [CAT:DF:id]:p111, [CAT:DF:pfx]:p112.

3.4 Healthiness Conditions

We are going to give analogues in slotted-*Circus* to the reactive and CSP healthiness conditions of UTP, namely **R** (**R1**,**R2**,**R3**) and **CSP1–5**.

3.4.1 Reactive Healthiness 1 (R1)

In *Circus* this asserts that $tr \leq tr'$. In a slotted theory, we use $\preccurlyeq$.

[GROW:def]	$GROW \mathrel{\widehat=} slots \preccurlyeq slots'$
[R1:def]	$\mathbf{R1}(P) \mathrel{\widehat=} P \land GROW$
[R1:idem]	$\mathbf{R1} \circ \mathbf{R1} = \mathbf{R1}$

Proofs: [R1:idem]:p113.

As **R1** has the form and_P, we immediately get the following distributive laws (exercises in elementary predicate calculus):

[R1:distr:and]	$\mathbf{R1}(P \land Q) = \mathbf{R1}(P) \land \mathbf{R1}(Q) = \mathbf{R1}(P) \land Q = P \land \mathbf{R1}(Q)$
[R1:distr:or]	$\mathbf{R1}(P \lor Q) = \mathbf{R1}(P) \lor \mathbf{R1}(Q)$
[R1:distr:all]	$\mathbf{R1}(\forall x \bullet P) = \forall x \bullet \mathbf{R1}(P), x \notin \{slots, slots'\}$
[R1:distr:any]	$\mathbf{R1}(\exists x \bullet P) = \exists x \bullet \mathbf{R1}(P), x \notin \{slots, slots'\}$
[R1:distr:cond]	$\mathbf{R1}(P \lhd c \rhd Q) = \mathbf{R1}(P) \lhd c \rhd \mathbf{R1}(Q)$

Distributivity of healthiness through sequential composition generally doesn't hold, but it should preserve healthiness:

[comp:R1:closed] $\quad (P \equiv \mathbf{R1}(P)) \land (Q \equiv \mathbf{R1}(Q)) \Rightarrow ((P;\ Q) \equiv \mathbf{R1}(P;\ Q))$

Proofs: [comp:R1:closed]:p147. A consequence of this is:

[GROW-GROW:eq:GROW] $\quad GROW;\ GROW \equiv GROW$

Sometimes it is worth exploring the consequences of being healthy/un-healthy in a little more detail:

$$
\begin{array}{lrcl}
 & (P \equiv \mathbf{R1}(P)) & \equiv & (P \equiv P \land GROW) \\
 & & \equiv & P \Rightarrow GROW \\
\text{[R1:alt]} & & \equiv & \neg P \lor GROW \\
 & (P \not\equiv \mathbf{R1}(P)) & \equiv & P \not\Rightarrow GROW \\
\text{[not-R1:alt]} & & \equiv & P \land \neg GROW
\end{array}
$$

3.4.2 Reactive Healthiness 2 (R2)

The appropriate definition of **R2** is the following, which is just strong enough to ensure that slot-sequence equality is itself **R2**-healthy:

[R2:def] $\quad \mathbf{R2}(P) \mathrel{\widehat=} \exists\, ss \bullet$
$$P[ss, ss \sharp (slots' \smallsetminus\!\!\smallsetminus slots)/slots, slots'] \land eqvref(ss) = eqvref(slots)$$

See §?? for details of how this formulation was developed.

For convenience we define the following shorthand:

[EQRF:def] $ER(ss_1, ss_2) \mathrel{\widehat{=}} eqvref(ss_1) = eqvref(ss_2)$
[R2:subs] $[r2] = [ss \frown \langle s \rangle, ss \frown Shift(s, slots, slots') \ / \ slots, slots']$

Using this and [R2:subs]:p24 gives the alternate compact definition of **R2**:

[R2:alt] $\mathbf{R2}(P) \mathrel{\widehat{=}} \exists\, ss \bullet R2_{ss}(P) \land ER(ss, slots)$

We note that $R2$ distributes over/through most predicate constructs, except through quantification over *slots* and *slots*$'$. The substitution and **R2** are both idempotent:

[R2:idem] $\mathbf{R2} \circ \mathbf{R2} = \mathbf{R2}$

Proof: [R2:idem]:p117.

As **R2** has the form $\exists\, x \bullet P[f(x), g(x)/slots, slots'] \land h(x)$, we immediately get the following distributive laws

[R2:distr:and] $\mathbf{R2}(P \land Q) \equiv \mathbf{R2}(P) \land Q, \quad slots, slots'$ not free in Q
[R2:distr:or] $\mathbf{R2}(P \lor Q) \equiv \mathbf{R2}(P) \lor \mathbf{R2}(Q)$
[R2:distr:cond] $\mathbf{R2}(P \lhd c \rhd Q) \equiv \mathbf{R2}(P) \lhd c \rhd \mathbf{R2}(Q), \quad slots, slots'$ not free in c

Proofs: [R2:distr:and]:p121, [R2:distr:or]:p121,[R2:distr:cond]:p122.

Note also that any predicate not mentioning *slots* and *slots*$'$ is **R2**-healthy, so:

[R2:no-slots] $\mathbf{R2}(Q) = Q, \quad slots, slots'$ not free in Q
[R2:distr:and'] $\mathbf{R2}(P \land Q) = \mathbf{R2}(P) \land \mathbf{R2}(Q),$
$(slots, slots'$ not free in $P) \lor (slots, slots'$ not free in $Q)$

Distributivity of healthiness through sequential composition generally doesn't hold, but it should preserve healthiness:

[comp:R2:closed] $(P \equiv \mathbf{R2}(P)) \land (Q \equiv \mathbf{R2}(Q)) \Rightarrow ((P;\ Q) \equiv \mathbf{R2}(P;\ Q))$

Proof: [comp:R2:closed]:p150.

3.4.3 Reactive Healthiness 3 (R3)

The definition of **R3** in the UTP book has to be adapted to avoid generating miracles as an interaction between state and external choice. The nature of this probelm and various possible solutions are discussed in [BGW09]. In essence, we have to encapsulate the principle that while a healthy process is waiting for events, its variable state is unobservable. We need to define $I\!I_R$ in order to specify **R3**.

[DIV:def] $DIV \mathrel{\widehat{=}} \neg\, ok \land GROW$
[RSTET:def] $RSTET \mathrel{\widehat{=}} wait' = wait \land slots' = slots$
[IIr:def] $I\!I_R \mathrel{\widehat{=}} DIV \lor ok' \land RSTET$
[R3:def] $\mathbf{R3}(P) \mathrel{\widehat{=}} I\!I_R \lhd wait \rhd P$
[R3:idem] $\mathbf{R3} \circ \mathbf{R3} = \mathbf{R3}$

Proofs: [R3:idem]:p123. $I\!I_R$ is a self-identity for composition:

[SkipR-SkipR:eq:SkipR] $\quad I\!I_R;\ I\!I_R \equiv I\!I_R$

Proofs: straightforward.

As **R3** has the form $I\!I_R \lhd wait \rhd \_$, we immediately get the following distributive laws (exercises in elementary predicate calculus):

$$
\begin{array}{rl}
\text{[R3:distr:and]} & \mathbf{R3}(P \land Q) = \mathbf{R3}(P) \land \mathbf{R3}(Q) \\
\text{[R3:distr:or]} & \mathbf{R3}(P \lor Q) = \mathbf{R3}(P) \lor \mathbf{R3}(Q) \\
\text{[R3:distr:cond]} & \mathbf{R3}(P \lhd c \rhd Q) = \mathbf{R3}(P) \lhd c \rhd \mathbf{R3}(Q) \\
\text{[R3:wait:Skip]} & (P \equiv \mathbf{R3}(P)) \Rightarrow wait \land P \equiv wait \land I\!I_R
\end{array}
$$

Distributivity of healthiness through sequential composition generally doesn't hold, but it should preserve healthiness:

[comp:R3:closed] $\quad (P \equiv \mathbf{R3}(P)) \land (Q \equiv \mathbf{R3}(Q)) \Rightarrow ((P;\ Q) \equiv \mathbf{R3}(P;\ Q))$

Proof: [comp:R3:closed]:p156.

We also have some laws regarding DIV and $RSTET$:

$$
\begin{array}{rl}
\text{[one-point:RSTET]} & (\exists\, obs_0 \bullet P \land (RSTET[obs_0/obs])) \equiv (\exists\, ok_0, state_0 \bullet P[wait', slots'/wait_0, slots_0]) \\
\text{[one-point:RSTET']} & (\exists\, obs_0 \bullet P \land (RSTET[obs_0/obs'])) \equiv (\exists\, ok_0, state_0 \bullet P[wait, slots/wait_0, slots_0]) \\
\text{[comp:RSTET]} & (P;\ Q \land RSTET) \\
 & \equiv \exists\, ok_0, state_0 \bullet P[ok_0, state_0/ok', state'] \land Q[ok_0, state_0, rest'/ok, state, rest] \\
\text{[comp:RSTET']} & (P \land RSTET;\ Q) \\
 & \equiv \exists\, ok_0, state_0 \bullet P[ok_0, state_0, rest/ok', state', rest'] \land Q[ok_0, state_0/ok, state] \\
\text{[DIV:DIV:eq:DIV]} & DIV;\ DIV \equiv DIV \\
\text{[IIr:DIV:eq:DIV]} & I\!I_R;\ DIV \equiv DIV \\
\text{[DIV:IIr:eq:DIV]} & DIV;\ I\!I_R \equiv DIV
\end{array}
$$

Proofs: the first two are an easy exercise in the one-point rule, the second two are straightforward corollaries of the first two whilst for the rest see: [DIV:DIV:eq:DIV]:p158, [IIr:DIV:eq:DIV]:p160, [DIV:IIr:eq:DIV]:p160.

3.4.4 Reactive Healthiness (R)

[R:def] $\quad \mathbf{R} = \mathbf{R3} \circ \mathbf{R2} \circ \mathbf{R1}$

In order to prove that **R** is idempotent, we shall show that each of the **Ri** commutes with each of the others. It is also useful to prove that some of these conditions, when applied to **true** also satisfy other healthiness conditions.

[R1:is:R2]	$\mathbf{R2}(\mathbf{R1}(\mathbf{true})) \equiv \mathbf{R1}(\mathbf{true})$
[IIr:is:R1]	$\mathbf{R1}(\mathit{II}_R)) \equiv \mathit{II}_R$
[R3:is:R1]	$\mathbf{R1}(P) \equiv P \Rightarrow \mathbf{R1}(\mathbf{R3}(P)) \equiv \mathbf{R3}(P)$
[IIr:is:R2]	$\mathbf{R2}(\mathit{II}_R)) \equiv \mathit{II}_R$
[IIr:is:R3]	$\mathbf{R3}(\mathit{II}_R)) \equiv \mathit{II}_R$
[R1:R2:comm]	$\mathbf{R1} \circ \mathbf{R2} = \mathbf{R2} \circ \mathbf{R1}$
[R1:R3:comm]	$\mathbf{R1} \circ \mathbf{R3} = \mathbf{R3} \circ \mathbf{R1}$
[R2:R3:comm]	$\mathbf{R2} \circ \mathbf{R3} = \mathbf{R3} \circ \mathbf{R2}$
[R:idem]	$\mathbf{R} \circ \mathbf{R} = \mathbf{R}$

Proofs: [R1:is:R2]:p124, [IIr:is:R1]:p133, [R3:is:R1]:p134, [IIr:is:R2]:p**??**, [IIr:is:R3]:p136, [R1:R2:comm]:p137, [R1:R3:comm]:p138, [R2:R3:comm]:p139, [R:idem]:p139.

We immediately obtain the following distributive laws for **R**, as a consequence of those above, and the definition of function composition:

[R:distr:and]	$\mathbf{R}(P \land Q) = \mathbf{R}(P) \land \mathbf{R}(Q)$,
	$(\mathit{slots}, \mathit{slots}'$ not free in $P) \lor (\mathit{slots}, \mathit{slots}'$ not free in $Q)$
[R:distr:or]	$\mathbf{R}(P \lor Q) = \mathbf{R}(P) \lor \mathbf{R}(Q)$
[R:distr:cond]	$\mathbf{R}(P \lhd c \rhd Q) = \mathbf{R}(P) \lhd c \rhd \mathbf{R}(Q)$, $\quad \mathit{slots}, \mathit{slots}'$ not free in c

It can be useful to expand out **R**-healthiness in a variety of ways:

	$\mathbf{R}(P)$
[R:expand:1]	$\equiv \mathit{II}_R \lhd \mathit{wait} \rhd (\mathbf{R2}(P) \land \mathit{slots} \preccurlyeq \mathit{slots}')$
[R:expand:2]	$\equiv (\mathit{II}_R \lhd \mathit{wait} \rhd \mathbf{R2}(P)) \land \mathit{slots} \preccurlyeq \mathit{slots}'$
[R:expand:3]	$\equiv \mathbf{R2}((\mathit{II}_R \lhd \mathit{wait} \rhd (P)) \land \mathit{slots} \preccurlyeq \mathit{slots}')$
[R:expand:3]	$\equiv \mathbf{R2}(\mathit{II}_R \lhd \mathit{wait} \rhd (P \land \mathit{slots} \preccurlyeq \mathit{slots}'))$

A surprising result is the following (Lemma 3.3, [She06, p47]):

[expand-R:eq:expand] $\quad P \equiv \mathbf{R}(P) \Rightarrow \mathit{GROW};\ P = \mathit{GROW}$

Proof: [expand-R:eq:expand]:p141

Less surprising is

[DIV-R:eq:DIV] $\quad P \equiv \mathbf{R}(P) \Rightarrow \mathit{DIV};\ P = \mathit{DIV}$

Proof: [DIV-R:eq:DIV]:p142

In both of the above, **R1** and **R3** are necessary, but **R2** is not.

We also have:

[R-DIV:eq:DIV] $\quad P \equiv \mathbf{R}(P) \Rightarrow P;\ \mathit{DIV} = \mathit{DIV}$

Proof: [R-DIV:eq:DIV]:p**??**

3.4.5 CSP Healthiness 1 (CSP1)

A process is **CSP1**-healthy if it behaves like DIV when started in an unstable state:

[CSP1:def]	$\mathbf{CSP1}(P) \mathrel{\widehat=} P \lor DIV$
[CSP1:alt]	$\mathbf{CSP1}(P) \equiv P \lor \mathbf{CSP1}(\mathbf{false})$
[DIV:is:CSP1]	$\mathbf{CSP1}(DIV) \equiv DIV$

Laws [CSP1:alt] and [DIV:is:CSP1] are easy exercises in prop. calculus.

[DIV:is:R1]	$\mathbf{R1}(DIV) = DIV$
[DIV:is:R2]	$\mathbf{R2}(DIV) = DIV$
[IIr:is:CSP1]	$\mathbf{CSP1}(I\!I_R) = I\!I_R$
[R1:CSP1:comm]	$\mathbf{R1} \circ \mathbf{CSP1} = \mathbf{CSP1} \circ \mathbf{R1}$
[R2:CSP1:comm]	$\mathbf{R2} \circ \mathbf{CSP1} = \mathbf{CSP1} \circ \mathbf{R2}$
[R3:CSP1:comm]	$\mathbf{R3} \circ \mathbf{CSP1} = \mathbf{CSP1} \circ \mathbf{R3}$

Proofs: [DIV:is:R1], [DIV:is:R2] are trivial; for the rest, see: [IIr:is:CSP1]:p143, [R1:CSP1:comm]:p144, [R2:CSP1:comm]:p144, [R3:CSP1:comm]:p146.

As **CSP1** has the form or_P, we get:

[CSP1:distr:and]	$\mathbf{CSP1}(P \land Q) = \mathbf{CSP1}(P) \land \mathbf{CSP1}(Q)$
[CSP1:distr:or]	$\mathbf{CSP1}(P \lor Q) = \mathbf{CSP1}(P) \lor \mathbf{CSP1}(Q)$
[CSP1:distr:cond]	$\mathbf{CSP1}(P \lhd c \rhd Q) = \mathbf{CSP1}(P) \lhd c \rhd \mathbf{CSP1}(Q)$
[CSP1:distr:all]	$\mathbf{CSP1}(\forall x \bullet P) = \forall x \bullet \mathbf{CSP1}(P), x \notin \{ok, slots, slots'\}$
[CSP1:distr:any]	$\mathbf{CSP1}(\exists x \bullet P) = \exists x \bullet \mathbf{CSP1}(P), x \notin \{ok, slots, slots'\}$

We would also expect **CSP1** to be closed under sequential composition:

[comp:CSP1:closed]	$(P \equiv \mathbf{CSP1}(P)) \land (Q \equiv \mathbf{CSP1}(Q)) \Rightarrow ((P;\ Q) \equiv \mathbf{CSP1}(P;\ Q))$

Proof: [comp:CSP1:closed]:p153.

3.4.6 CSP Healthiness 2 (CSP2)

[CSP2:def]	$\mathbf{CSP2}(P) \mathrel{\widehat=} P;\ (ok \Rightarrow ok' \land wait' = wait \land state' = state \land slots' = slots)$
[CSP:alt]	$\exists\, ok_0 \bullet P[ok_0/ok'] \land ok_0 \Rightarrow ok'$

Note that here we use equality on slots rather than equivalence in **CSP2**, *(Revisit proofs below)* which allows the alternative form to be used, as the one-point rule can immediately be used to eliminate variables $wait_0$, $state_0$ and $slots_0$.

Condition **CSP2** uses sequential composition, which embodies existential quantification over ok_0 so its distributivity properties are non-trivial exercises in predicate calculus:

[CSP2:distr:and]	$\mathbf{CSP2}(P \land Q) = \mathbf{CSP2}(P) \land Q, \quad ok'$ not free in Q
[CSP2:distr:or]	$\mathbf{CSP2}(P \lor Q) = \mathbf{CSP2}(P) \lor \mathbf{CSP2}(Q)$
[CSP2:distr:any]	$\mathbf{CSP2}(\exists\, x \bullet P) = \exists\, x \bullet \mathbf{CSP2}(P), \quad x \neq ok'$
[CSP2:distr:cond]	$\mathbf{CSP2}(P \lhd c \rhd Q) = \mathbf{CSP2}(P) \lhd c \rhd \mathbf{CSP2}(Q), \quad ok'$ not free in c

We explore its relationship with the other healthiness conditions:

[IIr:is:CSP2]	$\mathbf{CSP2}(I\!I_R) = I\!I_R$
[DIV:is:CSP2]	$\mathbf{CSP2}(DIV) = DIV$
[R1:CSP2:comm]	$\mathbf{R1} \circ \mathbf{CSP2} = \mathbf{CSP2} \circ \mathbf{R1}$
[R2:CSP2:comm]	$\mathbf{R2} \circ \mathbf{CSP2} = \mathbf{CSP2} \circ \mathbf{R2}$
[R3:CSP2:comm]	$\mathbf{R3} \circ \mathbf{CSP2} = \mathbf{CSP2} \circ \mathbf{R3}$
[CSP1:CSP2:comm]	$\mathbf{CSP1} \circ \mathbf{CSP2} = \mathbf{CSP2} \circ \mathbf{CSP1}$

Proofs: [IIr:is:CSP2]:p161, [DIV:is:CSP2]:p162, [R1:CSP2:comm]:p163, [R2:CSP2:comm]:p164, [R3:CSP2:comm]:p165, [CSP1:CSP2:comm]:p166.

3.4.7 CSP Healthiness 3 (CSP3)

[CSP3:def] $\mathbf{CSP3}(P) \mathrel{\widehat=} \mathit{Skip};\ P$

The definition of *Skip* is given later.

Condition **CSP3** uses sequential composition, which embodies existential quantification over obs_0 so its distributivity properties are non-trivial exercises in predicate calculus:

[CSP3:distr:and]	$\mathbf{CSP3}(P \land Q) = \mathbf{CSP3}(P) \land Q, \quad obs$ not free in Q
[CSP3:distr:or]	$\mathbf{CSP3}(P \lor Q) = \mathbf{CSP3}(P) \lor \mathbf{CSP3}(Q)$
[CSP3:distr:any]	$\mathbf{CSP3}(\exists\, x \bullet P) = \exists\, x \bullet \mathbf{CSP3}(P), \quad x \notin obs$
[CSP3:distr:cond]	$\mathbf{CSP3}(P \lhd c \rhd Q) = \mathbf{CSP3}(P) \lhd c \rhd \mathbf{CSP3}(Q), \quad obs$ not free in c

The idempotence of this, as well as **CSP4** below depends on the following law:

[Skip;Skip] $\mathit{Skip};\ \mathit{Skip} \equiv \mathit{Skip}$

3.4.8 CSP Healthiness 4 (CSP4)

[CSP4:def] $\mathbf{CSP4}(P) \mathrel{\widehat=} P;\ \mathit{Skip}$

Condition **CSP4** uses sequential composition, which embodies existential quantification over obs_0 so its distributivity properties are non-trivial exercises in predicate calculus:

[CSP4:distr:and]	$\mathbf{CSP4}(P \land Q) = \mathbf{CSP4}(P) \land Q, \quad obs'$ not free in Q
[CSP4:distr:or]	$\mathbf{CSP4}(P \lor Q) = \mathbf{CSP4}(P) \lor \mathbf{CSP4}(Q)$
[CSP4:distr:any]	$\mathbf{CSP4}(\exists\, x \bullet P) = \exists\, x \bullet \mathbf{CSP4}(P), \quad x \notin obs'$
[CSP4:distr:cond]	$\mathbf{CSP4}(P \lhd c \rhd Q) = \mathbf{CSP4}(P) \lhd c \rhd \mathbf{CSP4}(Q), \quad obs'$ not free in c

In order for external choice to give a healthy result for healthy components, we have to require that "healthy" encompasses at least **R3** (as modified by us) and **CSP4**. Without these, then we can get miracles (*False*) from external choice.

3.4.9 CSP Healthiness 5 (CSP5)

$$[\mathsf{CSP5{:}def}] \qquad \mathbf{CSP5}(P) \mathrel{\widehat=} P \llbracket \Delta P \mid \{\!\mid \phi \mid\!\} \mid \phi \rrbracket \; \mathit{Skip}$$

The definition of $\llbracket \ldots \mid \ldots \mid \ldots \rrbracket$ is given later.

3.4.10 Healthy Processes

In general a process is "healthy" if it satisfies **R**, **CSP1**, **CSP2** and **CSP4**. For now we reserve judgment regarding**CSP3** and **CSP5**, viewing them as "ultra-healthy".

3.5 Slotted-*Circus* Specific Actions

Now, we define some relations that are useful in defining the slotted-*Circus* actions. Most of these are a strengthening of $POSSEVTS$ [PEV:def]:p18.

Sometimes we will want to allow time to pass ($\#slots' > \#slots$) but disallow the occurrence of any events ($trace' = \langle\rangle$). This leads us to define a stronger version of $POSSEVTS$, called $NOEVTS$:

$$\text{[NEV:sig]} \quad NOEVTS_{\mathcal{S}} : (\mathcal{S}\ E)^{+} \leftrightarrow (\mathcal{S}\ E)^{+}$$
$$\text{[NEV:def]} \quad NOEVTS(slots, slots') \ \widehat{=}\ EQVTRACE(nil, slots' \setminus\!\!\setminus slots)$$

Another useful strengthening is a predicate that asserts that a given set of events have occurred, but that no time has passed:

$$\text{[EVN:sig]} \quad EVTSNOW_{\mathcal{S}} : \mathbb{P}\, E \rightarrow (\mathcal{S}\ E)^{+} \leftrightarrow (\mathcal{S}\ E)^{+}$$
$$\text{[EVN:def]} \quad EVTSNOW(E)(slots, slots')$$
$$\widehat{=}\ \exists\, tt \bullet elems(tt) = E \land EQVTRACE(tt, slots' \setminus\!\!\setminus slots) \land \#slots = \#slots'$$

We can then use these as building blocks for other predicates.

In some situations, we want to describe events that occur immediately (in the first slot), as described by the relation $IMMEVTS$:

$$\text{[IME:sig]} \quad IMMEVTS_{\mathcal{S}} : (\mathcal{S}\ E)^{+} \leftrightarrow (\mathcal{S}\ E)^{+}$$
$$\text{[IME:def]} \quad IMMEVTS(slots, slots') \ \widehat{=}\ \exists\, E \bullet E \neq \emptyset \land EVTSNOW(E)(slots, slots');\ GROW$$
$$\text{[FSE:sig]} \quad FSTEVT_{\mathcal{S}} : E \rightarrow (\mathcal{S}\ E)^{+} \leftrightarrow (\mathcal{S}\ E)^{+}$$
$$\text{[FSE:def]} \quad FSTEVT(c.e)(slots, slots') \ \widehat{=}\ EVTSNOW\{c.e\}(slots, slots');\ GROW$$

Conversely, we also want to describe situation when events occur, but only in the last slot:

$$\text{[EEV:sig]} \quad ENDEVTS_{\mathcal{S}} : (\mathcal{S}\ E)^{+} \leftrightarrow (\mathcal{S}\ E)^{+}$$
$$\text{[EEV:def]} \quad ENDEVTS(slots, slots')$$
$$\widehat{=}\ \exists\, E \bullet E \neq \emptyset \land (EVTSNOW(E)(slots, slots') \lor (NOEVTS(slots, slots')$$
$$;\ EVTSNOW(E)(slots, slots')))$$
$$\text{[LSE:sig]} \quad LASTEVT_{\mathcal{S}} : E \rightarrow (\mathcal{S}\ E)^{+} \leftrightarrow (\mathcal{S}\ E)^{+}$$
$$\text{[LSE:def]} \quad LASTEVT(c.e)(slots, slots')$$
$$\widehat{=}\ EVTSNOW\{c.e\}(slots, slots') \lor (NOEVTS(slots, slots');\ EVTSNOW\{c.e\}(slots, slots'))$$

The above definitions cover only the *slots* variables, and so it only makes sense to check if they are **R1** and **R2**:

$$\text{[NEV:is:R1:R2]} \quad \mathbf{R2}(\mathbf{R1}((NOEVTS(slots, slots'))) \equiv NOEVTS(slots, slots')$$
$$\text{[EVN:is:R1:R2]} \quad \mathbf{R2}(\mathbf{R1}(EVTSNOW(E)(slots, slots'))) \equiv EVTSNOW(E)(slots, slots')$$
$$\text{[FSE:is:R1:R2]} \quad \mathbf{R2}(\mathbf{R1}(FSTEVT(c.e)(slots, slots'))) \equiv \mathbf{???}\ FSTEVT(c.e)(slots, slots')$$
$$\text{[IME:is:R1:R2]} \quad \mathbf{R2}(\mathbf{R1}(IMMEVTS(slots, slots'))) \equiv \mathbf{???}\ IMMEVTS(slots, slots')$$

Proofs (to be redone with new definitions): [NEV:is:R1:R2]:p168, [EVN:is:R1:R2]:p169, [FSE:is:R1:R2]:p**??**, [IME:is:R1:R2]:p170.

We have just introduced three relations on slot-sequences, and it is a good idea to determine what, if any, of the "classical" relation properties they possess:

[NEV:refl]	$NOEVTS(ss, ss) \equiv \textsc{True}$
[NEV:trans]	$NOEVTS(ss, tt) \land NOEVTS(tt, uu) \Rightarrow NOEVTS(ss, uu)$
	$NOEVTS(slots, slots'); \; NOEVTS(slots, slots') \equiv NOEVTS(slots, slots')$
[NEV:anti]	$NOEVTS(ss, tt) \land NOEVTS(tt, ss) \Rightarrow ss \cong tt$
[EVN:irr]	$EVTSNOW(E)(ss, ss) \equiv \textsc{False}$
[FSE:irr]	$FSTEVT(c.e)(ss, ss) \equiv \textsc{False}$
[IME:irr]	$IMMEVTS(ss, ss) \equiv \textsc{False}$

Proofs yet to be done.

3.5.1 Laws

[specificALaw-1]	$EVTSNOW(\emptyset)(ss, tt) \equiv ss \cong tt$
[specificALaw-2]	$EVTSNOW(\emptyset)(ss, tt) \equiv NOEVTS(ss, tt) \land \sharp ss = \sharp tt$
[specificALaw-3]	$(c.e \to P) \land wait' \land NOEVTS(slots, slots')$
[specificALaw-4]	$\exists\, s, s' \bullet EVTSNOW\{c\}(s, s') \land sl' \searrow\!\!\searrow sl = map(SHide(\{c\}))(s' \searrow\!\!\searrow s) \equiv sl' \cong sl$

3.6 Actions

First, we define those relations whose definitions are the same as those found in UTP or *Circus*:

3.6.1 Nondeterministic Choice

[NDet:def] $P \sqcap Q \mathrel{\widehat=} P \lor Q$

3.6.2 Conditional Choice

[Cond:def] $P \lhd c \rhd Q \mathrel{\widehat=} c \land P \lor \neg\, c \land Q$

[Cond:alt] $P \lhd c \rhd Q \equiv (c \Rightarrow P) \land (\neg\, c \Rightarrow Q)$

3.6.3 Sequential Composition

[Seq:subs] $[seq] \mathrel{\widehat=} [obs_0/obs] = [ok_0, wait_0, state_0, slots_0/ok, wait, state, slots]$

[Seq:subs'] $[seq'] \mathrel{\widehat=} [obs_0/obs'] = [ok_0, wait_0, state_0, slots_0/ok', wait', state', slots']$

[Seq:def] $P;\ Q \mathrel{\widehat=} \exists\, obs_0 \bullet P[seq'] \land Q[seq]$

3.6.4 Chaos

[Chaos:def] $Chaos \mathrel{\widehat=} \mathbf{R}(\mathbf{true})$

3.6.5 Deadlock

We can now define *Stop* as:

[Stop:def] $Stop \mathrel{\widehat=} \mathbf{CSP1}(\mathbf{R3}(ok' \land wait' \land NOEVTS(slots, slots')))$

3.6.6 Guard

We can now define & as:

[Grd:def] $b \& A \mathrel{\widehat=} A \lhd b \rhd Stop$

3.6.7 Termination

Process *Skip* is reactive, and ignores any refusals at the start:

[Skip:def] $Skip \mathrel{\widehat=} \mathbf{R}(\exists\, slots_r \bullet slots_r \cong slots \land I\!I_R[slots_r/slots] \land state = state')$

This definition corrects an erroneous one that appears in the literature (§**??**).

3.6.8 Delay

Waiting means time passes with no events occurring:

[Wait:def]	$Wait\ t \mathrel{\widehat=} \mathbf{CSP1}(\mathbf{R}(ok' \land DELAY(t) \land NOEVTS(slots, slots')))$
[Del:def]	$DELAY(t) \mathrel{\widehat=} DELW(t) \lhd wait' \rhd DELD(t)$
[DELW:def]	$DELW(t) \mathrel{\widehat=} (\#slots' - \#slots < t)$
[DELD:def]	$DELD(t) \mathrel{\widehat=} (\#slots' - \#slots = t \land state' = state)$

3.6.9 Assignment

Assignment takes zero time in slotted-*Circus* (here val is a standard expression evaluator):

$$\text{[Asg:def]} \qquad x := e \mathrel{\widehat=} \mathbf{CSP1}\left(\mathbf{R}\left(\begin{array}{r} ok = ok' \land wait = wait' \land slots = slots' \\ \land\ state' = state \oplus \{x \mapsto val(e, state)\}\end{array}\right)\right)$$

$$val \ :\ \mathsf{Expr} \times (\mathsf{Name} \to Value) \nrightarrow Value$$

3.6.10 Communication (Prefix)

We distinguish between waiting for communication (WTC) and terminating it ($TRMC$).

[Comm:def]	$c.e \to Skip \mathrel{\widehat=} \mathbf{CSP1}\left(ok' \land \mathbf{R3}\left(WTC(c) \lhd wait' \rhd \left(\begin{array}{l} state' = state\ \land \\ WTC(c);\ TRMC(c)\end{array}\right)\right)\right)$
[WTC:def]	$WTC(c) \mathrel{\widehat=} POSS(c) \land NOEVTS(slots, slots')$
[POSS:def]	$POSS(c) \mathrel{\widehat=} c \notin \bigcup Refs(slots' \setminus\!\setminus slots)$
[TRMC:def]	$TRMC(c) \mathrel{\widehat=} EVTSNOW\{c\}(slots, slots')$

We show first that the key fragments are in fact **R2**- and **R1**-healthy:

[WTC:is:R1:R2]	$\mathbf{R2}(\mathbf{R1}(WTC(c))) \equiv_{\mathbf{???}} WTC(c)$
[TRMC:is:R1:R2]	$\mathbf{R2}(\mathbf{R1}(TRMC(c.e))) \equiv_{\mathbf{???}} TRMC(c.e)$

Proofs: [WTC:is:R1:R2]:p172, [CMPC:is:R1:R2]:p**??**
General prefixing can then be defined in the standard way:

[Out:def]	$c!e \to Skip \mathrel{\widehat=} c.e \to Skip$
[In:def]	$c?x \to Skip \mathrel{\widehat=} \Box_{k:T} \bullet (c.k \to Skip;\ x := k)$
[Pfx:def]	$comm \to A \mathrel{\widehat=} (comm \to Skip);\ A$

Here $\Box_{k:T} \bullet P(x)$ is shorthand for $P(k_1) \Box P(k_2) \Box \ldots$.

3.6.11 External Choice

The definition here is a major adaptation of that in [She06, p69].

$$\text{[Ext:def]} \qquad A \Box B \mathrel{\widehat=} A \land B \land Stop \lor Choice(A, B) \lor Choice(B, A)$$

$$[\textsf{Choice:def}]\,Choice(C,R) \quad \widehat{=} \quad \mathbf{CSP2}\left(C \land \left(\begin{array}{l} R \land NoEvents(slots, slots') \\ ; \\ \left(\begin{array}{l} ImmEvents(slots, slots') \lor \\ slots \cong slots' \land (\neg wait' \lor \neg ok') \end{array} \right) \end{array} \right) \right)$$

The definition of **R3** using $I\!I_R$ rather than $I\!I$ is crucial here, otherwise the following outcome results:

$$(x := 1;\ a \rightarrow Skip) \,\square\, (x := 2;\ b \rightarrow Skip) = \textsc{False} \lhd wait \rhd \ldots$$

As $STET$ propagates state changes through, and we are *wait*ing after the assignments, but before a or b is chosen, then the clause $A \land B \land Stop$ results a contradiction as it asserts, among other things that $x' = 1 \land x' = 2$. With $RSTET$ used in **R3**, this state information does not appear, so there is no conflict, and once the a or b event is complete, so that we are no longer *wait*ing, then the actual assignment outcome becomes visible. Details regarding this issue have been published in [BGW09].

3.6.12 Parallel Composition

We define slotted-parallel in a direct fashion, similar to that used for *Circus*, avoiding the complexities of the UTP/CTA approaches, and also handling error cases in passing:

$$\begin{array}{rl}
[\textsf{Par:def}] & A \,[\![\, s_A \mid \{\!|\, cs \,|\!\} \mid s_B \,]\!]\, B \mathrel{\widehat{=}} \exists\, obs_A, obs_B \bullet \\
& \quad A[obs_A/obs'] \land B[obs_B/obs'] \land \\
& \left(\begin{array}{ll}
\textbf{if} & \left(\begin{array}{l} s_A \lhd state_A \neq s_A \lhd state \lor \\ s_B \lhd state_B \neq s_B \lhd state \lor \\ s_A \cap s_B \neq \emptyset \end{array} \right) \\
\textbf{then} & \neg\ ok' \land slots \preccurlyeq slots' \\
\textbf{else} & \left(\begin{array}{l} ok' = ok_A \land ok_B \land \\ wait' = (wait_A \lor 1.wait_B) \land \\ (wait' \Rightarrow state' = (state_A - s_B) \oplus (state_B - s_A)) \land \\ (wait_A \Rightarrow \#slots_A \geq \#slots_B) \land \\ (wait_B \Rightarrow \#slots_B \geq \#slots_A) \land \\ VALIDMRG(cs)(slots, slots', slots_A, slots_B) \end{array} \right)
\end{array} \right) \\
[\textsf{VMrg:sig}] & VALIDMRG : \mathbb{P}\,E \rightarrow ((\mathcal{S}\ E)^+)^4 \rightarrow \mathbb{B} \\
[\textsf{VMrg:def}] & VALIDMRG(cs)(, s, s', s0, s_1) \mathrel{\widehat{=}} s' \setminus\!\setminus s \in tsync(cs)(s0 \setminus\!\setminus s, s_1 \setminus\!\setminus s) \\
[\textsf{TSnc:sig}] & tsync : \mathbb{P}\,E \rightarrow (\mathcal{S}\ E)^* \times (\mathcal{S}\ E)^* \rightarrow \mathbb{P}((\mathcal{S}\ E)^+) \\
[\textsf{TSnc:sym}] & tsync(cs)(s_1, s_2) = tsync(cs)(s_2, s_1) \\
[\textsf{TSnc:nil}] & tsync(cs)(\langle\rangle, \langle\rangle) \mathrel{\widehat{=}} \{\} \\
[\textsf{TSnc:one}] & tsync(cs)(\langle s \rangle, \langle\rangle) \mathrel{\widehat{=}} \{\ \langle s' \rangle \mid s' \in ssync(cs)(s, snull(sref(s)))\ \} \\
[\textsf{TSnc:both}] & tsync(cs)\left(\begin{array}{l} s_1 \mathbin{⨟} S_1, \\ s_2 \mathbin{⨟} S_2 \end{array} \right) \mathrel{\widehat{=}} \left\{ \begin{array}{l} s' \mathbin{⨟} S' \\ \mid s' \in ssync(cs)(s_1, s_2) \land \\ \quad S' \in tsync(cs)(S_1, S_2) \end{array} \right\}
\end{array}$$

The clause [TSnc:one] matches that on [She06, pp81], which amounts to treating the missing slot as refusing the same as the one present, in order to ensure that the refusal is unchanged. This may not be correct. If the missing slot should be treated as refusing everything, then we should have:

$$[\textsf{TSnc:one'}] \quad tsync(cs)(\langle s \rangle, \langle\rangle) \mathrel{\widehat{=}} \{\ \langle s' \rangle \mid s' \in ssync(cs)(s, snull(Events)))\ \}$$

Or should we select all s' with refusals the same as s, resulting from synchronising with null-slots ranging over arbitrary refusals:

$$\text{[TSnc:one'']} \qquad tsync(cs)(\langle s\rangle, \langle\rangle) \mathrel{\widehat=} \left\{ \begin{array}{l} \langle s'\rangle \mid \\ \quad \exists\, r \bullet \\ \qquad s' \in ssync(cs)(s, snull(r))) \land sref(s') = sref(s) \end{array} \right\}$$

3.6.13 Hiding

Hiding events in a slotted-action means they no longer appears as events that occurred, but are now considered as refusals (by the outside world). Also, hidden events do not wait, but occur immediately.

$$\text{[Hid:def]} \qquad A \setminus hidn \mathrel{\widehat=} \mathbf{R3} \left(\begin{array}{l} \exists\, s' \bullet A[s'/slots'] \land \\ \qquad slots' \mathbin{-\!\!-} slots = map(SHide(hidn))(s' \mathbin{-\!\!-} slots) \land \\ \qquad hidn \subseteq \bigcap Refs(s' \mathbin{-\!\!-} slots) \end{array} \right);\ Skip$$

3.6.14 Timeout

Timeout is modelled as per [She06, p86]

$$\text{[Tout:def]} \qquad A \overset{d}{\triangleright} B \mathrel{\widehat=} (A \Box (Wait\ d;\ int \to B)) \setminus \{int\}$$

3.6.15 Recursion

$$\text{[Rec:def]} \qquad \mu\, X \bullet F(X) \mathrel{\widehat=} \sqcap\{X \mid X \sqsupseteq F(X)\}$$

3.7 Laws

3.7.1 Prefix

Laws [She06, Property 3.5, p46]:

[prefixLaw-1] $\quad (c.e \to Skip) \land wait' \equiv \mathbf{CSP1}(ok' \land \mathbf{R}(WTC(c))) \land wait'$

[prefixLaw-2] $\quad (c.e \to Skip) \land \neg\, wait' \equiv \mathbf{CSP1}\left(ok' \land \mathbf{R3}\left(\begin{array}{l} state' = state \land \\ WTC(c);\ TRMC(c) \end{array}\right)\right) \land \neg wait'$

[prefixLaw-3] $\quad (c.e \to P) \land NOEVTS(slots, slots')$

$\equiv$ for healthy P

$\mathbf{CSP1}(ok' \land \mathbf{R3}(WTC(c) \land wait')) \land NOEVTS(slots, slots')$

3.7.2 Sequential Composition

Laws [She06, Property 3.6, p55]:

[seqLaw-1]	$STOP;\ A \equiv STOP, \qquad A$ healthy
[seqLaw-2]	$((x := e);\ (x := f(x)) \equiv x := f(e)$
[seqLaw-3]	$Wait\ n;\ Wait\ m \equiv Wait\ (m+n)$
[seqLaw-4]	$A;\ (B;\ C) \equiv (A;\ B);\ C$
[seqLaw-5]	$(A \lhd c \rhd B);\ Z \equiv (A;\ Z) \lhd c \rhd (B;\ Z)$

The proofs of the last two are predicate calculus exercises, while the first three require a little more work: [seqLaw-1]:p186, [seqLaw-2]:p188, [seqLaw-3]:p189.

3.7.3 Conditional

Laws [She06, Property 3.7, pp55–6]:

[condLaw-1]	$P \lhd c \rhd P \equiv P$
[condLaw-2]	$P \lhd c \rhd Q \equiv Q \lhd \neg\ c \rhd P$
[condLaw-3]	$(P \lhd b \rhd Q) \lhd c \rhd R \equiv P \lhd b \wedge c \rhd (Q \lhd c \rhd R)$
[condLaw-4]	$P \lhd b \rhd (Q \lhd c \rhd R) \equiv (P \lhd b \rhd Q) \lhd c \rhd (P \lhd b \rhd R)$
[condLaw-5]	$P \lhd \textsc{True} \rhd Q \equiv P \equiv Q \lhd \textsc{False} \rhd P$
[condLaw-6]	$P \lhd c \rhd (Q \lhd c \rhd R) \equiv P \lhd c \rhd R$
[condLaw-7]	$P \lhd b \rhd (P \lhd c \rhd Q) \equiv P \lhd b \wedge c \rhd Q$
[condLaw-8]	$(P \sqcap Q) \lhd c \rhd R \equiv (P \lhd c \rhd R) \sqcap (Q \lhd c \rhd R)$

All are straightforward exercises in propositional calculus.

3.7.4 Guards

Laws [She06, Property 3.8, pp56]:

[guardLaw-1]	$\textsc{False} \& P \equiv Stop$
[guardLaw-2]	$\textsc{True} \& P \equiv P$
[guardLaw-3]	$c \& Stop \equiv Stop$
[guardLaw-4]	$b \& (c \& P) \equiv (b \wedge c) \& P$
[guardLaw-5]	$b \& (P \sqcap Q) \equiv (b \& P) \sqcap (b \& Q)$
[guardLaw-6]	$b \& (P;\ Q) \equiv \mathbf{???}\ (b \& P);\ Q$
[guardLaw-7]	$b \& P \equiv \mathbf{???}\ (b \& Skip);\ P$

All but the last two are straightforward exercises in propositional calculus.

3.7.5 Non-deterministic Choice

Laws [She06, Property 3.9, p57]:

[intchoiceLaw-1]	$Chaos \sqcap P \equiv Chaos$
[intchoiceLaw-2]	$P \sqcap P \equiv P$
[intchoiceLaw-3]	$P \sqcap Q \equiv Q \sqcap P$
[intchoiceLaw-4]	$P \sqcap (Q \sqcap R) \equiv (P \sqcap Q) \sqcap R$
[intchoiceLaw-5]	$(P \sqcap Q);\ R \equiv P;\ R \sqcap Q;\ R$
[intchoiceLaw-6]	$P;\ (Q \sqcap R) \equiv P;\ Q \sqcap P;\ R$
[intchoiceLaw-7]	$P \lhd c \rhd (Q \sqcap R) \equiv (P \lhd c \rhd Q) \sqcap (P \lhd c \rhd R)$
[intchoiceLaw-8]	$P \sqcap (Q \lhd c \rhd R) \equiv (P \sqcap Q) \lhd c \rhd (P \sqcap R)$
[intchoiceLaw-9]	$(c \to P) \sqcap (c \to R) \equiv$**???** $c \to (P \sqcap R)$

All but the last are straightforward exercises in propositional calculus.

3.7.6 External Choice Composition

Laws [She06, Property 3.10, pp70–7]:

[extchoiceLaw-1]	$P \Box Stop \equiv$**???** $P, \quad P$ healthy
[extchoiceLaw-2]	$P \Box P \equiv$**???** P
[extchoiceLaw-3]	$P \Box Q \equiv$**???** $Q \Box P$
[extchoiceLaw-4]	$P \Box (Q \Box R) \equiv$**???** $(P \Box Q) \Box R$
[extchoiceLaw-5]	$P \Box (Q \sqcap R) \equiv$**???** $(P \Box Q) \sqcap (P \Box R)$
[extchoiceLaw-6]	$(a \to P) \Box (b \to P) \equiv$**???** $((a \to Skip) \Box (b \to Skip));\ P$
[extchoiceLaw-7]	$Wait\ n \Box Wait\ n + m \equiv$**???** $Wait\ n$
[extchoiceLaw-8]	$(Wait\ n;\ P) \Box (Wait\ n;\ Q) \equiv$**???** $Wait\ n;\ (P \Box Q)$
[extchoiceLaw-9]	$(P \sqcap Q) \Box (P \sqcap R) \equiv$**???** $P \sqcap (Q \Box R)$
[extchoiceLaw-10]	$(Skip \Box (Wait\ n;\ P)) \equiv$**???** $Skip, \quad n > 0$
[extchoiceLaw-11]	$(a \to P) \Box (Wait\ n;\ (a \to P)) \equiv$**???** $(a \to P)$

None of these are straightforward!
Law [extchoiceLaw-9] may not be true... Counter example. $P = Stop, Q = Skip, R = Wait 1;\ a \to Skip$

$$LeftSide = (P \sqcap Q) \Box (P \sqcap R) = P \sqcap Q \sqcap R$$

$$RightSide = P \sqcap (Q \Box R) = P \sqcap Q$$

3.7.7 Parallel Composition

Laws [She06, Property 3.12, pp82–3]:

[parallelLaw-1] $A \llbracket s_A \mid \{\!| cs |\!\} \mid s_B \rrbracket B \equiv ??? \; B \llbracket s_B \mid \{\!| cs |\!\} \mid s_A \rrbracket A$

[parallelLaw-2] $A \llbracket s_A \mid \{\!| cs |\!\} \mid s_B \cup s_C \rrbracket (B \llbracket s_B \mid \{\!| cs |\!\} \mid s_C \rrbracket C)$
$\equiv ??? \; (A \llbracket s_A \mid \{\!| cs |\!\} \mid s_B \rrbracket B) \llbracket s_A \cup s_B \mid \{\!| cs |\!\} \mid s_C \rrbracket C$

[parallelLaw-3] $Skip \llbracket \{\} \mid \{\!| \; |\!\} \mid \{\} \rrbracket Skip \equiv ??? \; Skip$

[parallelLaw-4] $A \llbracket s_A \mid \{\!| \; |\!\} \mid \{\} \rrbracket Chaos \equiv ??? \; Chaos$

[parallelLaw-5] $Stop \llbracket \{\} \mid \{\!| cs |\!\} \mid s_A \rrbracket c \rightarrow A \equiv ??? \; Stop, \qquad c \in cs$

[parallelLaw-6] $(A \lhd b \rhd B)[s_A \cup s_B \mid \{\!| cs |\!\} \mid s_C]C$
$\equiv ??? \; (A \llbracket s_A \mid \{\!| cs |\!\} \mid s_C \rrbracket C) \lhd (\rhd B \llbracket s_B \mid \{\!| cs |\!\} \mid s_C \rrbracket C)$

[parallelLaw-7] $(A \sqcap B)[s_A \cup s_B \mid \{\!| cs |\!\} \mid s_C]C$
$\equiv ??? \; (A \llbracket s_A \mid \{\!| cs |\!\} \mid s_C \rrbracket C) \sqcap (B \llbracket s_B \mid \{\!| cs |\!\} \mid s_C \rrbracket C)$

[parallelLaw-8] $(x := e; \; A) \llbracket s_A \mid \{\!| cs |\!\} \mid s_B \rrbracket B$
$\equiv ??? \; x := e; \; (A \llbracket s_A \mid \{\!| cs |\!\} \mid s_B \rrbracket B$
if m (?) not in e, and $x \in s_A \setminus s_B$

[parallelLaw-9] $(Wait\, n; \; A) \llbracket s_A \mid \{\!| cs |\!\} \mid s_B \rrbracket (Wait\, n; \; B)$
$\equiv ??? \; Wait\, n; \; (A \llbracket s_A \mid \{\!| cs |\!\} \mid s_B \rrbracket B)$

[parallelLaw-10] $a \notin cs \land b \notin cs \land c \in cs \Rightarrow$
$((a \rightarrow A) \Box (b \rightarrow B) \llbracket s_A \cup s_B \mid \{\!| cs |\!\} \mid s_C \rrbracket C) \setminus cs$
$\equiv ??? \; ((a \rightarrow A \llbracket s_A \mid \{\!| cs |\!\} \mid s_C \rrbracket C) \Box (b \rightarrow B \llbracket s_B \mid \{\!| cs |\!\} \mid s_C \rrbracket C)) \setminus cs$

[parallelLaw-11] $a \in cs \land b \in cs \Rightarrow$
$((a \rightarrow A) \llbracket s_A \mid \{\!| cs |\!\} \mid s_B \rrbracket ((a \rightarrow A') \Box (b \rightarrow B))) \setminus cs$
$\equiv ??? \; (a \rightarrow (A \llbracket s_A \mid \{\!| cs |\!\} \mid s_B \rrbracket A')) \setminus cs$

[parallelLaw-12] $a \in cs \land b \in cs \Rightarrow$
$((a \rightarrow A) \llbracket s_A \mid \{\!| cs |\!\} \mid s_B \rrbracket ((Wait\, d; \; a \rightarrow A') \Box (b \rightarrow B))) \setminus cs$
$\equiv ??? \; Wait\, d; \; (a \rightarrow (A \llbracket s_A \mid \{\!| cs |\!\} \mid s_B \rrbracket A')) \setminus cs$

[parallelLaw-13] $a \notin cs \Rightarrow$
$(a \rightarrow A) \llbracket s_A \mid \{\!| cs |\!\} \mid s_B \rrbracket (b \rightarrow B)$
$\equiv ??? \; a \rightarrow (A \llbracket s_A \mid \{\!| cs |\!\} \mid s_B \rrbracket (b \rightarrow B))$

All complicated!

3.7.8 Hiding

Laws [She06, Property 3.14, p85]:

[hidingLaw-1]	$P \setminus \{\} \equiv P, \qquad P$ healthy
[hidingLaw-2]	$P \setminus S \setminus T \equiv= P \setminus (S \cup T)$
[hidingLaw-3]	$(P \sqcap Q) \setminus S \equiv (P \setminus S) \sqcap (Q \setminus S)$
[hidingLaw-4]	$(c \rightarrow P) \setminus S \equiv c \rightarrow (P \setminus S), \qquad c \notin S$
[hidingLaw-5]	$(c \rightarrow P) \setminus S \equiv (P \setminus S), \qquad c \in S$
[hidingLaw-6]	$(\mathit{Wait}\ n) \setminus S \equiv \mathit{Wait}\ n$
[hidingLaw-7]	$\mathit{Skip} \setminus S \equiv \mathit{Skip}$
[hidingLaw-8]	$(P \lhd c \rhd Q) \setminus S \equiv (P \setminus S) \lhd c \rhd (Q \setminus S)$
[hidingLaw-9]	$((a \rightarrow \mathit{Skip}) \Box (b \rightarrow \mathit{Skip})) \setminus \{a\} \equiv (\mathit{Skip} \Box (b \rightarrow \mathit{Skip}))$
[hidingLaw-10]	$((a \rightarrow P) \Box (\mathit{Wait}\ n;\ Q)) \setminus \{a\} \equiv P \setminus \{a\}$
[hidingLaw-11]	$(P;\ Q) \setminus S \equiv (P \setminus S);\ (Q \setminus S)$
[hidingLaw-12]	$\mathit{Chaos} \setminus S \equiv \mathit{Chaos}$
[hidingLaw-13]	$(x := e) \setminus S \equiv (x := e)$
[hidingLaw-14]	$a \notin S \wedge b \notin S \Rightarrow$ $((a \rightarrow P) \Box (b \rightarrow Q)) \setminus S \equiv (a \rightarrow (P \setminus S)) \Box (b \rightarrow (Q \setminus S))$

All non-trivial.

4 Slotted-*Circus*—CTA Incarnation

This section presents the CTA theory of [She06, pp27–88] in the slotted-*Circus* framework:

4.1 Observational Variables

In CTA, a history is simply an event trace *str* so the history type-constructor for CTA is

[CTA:HIST] $\mathcal{CTA}\ E \mathrel{\widehat=} E^*$

4.2 Required Definitions and Proofs

4.2.1 Defining $acc_{\mathcal{CTA}}$

[CTA:ACC:sig] $acc_{\mathcal{CTA}} : E^* \rightarrow \mathbb{P}\, E$
[CTA:ACC:def] $acc(str) \mathrel{\widehat=} elems(str)$

4.2.2 Defining $EQVTRC_{\mathcal{CTA}}$

[CTA:ET:sig] $EQVTRC_{\mathcal{CTA}} : E^* \leftrightarrow \mathcal{CTA}\ E$
[CTA:ET:def] $EQVTRC(tr, str) \mathrel{\widehat=} tr = str$

Law:

[CTA:ET:elems] $EQVTRC(tr, str) \Rightarrow elems(tr) = acc(str)$

Proof:

$EQVTRC(tr, str)$
$\equiv$ “ [CTA:ET:defs] ”
$tr = str$
$\Rightarrow$ “ *elems* is a function ”
$elems(tr) = elems(str)$
$\equiv$ “ [CTA:ACC:def] ”
$elems(tr) = acc(str)$

4.2.3 Defining $hnull_{\mathcal{CTA}}$

[CTA:HN:sig] $hnull_{\mathcal{CTA}} : E^*$
[CTA:HN:def] $hnull \mathrel{\widehat=} \langle\rangle$

Law:

[CTA:HN:null] $acc(hnull) = \{\}$

Proof:

$$
\begin{array}{ll}
 & acc(hnull) \\
= & \quad \text{“ [CTA:HN:def] ”} \\
 & acc(\langle\rangle) \\
= & \quad \text{“ [CTA:ACC:def]:p40 ”} \\
 & elems(\langle\rangle) \\
= & \quad \text{“ defn. } elems \text{ ”} \\
 & \{\}
\end{array}
$$

4.2.4 Defining $\preceq_{\mathcal{CTA}}$

[CTA:pfx:sig] $\preceq_{\mathcal{CTA}}: E^* \leftrightarrow E^*$

[CTA:pfx:def] $str_1 \preceq str_2 \mathrel{\widehat=} str_1 \leq str_2$

Law:

[CTA:pfx:refl] $str \preceq str = \textsc{True}$

Proof:

$$
\begin{array}{ll}
 & str \preceq str \\
\equiv & \quad \text{“ [CTA:pfx:def] ”} \\
 & str \leq str \\
\equiv & \quad \text{“ sequence } \leq \text{ is reflexive ”} \\
 & \textsc{True}
\end{array}
$$

Law:

[CTA:pfx:trans] $str_1 \preceq str_2 \land str_2 \preceq str_3 \Rightarrow str_1 \preceq str_3$

Proof:

$$
\begin{array}{ll}
 & str_1 \preceq str_2 \land str_2 \preceq str_3 \\
\equiv & \quad \text{“ [CTA:pfx:def] ”} \\
 & str_1 \leq str_2 \land str_2 \leq str_3 \\
\Rightarrow & \quad \text{“ sequence } \leq \text{ is transitive ”} \\
 & str_1 \leq str_3 \\
\equiv & \quad \text{“ [CTA:pfx:def], backwards ”} \\
 & str_1 \preceq str_3
\end{array}
$$

Law:

[CTA:pfx:anti-sym] $str_1 \preceq str_2 \land str_2 \preceq str_1 \Rightarrow str_1 = str_2$

Proof:

$$
\begin{array}{ll}
 & str_1 \preceq str_2 \wedge str_2 \preceq str_1 \Rightarrow str_1 = str_2 \\
\equiv & \quad \text{“ [CTA:pfx:def] ”} \\
 & str_1 \leq str_2 \wedge str_2 \leq str_1 \Rightarrow str_1 = str_2 \\
\equiv & \quad \text{“ string prefix order is anti-symmetric ”} \\
 & \mathbf{true}
\end{array}
$$

Law:

[CTA:SN:pfx] $\quad hnull \preceq str$

Proof:

$$
\begin{array}{ll}
 & hnull \preceq str \\
\equiv & \quad \text{“ [CTA:HN:def] ”} \\
 & \langle\rangle \preceq str \\
\equiv & \quad \text{“ [CTA:pfx:def] ”} \\
 & \langle\rangle \leq str \\
\equiv & \quad \text{“ property of } \langle\rangle \text{ and } \leq \text{ ”} \\
 & \textsc{True}
\end{array}
$$

Law:

[CTA:ET:pfx] $\quad str_1 \preceq str_2 \Rightarrow \exists\, tr_1, tr_2 \bullet EQVTRC(tr_1, str_1) \wedge EQVTRC(tr_2, str_2) \wedge tr_1 \leq tr_2$

Proof (first step):

$$
\begin{array}{ll}
 & str_1 \preceq str_2 \\
\equiv & \quad \text{“ [CTA:pfx:def] ”} \\
 & str_1 \leq str_2 \qquad \text{[CTA:ET:pfx:hyp]}
\end{array}
$$

Proof (second step)—assume [CTA:ET:pfx:hyp]:

$$
\begin{array}{ll}
 & \exists\, tr_1, tr_2 \bullet EQVTRC(tr_1, str_1) \wedge EQVTRC(tr_2, str_2) \wedge tr_1 \leq tr_2 \\
\equiv & \quad \text{“ [CTA:ET:def]:p40 ”} \\
 & \exists\, tr_1, tr_2 \bullet tr_1 = str_1 \wedge tr_2 = str_2 \wedge tr_1 \leq tr_2 \\
\equiv & \quad \text{“ one-point rule ”} \\
 & str_1 \leq str_2 \\
\equiv & \quad \text{“ By hypothesis: [CTA:ET:pfx:hyp] ”} \\
 & \textsc{True}
\end{array}
$$

4.2.5 Defining $sadd_{\mathcal{CTA}}$

We also need to have the notion of adding and subtracting slots, obeying laws, some obvious, the others not so:

$$\text{[CTA:sadd:sig]} \qquad sadd_{\mathcal{CTA}} : E^* \times E^* \rightarrow E^*$$
$$\text{[CTA:sadd:def]} \qquad sadd(str_1, str_2) \mathrel{\widehat{=}} str_1 \frown str_2$$

Law:

$$\text{[CTA:sadd:events]} \qquad acc(sadd(str_1, str_2) = acc(str_1) \cup acc(str_2)$$

Proof:

$$\begin{array}{ll} & acc(sadd(str_1, str_2)) \\ = & \quad \text{“ [CTA:sadd:def] ”} \\ & acc(str_1 \frown str_2) \\ = & \quad \text{“ [CTA:ACC:def]:p40 ”} \\ & elems(str_1 \frown str_2) \\ = & \quad \text{“ } elems \text{ homomorphism ”} \\ & elems(str_1) \cup elems(str_2) \\ = & \quad \text{“ [CTA:ACC:def]:p40, backwards ”} \\ & acc(str_1) \cup acc(str_2) \end{array}$$

Law:

$$\text{[CTA:sadd:unit]} \qquad sadd(str_1, str_2) = str_1 \equiv (str_2 = hnull)$$

Proof:

$$\begin{array}{ll} & sadd(str_1, str_2) = str_1 \\ \equiv & \quad \text{“ [CTA:sadd:def]:p43 ”} \\ & str_1 \frown str_2 = str_1 \\ \equiv & \quad \text{“ seq. prop ”} \\ & str_2 = \langle\rangle \\ \equiv & \quad \text{“ [CTA:SN:def]:p}\textbf{??}\text{, backwards ”} \\ & str_2 = hnull \end{array}$$

Law:

$$\text{[CTA:sadd:assoc]} \qquad sadd(str_1, sadd(str_2, str_3)) = sadd(sadd(str_1, str_2), str_3)$$

Proof:

$$
\begin{array}{ll}
 & sadd(str_1, sadd(str_2, str_3,)) \\
= & \quad \text{“ [CTA:sadd:def]:p43 ”} \\
 & sadd(str_1, str_2 \frown str_3) \\
= & \quad \text{“ [CTA:sadd:def]:p43 ”} \\
 & str_1 \frown (str_2 \frown str_3)) \\
= & \quad \text{“ } \frown \text{ assoc. ”} \\
 & (str_1 \frown str_2) \frown str_3 \\
= & \quad \text{“ [CTA:sadd:def]:p43, backwards ”} \\
 & sadd(str_1 \frown str_2), str_3) \\
= & \quad \text{“ [CTA:sadd:def]:p43, backwards ”} \\
 & sadd(sadd(str_1), str_2), str_3)
\end{array}
$$

Law:

$$\text{[CTA:sadd:prefix]} \qquad str \preceq sadd(str, str)$$

Proof:

$$
\begin{array}{ll}
 & str) \preceq sadd(str, str') \\
\equiv & \quad \text{“ [CTA:sadd:def]:p43 ”} \\
 & str \preceq str \frown str' \\
\equiv & \quad \text{“ [CTA:pfx:def]:p41 ”} \\
 & str \leq str \frown str' \\
\equiv & \quad \text{“ seq property ”} \\
 & \textsc{True}
\end{array}
$$

4.2.6 Defining $ssub_{\mathcal{CTA}}$

We also need to have the notion of adding and subtracting slots, obeying laws, some obvious, the others not so:

$$
\begin{array}{ll}
\text{[CTA:ssub:sig]} & ssub_{\mathcal{CTA}} : E^* \times E^* \nrightarrow E^* \\
\text{[CTA:ssub:def]} & ssub(str_1, str_2) \mathrel{\widehat=} str_1 - str_2
\end{array}
$$

Law:

$$\text{[CTA:ssub:pre]} \qquad \text{pre } ssub(str_1, str_2) = str_2 \preceq str_1$$

Here we want to show that the pre-condition above implies the definition is well-defined. First we expand the precondition as supplied:

$$
\begin{array}{ll}
 & \text{pre } ssub(str_1, str_2) \\
\equiv & \quad \text{“ [CTA:ssub:pre] ”} \\
 & str_2 \preceq str_1 \\
\equiv & \quad \text{“ [CTA:pfx:def]:p41 ”} \\
 & str_2 \leq str_1
\end{array}
$$

We now compute the precondition of *ssub*:

$$
\begin{array}{ll}
 & \mathcal{D}(ssub(str_1, str_2)) \\
\equiv & \quad \text{“ [CTA:ssub:def] ”} \\
 & \mathcal{D}(str_1 - str_2) \\
\equiv & \quad \text{“ pre-condition for sequence-subtraction ”} \\
 & str_2 \leq str_1
\end{array}
$$

Law:

[CTA:ssub:events] $\quad str_2 \preceq str_1 \land s' = ssub(str_1, str_2) \Rightarrow acc(str_1) \setminus acc(str_2) \subseteq acc(s') \subseteq acc(str_1)$

We expand and simplify the antecedent:

$$
\begin{array}{ll}
 & str_2 \preceq str_1 \land s' = ssub(str_1, str_2) \\
\equiv & \quad \text{“ [CTA:pfx:def]:p41,[CTA:ssub:def]:p44 ”} \\
 & str_2 \leq str_1 \land s' = str_1 - str_2 \qquad \text{[CTA:ssub:events:hyp]}
\end{array}
$$

Now assume the above and look at consequent:

$$
\begin{array}{ll}
 & acc(str_1) \setminus acc(str_2) \subseteq acc(s') \subseteq acc(str_1) \\
\equiv & \quad \text{“ [CTA:ACC:def]:p40 ”} \\
 & elems(str_1) \setminus elems(str_2) \subseteq elems(s') \subseteq elems(str_1) \\
\equiv & \quad \text{“ [CTA:ssub:events:hyp] ”} \\
 & elems(str_1) \setminus elems(str_2) \subseteq elems(str_1 - str_2) \subseteq elems(str_1) \\
\equiv & \quad \text{“ properties of } elems \text{ w.r.t } -, \subseteq \text{ and } \setminus. \text{ ”} \\
 & \textsc{True} \land \textsc{True}
\end{array}
$$

Law:

[CTA:ssub:self] $\quad SSub(str, str) = hnull$

Proof:

$$
\begin{array}{ll}
 & SSub(str, (str,) \\
= & \quad \text{“ [CTA:ssub:def]:p44 ”} \\
 & str - str \\
= & \quad \text{“ property of seq. sub. ”} \\
 & \langle\rangle \\
= & \quad \text{“ [CTA:HN:def]:p40 ”} \\
 & hnull
\end{array}
$$

Law:

[CTA:ssub:nil] $\quad SSub(str, hnull) = str$

Proof:

$$
\begin{aligned}
& SSub(str, hnull) \\
= & \quad \text{“ [CTA:ssub:def]:p44,[CTA:HN:def]:p40 ”} \\
& str - \langle\rangle \\
= & \quad \text{“ property of seq. sub. ”} \\
& str
\end{aligned}
$$

Law:

[CTA:SSub:same] $str \preceq str'_a \land str \preceq str'_b \Rightarrow$
$ssub(str'_a, str, ref) = ssub(str'_b, str)$
$\equiv \ str'_a = str'_b$

Proof: the antecedent reduces by [CTA:pfx:def]:p41 to

$$str \leq str'_a \land str \leq str'_b$$

$$
\begin{aligned}
& ssub(str'_a, str) = ssub(str'_b, str) \\
\equiv & \quad \text{“ [CTA:ssub:def]:p44 ”} \\
& str'_a - str = str'_b - str \\
\equiv & \quad \text{“ } \sigma - \tau = \nu - \tau \equiv \sigma = \nu \text{ ”} \\
& str'_a = str'_b
\end{aligned}
$$

Law:

[CTA:ssub:subsub] $str_c \preceq str_a \land str_c \preceq str_b$
$\land\ str_b \preceq str_a$
$\Rightarrow ssub(ssub(str_a, str_c), ssub(str_b, str_c))$
$= ssub(str_a, str_b)$

Proof: The antecedent reduces to:

$$str_c \leq str_a \land str_c \leq str_b \land str_b \leq str_a$$

$$
\begin{aligned}
& ssub(ssub(str_a, str_c), ssub(str_b, str_c)) \\
= & \quad \text{“ [CTA:ssub:def]:p44 ”} \\
& ssub(str_a - str_c, str_b - str_c) \\
= & \quad \text{“ [CTA:ssub:def]:p44 ”} \\
& (str_a - str_c) - (str_b - str_c) \\
= & \quad \text{“ antecedents, and sequence subtraction property ”} \\
& str_a - str_b \\
= & \quad \text{“ [CTA:ssub:def]:p44, backwards ”} \\
& ssub(str_a, str_b)
\end{aligned}
$$

Law:

$$\text{[CTA:sadd:ssub]} \qquad str \preceq str' \Rightarrow sadd(str, ssub(str', str)) = str'$$

First, simplify the antecedent:

$$\begin{array}{ll} & str \preceq str' \\ \equiv & \quad \text{“ [CTA:pfx:def]:p41 ”} \\ & str \leq str' \qquad \text{[CTA:sadd:ssub:hyp]} \end{array}$$

Now, the consequent:

$$\begin{array}{ll} & sadd(str, ssub(str', str)) \\ = & \quad \text{“ [CTA:ssub:def]:p44 ”} \\ & sadd(str, str' - str) \\ = & \quad \text{“ [CTA:sadd:def]:p43 ”} \\ & str \frown (str' - str) \\ = & \quad \text{“ law of sequence subtraction, [CTA:sadd:ssub:hyp] ”} \\ & str' \end{array}$$

Law:

$$\text{[CTA:ssub:sadd]} \qquad ssub(sadd(str_1, str_2), str_1) = str_2$$

Proof:

$$\begin{array}{ll} & ssub(sadd(str_1, str_2), str_1) \\ = & \quad \text{“ [CTA:sadd:def]:p43 ”} \\ & ssub(str_1 \frown str_2, str_1) \\ = & \quad \text{“ [CTA:ssub:def]:p44 ”} \\ & (str_1 \frown str_2) - str_1 \\ = & \quad \text{“ law of sequences ”} \\ & str_2 \end{array}$$

4.2.7 Defining $ssync_{\mathcal{CTA}}$

$$\text{[CTA:SNC:sig]} \qquad ssync_{\mathcal{CTA}} : \mathbb{P}\,E \rightarrow E^* \times E^* \rightarrow \mathbb{P}(E^*)$$

We would like the following to hold:

$$ssync(cs)(str_1, str_2) \quad = \quad Sync(str_1, str_2, cs) \text{ as defined in [She06, p81]}$$

We shall follow the definition as given in [Ros97, pp69–70]:

$$
\begin{array}{rcl}
\textsf{[CTA:SNC:sym]} & & ssync(cs)(s_1, s_2) = ssync(cs)(s_2, s_1) \\
\textsf{[CTA:SNC:def]} & & \\
ssync(cs)(\langle\rangle, \langle\rangle) & \mathrel{\widehat=} & \{\langle\rangle\} \\
ssync(cs)(\langle\rangle, \langle a\rangle) & \mathrel{\widehat=} & \emptyset \lhd a \in cs \rhd \{\langle a\rangle\} \\
ssync(cs)(a \mathbin{⨾} s, b \mathbin{⨾} t) & \mathrel{\widehat=} & \\
a \notin cs \land b \notin cs & \Rightarrow & (a⨾)(ssync(cs)(s, b⨾t)) \cup (b⨾)(ssync(cs)(a⨾s, t)) \\
a \notin cs \land b \in cs & \Rightarrow & (a⨾)(ssync(cs)(s, b⨾t)) \\
a \in cs \land b \notin cs & \Rightarrow & (b⨾)(ssync(cs)(a⨾s, t)) \\
a \in cs \land b \in cs & \Rightarrow & \emptyset \lhd a \neq b \rhd (a⨾)(ssync(cs)(s, t))
\end{array}
$$

The following laws need proving:

$$
\begin{array}{rl}
\textsf{[CTA:SNC:one]} & \forall s' \in ssync(cs)(s_1, hnull) \bullet acc(s') \subseteq acc(s_1) \setminus cs \\
\textsf{[CTA:SNC:only]} & s' \in acc(ssync(cs)(s_1, s_2)) \Rightarrow acc(s') \subseteq acc(s_1) \cup acc(s_2) \\
\textsf{[CTA:SNC:sync]} & s' \in acc(ssync(cs)(s_1, s_2)) \Rightarrow cs \cap acc(s') \subseteq cs \cap (acc(s_1) \cap acc(s_2)) \\
\textsf{[CTA:SNC:assoc]} & syncset(cs)(s_1)(ssync(cs)(s_2, s_3)) = syncset(cs)(s_3)(ssync(cs)(s_1, s_2))
\end{array}
$$

4.2.8 Defining $shide_{\mathcal{CTA}}$

Finally we need to specify how to hide events in a slot:

$$
\begin{array}{rl}
\textsf{[CTA:SHid:sig]} & shide_{\mathcal{CTA}} : \mathbb{P}\, E \to E^* \to E^* \\
\textsf{[CTA:SHid:def]} & shide(hdn)str \mathrel{\widehat=} str \setminus hdn
\end{array}
$$

Law:

$$
\textsf{[CTA:SHid:evts]} \qquad acc(shide(hdn)str) = acc(str) \setminus hdn
$$

Proof:

$$
\begin{array}{cl}
 & acc(shide(hdn)str) \\
= & \quad \text{“ [CTA:SHid:def] ”} \\
 & acc(str \setminus hdn) \\
= & \quad \text{“ [CTA:ACC:def]:p40 ”} \\
 & elems(str \setminus hdn) \\
= & \quad \text{“ sequence property ”} \\
 & elems(str) \setminus hdn \\
= & \quad \text{“ [CTA:ACC:def]:p40, backwards ”} \\
 & acc(str) \setminus hdn
\end{array}
$$

5 Slotted-*Circus*—MSA Incarnation

This section presents an incarnation based on the notion of an event history being a multiset (bag) of events.

5.1 Observational Variables

In MSA, a history is simply an event bag so the history type-constructor for MSA is

[MSA:HIST] $\mathcal{MSA}\ E \mathrel{\widehat{=}} E \nrightarrow \mathbb{N}_1$

5.2 Required Definitions and Proofs

5.2.1 Defining $acc_{\mathcal{MSA}}$

[MSA:ACC:sig] $acc_{\mathcal{MSA}} : E^* \rightarrow \mathbb{P}\, E$

[MSA:ACC:def] $acc(bag) \mathrel{\widehat{=}} dom(bag)$

5.2.2 Defining $EQVTRC_{\mathcal{MSA}}$

[MSA:ET:sig] $EQVTRC_{\mathcal{MSA}} : E^* \leftrightarrow \mathcal{MSA}\ E$

[MSA:ET:def] $EQVTRC(tr, bag) \mathrel{\widehat{=}} items(tr) = bag$

Law:

[MSA:ET:elems] $EQVTRC(tr, bag) \Rightarrow elems(tr) = acc(bag)$

Proof:

$EQVTRC(tr, bag)$

$\equiv$ “ [MSA:ET:defs] ”

$items(tr) = bag$

$\Rightarrow$ “ dom is a function ”

$dom(items(tr)) = dom(bag)$

$\equiv$ “ [MSA:ACC:def] ”

$dom(items(tr)) = acc(bag)$

$\equiv$ “ $dom \circ items = elems$ ”

$elems(tr) = acc(bag)$

5.2.3 Defining $hnull_{\mathcal{MSA}}$

[MSA:HN:sig] $hnull_{\mathcal{MSA}} : E^*$

[MSA:HN:def] $hnull \mathrel{\widehat{=}} [\![\,]\!]$

Law:

[MSA:HN:null] $acc(hnull) = \{\}$

Proof:

$$
\begin{array}{ll}
 & acc(hnull) \\
= & \quad \text{“ [MSA:HN:def] ”} \\
 & acc([\![]\!])) \\
= & \quad \text{“ [MSA:ACC:def]:p49 ”} \\
 & ran([\![]\!]) \\
= & \quad \text{“ defn. } ran, [\![]\!] \text{ ”} \\
 & \{\}
\end{array}
$$

5.2.4 Defining $\preceq_{\mathcal{MSA}}$

[MSA:pfx:sig] $\preceq_{\mathcal{MSA}}: E^* \leftrightarrow E^*$

[MSA:pfx:def] $bag_1 \preceq bag_2 \mathrel{\widehat{=}} bag_1 \sqsubseteq bag_2$

Law:

[MSA:pfx:refl] $bag \preceq bag = \textsc{True}$

Proof:

$$
\begin{array}{ll}
 & bag \preceq bag \\
\equiv & \quad \text{“ [MSA:pfx:def] ”} \\
 & bag \sqsubseteq bag \\
\equiv & \quad \text{“ bag } \sqsubseteq \text{ is reflexive ”} \\
 & \textsc{True}
\end{array}
$$

Law:

[MSA:pfx:trans] $bag_1 \preceq bag_2 \land bag_2 \preceq bag_3 \Rightarrow bag_1 \preceq bag_3$

Proof:

$$
\begin{array}{ll}
 & bag_1 \preceq bag_2 \land bag_2 \preceq bag_3 \\
\equiv & \quad \text{“ [MSA:pfx:def] ”} \\
 & bag_1 \sqsubseteq bag_2 \land bag_2 \sqsubseteq bag_3 \\
\Rightarrow & \quad \text{“ bag } \sqsubseteq \text{ is transitive ”} \\
 & bag_1 \sqsubseteq bag_3 \\
\equiv & \quad \text{“ [MSA:pfx:def], backwards ”} \\
 & bag_1 \preceq bag_3
\end{array}
$$

Law:

[MSA:pfx:anti-sym] $bag_1 \preceq bag_2 \land bag_2 \preceq bag_1 \Rightarrow bag_1 = bag_2$

Proof:

$$
\begin{array}{ll}
 & bag_1 \preceq bag_2 \land bag_2 \preceq bag_1 \Rightarrow bag_1 = bag_2 \\
\equiv & \quad \text{“ [MSA:pfx:def] ”} \\
 & bag_1 \sqsubseteq bag_2 \land bag_2 \sqsubseteq bag_1 \Rightarrow bag_1 = bag_2 \\
\equiv & \quad \text{“ sub-bag relation is anti-symmetric ”} \\
 & \mathbf{true}
\end{array}
$$

Law:

[MSA:SN:pfx] $hnull \preceq bag$

Proof:

$$
\begin{array}{ll}
 & hnull \preceq bag \\
\equiv & \quad \text{“ [MSA:HN:def] ”} \\
 & [\![]\!] \preceq bag \\
\equiv & \quad \text{“ [MSA:pfx:def] ”} \\
 & [\![]\!] \sqsubseteq bag \\
\equiv & \quad \text{“ property of } [\![]\!] \text{ and } \sqsubseteq \text{ ”} \\
 & \textsc{True}
\end{array}
$$

Law, which in this case can be strengthened to an equivalence:

[MSA:ET:pfx]
$$
\begin{array}{l}
bag_1 \preceq bag_2 \\
\equiv \\
\quad \exists\, tr_1, tr_2 \bullet EQVTRC(tr_1, bag_1) \land EQVTRC(tr_2, bag_2) \land tr_1 \leq tr_2
\end{array}
$$

Proof:

$$
\begin{array}{cl}
 & bag_1 \preceq bag_2 \\
\equiv & \text{“ [MSA:pfx:def]:p50 ”} \\
 & bag_1 \sqsubseteq bag_2 \\
\equiv & \text{“ bag property ”} \\
 & \exists\, bag_\Delta \bullet bag_2 = bag_1 \oplus bag_\Delta \\
\equiv & \text{“ bag property: } \forall\, bag \bullet \exists\, tr \bullet items(tr) = bag \text{ ”} \\
 & \exists\, bag_\Delta, tr_\Delta, tr_1, \bullet\ bag_2 = bag_1 \oplus bag_\Delta \land items(tr_\Delta) = bag_\Delta \land items(tr_1) = bag_1 \\
\equiv & \text{“ One-point rule backwards } tr_2 = tr_1 \frown tr_\Delta \text{ ”} \\
 & \exists\, bag_\Delta, tr_\Delta, tr_1, tr_2 \bullet bag_2 = bag_1 \oplus bag_\Delta \land items(tr_\Delta) = bag_\Delta \land items(tr_1) = bag_1 \land tr_2 = tr_1 \frown tr_\Delta \\
\equiv & \text{“ One-point rule } bag_\Delta\text{, Liebniz } bag_1 \text{ ”} \\
 & \exists\, tr_\Delta, tr_1, tr_2 \bullet bag_2 = items(tr_1) \oplus items(tr_\Delta) \land items(tr_1) = bag_1 \land tr_2 = tr_1 \frown tr_\Delta \\
\equiv & \text{“ } items \text{ is a sequence homomorphism ”} \\
 & \exists\, tr_\Delta, tr_1, tr_2 \bullet bag_2 = items(tr_2) \land bag_1 = items(tr_1) \land tr_2 = tr_1 \frown tr_\Delta \\
\equiv & \text{“ sequence property ”} \\
 & \exists\, tr_\Delta, tr_1, tr_2 \bullet bag_2 = items(tr_2) \land bag_1 = items(tr_1) \land tr_\Delta = tr_2 - tr_1 \\
\equiv & \text{“ One point rule: } tr_\Delta\text{, requires definedness of } tr_2 - tr_1 \text{ ”} \\
 & \exists\, tr_1, tr_2 \bullet bag_2 = items(tr_2) \land bag_1 = items(tr_1) \land tr_1 \leq tr_2 \\
\equiv & \text{“ [MSA:ET:def]:p49, backwards ”} \\
 & \exists\, tr_1, tr_2 \bullet EQVTRC(tr_2, bag_2) \land EQVTRC(tr_1, bag_1) \land tr_1 \leq tr_2
\end{array}
$$

5.2.5 Defining $sadd_{\mathcal{MSA}}$

We also need to have the notion of adding and subtracting slots, obeying laws, some obvious, the others not so:

[MSA:sadd:sig]	$sadd_{\mathcal{MSA}} : E^* \times E^* \to E^*$
[MSA:sadd:def]	$sadd(bag_1, bag_2) \mathrel{\widehat=} bag_1 \oplus bag_2$

Law:

[MSA:sadd:events]	$acc(sadd(bag_1, bag_2) = acc(bag_1) \cup acc(bag_2)$

Proof:

$$
\begin{array}{cl}
 & acc(sadd(bag_1, bag_2)) \\
= & \text{“ [MSA:sadd:def] ”} \\
 & acc(bag_1 \oplus bag_2) \\
= & \text{“ [MSA:ACC:def]:p49 ”} \\
 & dom(bag_1 \oplus bag_2) \\
= & \text{“ } elems \text{ homomorphism ”} \\
 & dom(bag_1) \cup dom(bag_2) \\
= & \text{“ [MSA:ACC:def]:p49, backwards ”} \\
 & acc(bag_1) \cup acc(bag_2)
\end{array}
$$

Law:

$$[\mathsf{MSA{:}sadd{:}unit}] \qquad sadd(bag_1, bag_2) = bag_1 \equiv (bag_2 = hnull)$$

Proof:

$$\begin{array}{ll} & sadd(bag_1, bag_2) = bag_1 \\ \equiv & \quad \text{“ [MSA:sadd:def]:p52 ”} \\ & bag_1 \oplus bag_2 = bag_1 \\ \equiv & \quad \text{“ seq. prop ”} \\ & bag_2 = [\![]\!] \\ \equiv & \quad \text{“ [MSA:SN:def]:p??, backwards ”} \\ & bag_2 = hnull \end{array}$$

Law:

$$[\mathsf{MSA{:}sadd{:}assoc}] \qquad sadd(bag_1, sadd(bag_2, bag_3)) = sadd(sadd(bag_1, bag_2), bag_3)$$

Proof:

$$\begin{array}{ll} & sadd(bag_1, sadd(bag_2, bag_3,)) \\ = & \quad \text{“ [MSA:sadd:def]:p52 ”} \\ & sadd(bag_1, bag_2 \oplus bag_3) \\ = & \quad \text{“ [MSA:sadd:def]:p52 ”} \\ & bag_1 \oplus (bag_2 \oplus bag_3)) \\ = & \quad \text{“ } \oplus \text{ assoc. ”} \\ & (bag_1 \oplus bag_2) \oplus bag_3 \\ = & \quad \text{“ [MSA:sadd:def]:p52, backwards ”} \\ & sadd(bag_1 \oplus bag_2), bag_3) \\ = & \quad \text{“ [MSA:sadd:def]:p52, backwards ”} \\ & sadd(sadd(bag_1), bag_2), bag_3) \end{array}$$

Law:

$$[\mathsf{MSA{:}sadd{:}prefix}] \qquad bag \preceq sadd(bag, bag)$$

Proof:

$$\begin{array}{ll} & bag \preceq sadd(bag, bag') \\ \equiv & \quad \text{“ [MSA:sadd:def]:p52 ”} \\ & bag \preceq bag \oplus bag' \\ \equiv & \quad \text{“ [MSA:pfx:def]:p50 ”} \\ & bag \sqsubseteq bag \oplus bag' \\ \equiv & \quad \text{“ bag property ”} \\ & \textsc{True} \end{array}$$

5.2.6 Defining $ssub_{\mathcal{MSA}}$

We also need to have the notion of adding and subtracting slots, obeying laws, some obvious, the others not so:

$$\text{[MSA:ssub:sig]} \qquad ssub_{\mathcal{MSA}} : E^* \times E^* \nrightarrow E^*$$
$$\text{[MSA:ssub:def]} \qquad ssub(bag_1, bag_2) \mathrel{\widehat{=}} bag_1 \ominus bag_2$$

Law:

$$\text{[MSA:ssub:pre]} \qquad \text{pre } ssub(bag_1, bag_2) = bag_2 \preceq bag_1$$

Here we want to show that the pre-condition above implies the definition is well-defined. First we expand the precondition as supplied:

$$\begin{array}{ll} & \text{pre } ssub(bag_1, bag_2) \\ \equiv & \quad \text{“ [MSA:ssub:pre] ”} \\ & bag_2 \preceq bag_1 \\ \equiv & \quad \text{“ [MSA:pfx:def]:p50 ”} \\ & bag_2 \sqsubseteq bag_1 \end{array}$$

We now compute the precondition of $ssub$:

$$\begin{array}{ll} & \mathcal{D}(ssub(bag_1, bag_2)) \\ \equiv & \quad \text{“ [MSA:ssub:def] ”} \\ & \mathcal{D}(bag_1 \ominus bag_2) \\ \equiv & \quad \text{“ pre-condition for sequence-subtraction ”} \\ & bag_2 \sqsubseteq bag_1 \end{array}$$

Law:

$$\text{[MSA:ssub:events]} \qquad bag_2 \preceq bag_1 \land s' = ssub(bag_1, bag_2) \\ \Rightarrow acc(bag_1) \setminus acc(bag_2) \subseteq acc(s') \subseteq acc(bag_1)$$

We expand and simplify the antecedent:

$$\begin{array}{ll} & bag_2 \preceq bag_1 \land s' = ssub(bag_1, bag_2) \\ \equiv & \quad \text{“ [MSA:pfx:def]:p50,[MSA:ssub:def]:p54 ”} \\ & bag_2 \sqsubseteq bag_1 \land s' = bag_1 \ominus bag_2 \qquad \text{[MSA:ssub:events:hyp]} \end{array}$$

Now assume the above and look at consequent:

$$\begin{array}{ll} & acc(bag_1) \setminus acc(bag_2) \subseteq acc(s') \subseteq acc(bag_1) \\ \equiv & \quad \text{“ [MSA:ACC:def]:p49 ”} \\ & dom(bag_1) \setminus dom(bag_2) \subseteq dom(s') \subseteq dom(bag_1) \\ \equiv & \quad \text{“ [MSA:ssub:events:hyp] ”} \\ & dom(bag_1) \setminus dom(bag_2) \subseteq dom(bag_1 \ominus bag_2) \subseteq dom(bag_1) \\ \equiv & \quad \text{“ properties of } dom \text{ w.r.t } \ominus, \subseteq \text{ and } \setminus. \text{ ”} \\ & \textsc{True} \land \textsc{True} \end{array}$$

Law:

[MSA:ssub:self] $SSub(bag, bag) = hnull$

Proof:

$$
\begin{aligned}
& SSub(bag, bag) \\
= & \quad \text{“ [MSA:ssub:def]:p54 ”} \\
& bag \ominus bag \\
= & \quad \text{“ property of seq. sub. ”} \\
& [\![]\!] \\
= & \quad \text{“ [MSA:HN:def]:p49 ”} \\
& hnull
\end{aligned}
$$

Law:

[MSA:ssub:nil] $SSub(bag, hnull) = bag$

Proof:

$$
\begin{aligned}
& SSub(bag, hnull) \\
= & \quad \text{“ [MSA:ssub:def]:p54,[MSA:HN:def]:p49 ”} \\
& bag \ominus [\![]\!] \\
= & \quad \text{“ property of seq. sub. ”} \\
& bag
\end{aligned}
$$

Law:

[MSA:SSub:same]
$$
\begin{aligned}
& bag \preceq bag'_a \land bag \preceq bag'_b \Rightarrow \\
& \quad ssub(bag'_a, bag, ref) = ssub(bag'_b, bag) \\
& \quad\quad \equiv \; bag'_a = bag'_b
\end{aligned}
$$

Proof: the antecedent reduces by [MSA:pfx:def]:p50 to

$$bag \sqsubseteq bag'_a \land bag \sqsubseteq bag'_b$$

$$
\begin{aligned}
& ssub(bag'_a, bag) = ssub(bag'_b, bag) \\
\equiv & \quad \text{“ [MSA:ssub:def]:p54 ”} \\
& bag'_a \ominus bag = bag'_b \ominus bag \\
\equiv & \quad \text{“ } \sigma \ominus \tau = \nu \ominus \tau \equiv \sigma = \nu \text{ ”} \\
& bag'_a = bag'_b
\end{aligned}
$$

Law:

[MSA:ssub:subsub]
$$
\begin{aligned}
& bag_c \preceq bag_a \land bag_c \preceq bag_b \\
& \land\, bag_b \preceq bag_a \\
& \quad \Rightarrow ssub(ssub(bag_a, bag_c), ssub(bag_b, bag_c)) \\
& \quad\quad = ssub(bag_a, bag_b)
\end{aligned}
$$

Proof: The antecedent reduces to:

$$bag_c \sqsubseteq bag_a \land bag_c \sqsubseteq bag_b \land bag_b \sqsubseteq bag_a$$

$$
\begin{array}{ll}
 & ssub(ssub(bag_a, bag_c), ssub(bag_b, bag_c)) \\
= & \quad \text{“ [MSA:ssub:def]:p54 ”} \\
 & ssub(bag_a \ominus bag_c, bag_b \ominus bag_c) \\
= & \quad \text{“ [MSA:ssub:def]:p54 ”} \\
 & (bag_a \ominus bag_c) \ominus (bag_b \ominus bag_c) \\
= & \quad \text{“ antecedents, and sequence subtraction property ”} \\
 & bag_a \ominus bag_b \\
= & \quad \text{“ [MSA:ssub:def]:p54, backwards ”} \\
 & ssub(bag_a, bag_b)
\end{array}
$$

Law:

$$\text{[MSA:sadd:ssub]} \qquad bag \preceq bag' \Rightarrow sadd(bag, ssub(bag', bag)) = bag'$$

First, simplify the antecedent:

$$
\begin{array}{ll}
 & bag \preceq bag' \\
\equiv & \quad \text{“ [MSA:pfx:def]:p50 ”} \\
 & bag \sqsubseteq bag' \qquad \text{[MSA:sadd:ssub:hyp]}
\end{array}
$$

Now, the consequent:

$$
\begin{array}{ll}
 & sadd(bag, ssub(bag', bag)) \\
= & \quad \text{“ [MSA:ssub:def]:p54 ”} \\
 & sadd(bag, bag' \ominus bag) \\
= & \quad \text{“ [MSA:sadd:def]:p52 ”} \\
 & bag \oplus (bag' \ominus bag) \\
= & \quad \text{“ law of bag subtraction, [MSA:sadd:ssub:hyp] ”} \\
 & bag'
\end{array}
$$

Law:

$$\text{[MSA:ssub:sadd]} \qquad ssub(sadd(bag_1, bag_2), bag_1) = bag_2$$

Proof:

$$
\begin{array}{ll}
 & ssub(sadd(bag_1, bag_2), bag_1) \\
= & \quad \text{“ [MSA:sadd:def]:p52 ”} \\
 & ssub(bag_1 \oplus bag_2, bag_1) \\
= & \quad \text{“ [MSA:ssub:def]:p54 ”} \\
 & (bag_1 \oplus bag_2) \ominus bag_1 \\
= & \quad \text{“ law of bags ”} \\
 & bag_2
\end{array}
$$

5.2.7 Defining $ssync_{\mathcal{MSA}}$

$$\text{[MSA:SNC:sig]} \qquad ssync_{\mathcal{MSA}} : \mathbb{P}\, E \to E^* \times E^* \to \mathbb{P}(E^*)$$

We simply work with bag restriction, removal sum and intersection, and return a singleton set:

$$\begin{array}{rcl}
\text{[MSA:SNC:sym]} & & ssync(cs)(bag_1, bag_2) = ssync(cs)(bag_2, bag_1) \\
\text{[MSA:SNC:def]} & & \\
ssync(cs)(bag_1, bag_2) & \widehat{=} & \{(cs \lhd (bag_1 \oplus bag_2)) \oplus (cs \lhd (bag_1 \cap bag_2))\} \\
\textbf{where} & & \cap \text{ is bag interesection}
\end{array}$$

The following laws need proving:

$$\begin{array}{rl}
\text{[MSA:SNC:one]} & \forall s' \in ssync(cs)(s_1, hnull) \bullet acc(s') \subseteq acc(s_1) \setminus cs \\
\text{[MSA:SNC:only]} & s' \in acc(ssync(cs)(s_1, s_2)) \Rightarrow acc(s') \subseteq acc(s_1) \cup acc(s_2) \\
\text{[MSA:SNC:sync]} & s' \in acc(ssync(cs)(s_1, s_2)) \Rightarrow cs \cap acc(s') \subseteq cs \cap (acc(s_1) \cap acc(s_2)) \\
\text{[MSA:SNC:assoc]} & syncset(cs)(s_1)(ssync(cs)(s_2, s_3)) = syncset(cs)(s_3)(ssync(cs)(s_1, s_2))
\end{array}$$

The first three can be strengthened to equalities.

5.2.8 Defining $shide_{\mathcal{MSA}}$

Finally we need to specify how to hide events in a slot:

$$\begin{array}{rl}
\text{[MSA:SHid:sig]} & shide_{\mathcal{MSA}} : \mathbb{P}\, E \to E^* \to E^* \\
\text{[MSA:SHid:def]} & shide(hdn)bag \mathrel{\widehat{=}} hdn \lhd bag
\end{array}$$

Law:

$$\text{[MSA:SHid:evts]} \qquad acc(shide(hdn)bag) = acc(bag) \setminus hdn$$

Proof:

$$\begin{array}{rl}
 & acc(shide(hdn)bag) \\
= & \quad \text{“ [MSA:SHid:def] ”} \\
 & acc(hdn \lhd bag) \\
= & \quad \text{“ [MSA:ACC:def]:p49 ”} \\
 & dom(hdn \lhd bag) \\
= & \quad \text{“ bag property ”} \\
 & dom(bag) \setminus hdn \\
= & \quad \text{“ [MSA:ACC:def]:p49, backwards ”} \\
 & acc(bag) \setminus hdn
\end{array}$$

6 Slotted-*Circus*—SA Incarnation

This section presents an incarnation based on the notion of an event history being a set of events.

6.1 Observational Variables

In SA, a history is simply an event set so the history type-constructor for SA is

[SA:HIST] $\mathcal{SA}\ E \mathrel{\widehat=} \mathbb{P}\, E$

6.2 Required Definitions and Proofs

6.2.1 Defining $acc_{\mathcal{SA}}$

[SA:ACC:sig] $acc_{\mathcal{SA}} : E^* \rightarrow \mathbb{P}\, E$
[SA:ACC:def] $acc(evts) \mathrel{\widehat=} evts$

6.2.2 Defining $EQVTRC_{\mathcal{SA}}$

[SA:ET:sig] $EQVTRC_{\mathcal{SA}} : E^* \leftrightarrow \mathcal{SA}\ E$
[SA:ET:def] $EQVTRC(tr, evts) \mathrel{\widehat=} elems(tr) = evts$

Law:

[SA:ET:elems] $EQVTRC(tr, evts) \Rightarrow elems(tr) = acc(evts)$

Proof:

$EQVTRC(tr, evts)$
$\equiv$ “ [SA:ET:defs] ”
$elems(tr) = evts$
$\equiv$ “ [SA:ACC:def] ”
$elems(tr) = acc(evts)$

6.2.3 Defining $hnull_{\mathcal{SA}}$

[SA:HN:sig] $hnull_{\mathcal{SA}} : \mathbb{P}\, E$
[SA:HN:def] $hnull \mathrel{\widehat=} \{\}$

Law:

[SA:HN:null] $acc(hnull) = \{\}$

Proof:

$$
\begin{array}{ll}
 & acc(hnull) \\
= & \quad \text{“ [SA:HN:def] ”} \\
 & acc(\{\})) \\
= & \quad \text{“ [SA:ACC:def]:p58 ”} \\
 & \{\}
\end{array}
$$

6.2.4 Defining $\preceq_{\mathcal{SA}}$

[SA:pfx:sig]	$\preceq_{\mathcal{SA}} : \mathbb{P}\, E \leftrightarrow \mathbb{P}\, E$
[SA:pfx:def]	$evts_1 \preceq evts_2 \mathrel{\widehat=} evts_1 \subseteq evts_2$

Law:

[SA:pfx:refl] $\quad evts \preceq evts = \textsc{True}$

Proof:

$$
\begin{array}{ll}
 & evts \preceq evts \\
\equiv & \quad \text{“ [SA:pfx:def] ”} \\
 & evts \subseteq evts \\
\equiv & \quad \text{“ } \subseteq \text{ is reflexive ”} \\
 & \textsc{True}
\end{array}
$$

Law:

[SA:pfx:trans] $\quad evts_1 \preceq evts_2 \land evts_2 \preceq evts_3 \Rightarrow evts_1 \preceq evts_3$

Proof:

$$
\begin{array}{ll}
 & evts_1 \preceq evts_2 \land evts_2 \preceq evts_3 \\
\equiv & \quad \text{“ [SA:pfx:def] ”} \\
 & evts_1 \subseteq evts_2 \land evts_2 \subseteq evts_3 \\
\Rightarrow & \quad \text{“ } \subseteq \text{ is transitive ”} \\
 & evts_1 \subseteq evts_3 \\
\equiv & \quad \text{“ [SA:pfx:def], backwards ”} \\
 & evts_1 \preceq evts_3
\end{array}
$$

Law:

[SA:pfx:anti-sym] $\quad evts_1 \preceq evts_2 \land evts_2 \preceq evts_1 \Rightarrow evts_1 = evts_2$

Proof:

$$
\begin{aligned}
& evts_1 \preceq evts_2 \land evts_2 \preceq evts_1 \Rightarrow evts_1 = evts_2 \\
\equiv & \quad \text{“ [SA:pfx:def] ”} \\
& evts_1 \subseteq evts_2 \land evts_2 \subseteq evts_1 \Rightarrow evts_1 = evts_2 \\
\equiv & \quad \text{“ subset relation is anti-symmetric ”} \\
& \mathbf{true}
\end{aligned}
$$

Law:

[SA:SN:pfx] $\quad hnull \preceq evts$

Proof:

$$
\begin{aligned}
& hnull \preceq evts \\
\equiv & \quad \text{“ [SA:HN:def] ”} \\
& \{\} \preceq evts \\
\equiv & \quad \text{“ [SA:pfx:def] ”} \\
& \{\} \subseteq evts \\
\equiv & \quad \text{“ property of \{\} and } \subseteq \text{ ”} \\
& \textsc{True}
\end{aligned}
$$

Law, which in this case can be strengthened to an equivalence:

[SA:ET:pfx] $\quad evts_1 \preceq evts_2$

$$
\equiv
$$

$$
\exists\, tr_1, tr_2 \bullet EQVTRC(tr_1, evts_1) \land EQVTRC(tr_2, evts_2) \land tr_1 \leq tr_2
$$

Proof (Rhs implies Lhs):

$$
\begin{aligned}
& \exists\, tr_1, tr_2 \bullet EQVTRC(tr_1, evts_1) \land EQVTRC(tr_2, evts_2) \land tr_1 \leq tr_2 \\
\equiv & \quad \text{“ [SA:ET:def]:p58, } s \leq t \Rightarrow elem(s) \subseteq elems(t) \text{ ”} \\
& \exists\, tr_1, tr_2 \bullet elems(tr_1) = evts_1 \land elems(tr_2) = evts_2 \land tr_1 \leq tr_2 \land elems(tr_1) \subseteq elems(tr_2) \\
\equiv & \quad \text{“ Liebniz ”} \\
& \exists\, tr_1, tr_2 \bullet elems(tr_1) = evts_1 \land elems(tr_2) = evts_2 \land tr_1 \leq tr_2 \land evts_1 \subseteq evts_2 \\
\equiv & \quad \text{“ Move last conjunct out of quantifier ”} \\
& (\exists\, tr_1, tr_2 \bullet elems(tr_1) = evts_1 \land elems(tr_2) = evts_2 \land tr_1 \leq tr_2) \land evts_1 \subseteq evts_2 \\
\Rightarrow & \quad \text{“ weaken conunct ”} \\
& evts_1 \subseteq evts_2
\end{aligned}
$$

(Lhs implies Rhs): Intuitively true, but difficult to prove (**???**). Note that this is the direction we require.

6.2.5 Defining $sadd_{\mathcal{SA}}$

We also need to have the notion of adding and subtracting slots, obeying laws, some obvious, the others not so:

[SA:sadd:sig] $sadd_{\mathcal{SA}} : \mathbb{P}\,E \times \mathbb{P}\,E \rightarrow \mathbb{P}\,E$

[SA:sadd:def] $sadd(evts_1, evts_2) \mathrel{\widehat=} evts_1 \cup evts_2$

Law:

[SA:sadd:events] $acc(sadd(evts_1, evts_2) = acc(evts_1) \cup acc(evts_2)$

Proof:

$acc(sadd(evts_1, evts_2))$
$=$ " [SA:sadd:def] "
$evts_1 \cup evts_2)$
$=$ " [SA:ACC:def]:p58, backwards "
$acc(evts_1) \cup acc(evts_2)$

Law:

[SA:sadd:unit] $sadd(evts_1, evts_2) = evts_1 \equiv (evts_2 = hnull)$

Not true — simplest counterexample is $evts_1 = evts_2$.
Law:

[SA:sadd:assoc] $sadd(evts_1, sadd(evts_2, evts_3)) = sadd(sadd(evts_1, evts_2), evts_3)$

Proof:

$sadd(evts_1, sadd(evts_2, evts_3,))$
$=$ " [SA:sadd:def]:p61 "
$sadd(evts_1, evts_2 \cup evts_3)$
$=$ " [SA:sadd:def]:p61 "
$evts_1 \cup (evts_2 \cup evts_3))$
$=$ " $\cup$ assoc. "
$(evts_1 \cup evts_2) \cup evts_3$
$=$ " [SA:sadd:def]:p61, backwards "
$sadd(evts_1 \cup evts_2), evts_3)$
$=$ " [SA:sadd:def]:p61, backwards "
$sadd(sadd(evts_1), evts_2), evts_3)$

Law:

[SA:sadd:prefix] $evts \preceq sadd(evts, evts)$

Proof:

$$\begin{array}{ll} & evts \preceq sadd(evts, evts') \\ \equiv & \quad \text{“ [SA:sadd:def]:p61 ”} \\ & evts \preceq evts \cup evts' \\ \equiv & \quad \text{“ [SA:pfx:def]:p59 ”} \\ & evts \subseteq evts \cup evts' \\ \equiv & \quad \text{“ set property ”} \\ & \textsc{True} \end{array}$$

6.2.6 Defining $ssub_{\mathcal{SA}}$

We also need to have the notion of adding and subtracting slots, obeying laws, some obvious, the others not so:

$$\begin{array}{ll} \text{[SA:ssub:sig]} & ssub_{\mathcal{SA}} : \mathbb{P}\, E \times \mathbb{P}\, E \rightarrow\!\!\!\!\!\!+\; \mathbb{P}\, E \\ \text{[SA:ssub:def]} & ssub(evts_1, evts_2) \mathrel{\widehat{=}} evts_1 \setminus evts_2 \end{array}$$

Law:

$$\text{[SA:ssub:pre]} \qquad \text{pre } ssub(evts_1, evts_2) = evts_2 \preceq evts_1$$

Here we want to show that the pre-condition above implies the definition is well-defined. First we expand the precondition as supplied:

$$\begin{array}{ll} & \text{pre } ssub(evts_1, evts_2) \\ \equiv & \quad \text{“ [SA:ssub:pre] ”} \\ & evts_2 \preceq evts_1 \\ \equiv & \quad \text{“ [SA:pfx:def]:p59 ”} \\ & evts_2 \subseteq evts_1 \end{array}$$

We now compute the precondition of $ssub$:

$$\begin{array}{ll} & \mathcal{D}(ssub(evts_1, evts_2)) \\ \equiv & \quad \text{“ [SA:ssub:def] ”} \\ & \mathcal{D}(evts_1 \setminus evts_2) \\ \equiv & \quad \text{“ pre-condition for sequence-subtraction ”} \\ & \textsc{True} \end{array}$$

We find that $SSub$ is in fact total.
Law:

$$\begin{array}{ll} \text{[SA:ssub:events]} & evts_2 \preceq evts_1 \land s' = ssub(evts_1, evts_2) \\ & \quad \Rightarrow acc(evts_1) \setminus acc(evts_2) \subseteq acc(s') \subseteq acc(evts_1) \end{array}$$

We expand and simplify the antecedent:

$$\begin{array}{ll} & evts_2 \preceq evts_1 \land s' = ssub(evts_1, evts_2) \\ \equiv & \quad \text{“ [SA:pfx:def]:p59,[SA:ssub:def]:p62 ”} \\ & evts_2 \subseteq evts_1 \land s' = evts_1 \setminus evts_2 \qquad \text{[SA:ssub:events:hyp]} \end{array}$$

Now assume the above and look at consequent:

$$
\begin{array}{ll}
 & acc(evts_1) \setminus acc(evts_2) \subseteq acc(s') \subseteq acc(evts_1) \\
\equiv & \quad \text{“ [SA:ACC:def]:p58 ”} \\
 & evts_1 \setminus evts_2 \subseteq s' \subseteq evts_1 \\
\equiv & \quad \text{“ [SA:ssub:events:hyp] ”} \\
 & evts_1 \setminus evts_2 \subseteq evts_1 \setminus evts_2 \subseteq evts_1 \\
\equiv & \quad \text{“ properties of } \setminus \text{ and } \subseteq. \text{ ”} \\
 & \textsc{True} \land \textsc{True}
\end{array}
$$

Law:

[SA:ssub:self] $\quad SSub(evts, evts) = hnull$

Proof:

$$
\begin{array}{ll}
 & SSub(evts, evts) \\
= & \quad \text{“ [SA:ssub:def]:p62 ”} \\
 & evts \setminus evts \\
= & \quad \text{“ property of set sub. ”} \\
 & \{\} \\
= & \quad \text{“ [SA:HN:def]:p58 ”} \\
 & hnull
\end{array}
$$

Law:

[SA:ssub:nil] $\quad SSub(evts, hnull) = evts$

Proof:

$$
\begin{array}{ll}
 & SSub(evts, hnull) \\
= & \quad \text{“ [SA:ssub:def]:p62,[SA:HN:def]:p58 ”} \\
 & evts \setminus \{\} \\
= & \quad \text{“ property of set sub. ”} \\
 & evts
\end{array}
$$

Law:

[SA:SSub:same] $\quad evts \preceq evts'_a \land evts \preceq evts'_b \Rightarrow$
$\qquad ssub(evts'_a, evts, ref) = ssub(evts'_b, evts)$
$\qquad\quad \equiv \; evts'_a = evts'_b$

Proof: the antecedent reduces by [SA:pfx:def]:p59 to

$$evts \subseteq evts'_a \land evts \subseteq evts'_b$$

$$
\begin{array}{ll}
 & ssub(evts'_a, evts) = ssub(evts'_b, evts) \\
\equiv & \quad \text{“ [SA:ssub:def]:p62 ”} \\
 & evts'_a \setminus evts = evts'_b \setminus evts \\
\equiv & \quad \text{“ } S \setminus T = U \setminus T \equiv S = U, \text{ when } T \subseteq S, U. \text{ ”} \\
 & evts'_a = evts'_b
\end{array}
$$

Here we see that the precondition is important, as this property does not hold otherwise.
Law:

$$
\begin{array}{ll}
\text{[SA:ssub:subsub]} & evts_c \preceq evts_a \wedge evts_c \preceq evts_b \\
 & \wedge\ evts_b \preceq evts_a \\
 & \quad \Rightarrow ssub(ssub(evts_a, evts_c), ssub(evts_b, evts_c)) \\
 & \qquad = ssub(evts_a, evts_b)
\end{array}
$$

Proof: The antecedent reduces to:

$$
evts_c \subseteq evts_a \wedge evts_c \subseteq evts_b \wedge evts_b \subseteq evts_a
$$

$$
\begin{array}{ll}
 & ssub(ssub(evts_a, evts_c), ssub(evts_b, evts_c)) \\
= & \quad \text{“ [SA:ssub:def]:p62 ”} \\
 & ssub(evts_a \setminus evts_c, evts_b \setminus evts_c) \\
= & \quad \text{“ [SA:ssub:def]:p62 ”} \\
 & (evts_a \setminus evts_c) \setminus (evts_b \setminus evts_c) \\
= & \quad \text{“ antecedents, and set properties ”} \\
 & evts_a \setminus evts_b \\
= & \quad \text{“ [SA:ssub:def]:p62, backwards ”} \\
 & ssub(evts_a, evts_b)
\end{array}
$$

Law:

$$
\begin{array}{ll}
\text{[SA:sadd:ssub]} & evts \preceq evts' \Rightarrow \\
 & \quad sadd(evts, ssub(evts', evts)) = evts'
\end{array}
$$

First, simplify the antecedent:

$$
\begin{array}{ll}
 & evts \preceq evts' \\
\equiv & \quad \text{“ [SA:pfx:def]:p59 ”} \\
 & evts \subseteq evts' \qquad \text{[SA:sadd:ssub:hyp]}
\end{array}
$$

Now, the consequent:

$$
\begin{array}{ll}
& sadd(evts, ssub(evts', evts)) \\
= & \quad \text{“ [SA:ssub:def]:p62 ”} \\
& sadd(evts, evts' \setminus evts) \\
= & \quad \text{“ [SA:sadd:def]:p61 ”} \\
& evts \cup (evts' \setminus evts) \\
= & \quad \text{“ law of set subtraction, [SA:sadd:ssub:hyp] ”} \\
& evts'
\end{array}
$$

Law:

$$\text{[SA:ssub:sadd]} \qquad ssub(sadd(evts_1, evts_2), evts_1) = evts_2$$

Proof:

$$
\begin{array}{ll}
& ssub(sadd(evts_1, evts_2), evts_1) \\
= & \quad \text{“ [SA:sadd:def]:p61 ”} \\
& ssub(evts_1 \cup evts_2, evts_1) \\
= & \quad \text{“ [SA:ssub:def]:p62 ”} \\
& (evts_1 \cup evts_2) \setminus evts_1 \\
= & \quad \text{“ law of bags ”} \\
& evts_2
\end{array}
$$

6.2.7 Defining $ssync_{\mathcal{SA}}$

$$\text{[SA:SNC:sig]} \qquad ssync_{\mathcal{SA}} : \mathbb{P}\, E \to \mathbb{P}\, E \times \mathbb{P}\, E \to \mathbb{P}(\mathbb{P}\, E)$$

We simply work with evts restriction, removal sum and intersection, and return a singleton set:

$$
\begin{array}{rcl}
\text{[SA:SNC:sym]} & & ssync(cs)(evts_1, evts_2) = ssync(cs)(evts_2, evts_1) \\
\text{[SA:SNC:def]} & & \\
ssync(cs)(evts_1, evts_2) & \widehat{=} & \{((evts_1 \cup evts_2) \setminus cs) \cup ((evts_1 \cap evts_2) \cap cs)\}
\end{array}
$$

The following laws need proving:

$$
\begin{array}{rl}
\text{[SA:SNC:one]} & \forall\, s' \in ssync(cs)(s_1, hnull) \bullet acc(s') \subseteq acc(s_1) \setminus cs \\
\text{[SA:SNC:only]} & s' \in acc(ssync(cs)(s_1, s_2)) \Rightarrow acc(s') \subseteq acc(s_1) \cup acc(s_2) \\
\text{[SA:SNC:sync]} & s' \in acc(ssync(cs)(s_1, s_2)) \Rightarrow cs \cap acc(s') \subseteq cs \cap (acc(s_1) \cap acc(s_2)) \\
\text{[SA:SNC:assoc]} & syncset(cs)(s_1)(ssync(cs)(s_2, s_3)) = syncset(cs)(s_3)(ssync(cs)(s_1, s_2))
\end{array}
$$

The first three can be strengthened to equalities.

6.2.8 Defining $shide_{\mathcal{SA}}$

Finally we need to specify how to hide events in a slot:

$$\text{[SA:SHid:sig]} \qquad shide_{\mathcal{SA}} : \mathbb{P}\,E \rightarrow \mathbb{P}\,E \rightarrow \mathbb{P}\,E$$
$$\text{[SA:SHid:def]} \qquad shide(hdn)evts \mathrel{\widehat=} evts \setminus hid$$

Law:

$$\text{[SA:SHid:evts]} \qquad acc(shide(hdn)evts) = acc(evts) \setminus hdn$$

Proof:

$$\begin{array}{ll} & acc(shide(hdn)evts) \\ = & \quad \text{“ [SA:SHid:def] ”} \\ & acc(evts \setminus hid) \\ = & \quad \text{“ [SA:ACC:def]:p58 ”} \\ & evts \setminus hid \\ = & \quad \text{“ [SA:ACC:def]:p58, backwards ”} \\ & acc(evts) \setminus hdn \end{array}$$

A Slotted-*Circus* Foundation Proofs

A.1 Proofs for Derived Definitions

A.1.1 Proof

Of [ETs:sngl]:p18

$$EQVTRACE(tr, \langle slot \rangle) \equiv EQVTRC(tr, slot)$$

$$
\begin{array}{ll}
 & EQVTRACE(tr, \langle slot \rangle) \\
\equiv & \quad \text{“ list cons notation ”} \\
 & EQVTRACE(tr, slot \mathbin{⨾} \langle\rangle) \\
\equiv & \quad \text{“ [ETs:def:cons]:p18 ”} \\
 & \exists\, tr_0 \bullet tr_0 \leq tr \land EQVTRC(tr_0, slot) \land EQVTRACE(tr - tr_0, \langle\rangle) \\
\equiv & \quad \text{“ [ETs:def:nil]:p18 ”} \\
 & \exists\, tr_0 \bullet tr_0 \leq tr \land EQVTRC(tr_0, slot) \land tr - tr_0 = \langle\rangle \\
\equiv & \quad \text{“ } \sigma - \tau = \langle\rangle \equiv \sigma = \tau \text{ ”} \\
 & \exists\, tr_0 \bullet tr_0 \leq tr \land EQVTRC(tr_0, slot) \land tr = tr_0 \\
\equiv & \quad \text{“ one-point rule on } tr_0 \text{ ”} \\
 & tr \leq tr \land EQVTRC(tr, slot) \\
\equiv & \quad \text{“ simplify ”} \\
 & EQVTRC(tr, slot) \\
\square &
\end{array}
$$

A.1.2 Proof

of [ETs:cat]:p18

$$\begin{array}{l} EQVTRACE(tr_a, slots_a) \land EQVTRACE(tr_b, slots_b) \\ \Rightarrow \\ EQVTRACE(tr_a \frown tr_b, slots_a \frown slots_b) \end{array}$$

Proof: by induction on structure of $slots_a$.

Base case:

$$\begin{array}{l} EQVTRACE(tr_a, \langle\rangle) \land EQVTRACE(tr_b, slots_b) \\ \Rightarrow \\ EQVTRACE(tr_a \frown tr_b, \langle\rangle \frown slots_b) \end{array}$$

$$\begin{array}{ll}
 & EQVTRACE(tr_a, \langle\rangle) \land EQVTRACE(tr_b, slots_b) \Rightarrow EQVTRACE(tr_a \frown tr_b, \langle\rangle \frown slots_b) \\
\equiv & \text{“ [ETs:def:nil]:p18,} \langle\rangle \text{ is unit for } \frown \text{ ”} \\
 & tr_a = \langle\rangle \land EQVTRACE(tr_b, slots_b) \Rightarrow EQVTRACE(tr_a \frown tr_b, slots_b) \\
\equiv & \text{“ Liebniz, } tr_a = \langle\rangle \text{ ”} \\
 & tr_a = \langle\rangle \land EQVTRACE(tr_b, slots_b) \Rightarrow EQVTRACE(\langle\rangle \frown tr_b, slots_b) \\
\equiv & \text{“ } \langle\rangle \text{ is unit for } \frown \text{ ”} \\
 & tr_a = \langle\rangle \land EQVTRACE(tr_b, slots_b) \Rightarrow EQVTRACE(tr_b, slots_b) \\
\equiv & \text{“ } A \land B \Rightarrow B \text{ ”} \\
 & \textsc{True}
\end{array}$$

Inductive step, assume:

$$\begin{array}{l} EQVTRACE(tr_a, slots_a) \land EQVTRACE(tr_b, slots_b) \\ \Rightarrow \\ EQVTRACE(tr_a \frown tr_b, slots_a \frown slots_b) \end{array} \qquad \text{[prf:ETs:cat:hyp]}$$

Show:

$$\begin{array}{l} EQVTRACE(tr_c, slot \mathbin{⨾} slots_a) \land EQVTRACE(tr_b, slots_b) \\ \Rightarrow \\ EQVTRACE(tr_c \frown tr_b, slot \mathbin{⨾} slots_a \frown slots_b) \end{array}$$

$EQVTRACE(tr_c, slot \fatsemi slots_a) \land EQVTRACE(tr_b, slots_b)$
$\Rightarrow EQVTRACE(tr_c \frown tr_b, slot \fatsemi slots_a \frown slots_b)$

$\equiv$ " [ETs:def:cons]:p18 "

$(\exists\, tr_0 \bullet tr_0 \leq tr_c \land EQVTRC(tr_0, slot) \land EQVTRACE(tr_c - tr_0, slots_a))$
$\land EQVTRACE(tr_b, slots_b)$
$\Rightarrow EQVTRACE(tr_c \frown tr_b, slot \fatsemi slots_a \frown slots_b)$

$\equiv$ " tr_0 not free in $EQVTRACE(tr_b, slots_b)$ "

$(\exists\, tr_0 \bullet$
$tr_0 \leq tr_c \land EQVTRC(tr_0, slot)$
$\land EQVTRACE(tr_c - tr_0, slots_a) \land EQVTRACE(tr_b, slots_b))$
$\Rightarrow EQVTRACE(tr_c \frown tr_b, slot \fatsemi slots_a \frown slots_b)$

$\equiv$ " $(\exists\, x \bullet P(x)) \Rightarrow Q$ equivales $(\forall\, x \bullet P(x) \Rightarrow Q)$ "

$\forall\, tr_0 \bullet$
$tr_0 \leq tr_c \land EQVTRC(tr_0, slot)$
$\land EQVTRACE(tr_c - tr_0, slots_a) \land EQVTRACE(tr_b, slots_b)$
$\Rightarrow EQVTRACE(tr_c \frown tr_b, slot \fatsemi slots_a \frown slots_b)$

$\equiv$ " assume univ. quant. over free vars "

$tr_0 \leq tr_c \land EQVTRC(tr_0, slot)$
$\land EQVTRACE(tr_c - tr_0, slots_a) \land EQVTRACE(tr_b, slots_b)$
$\Rightarrow EQVTRACE(tr_c \frown tr_b, slot \fatsemi slots_a \frown slots_b)$

$\equiv$ " introduce $tr_a = tr_c - tr_0$, and hence $tr_c = tr_0 \frown tr_a$ "

$tr_0 \leq tr_0 \frown tr_a \land EQVTRC(tr_0, slot)$
$\land EQVTRACE(tr_a, slots_a) \land EQVTRACE(tr_b, slots_b)$
$\Rightarrow EQVTRACE(tr_0 \frown tr_a \frown tr_b, slot \fatsemi slots_a \frown slots_b)$

$\equiv$ " [ETs:def:cons]:p18 "

$tr_0 \leq tr_0 \frown tr_a \land EQVTRC(tr_0, slot)$
$\land EQVTRACE(tr_a, slots_a) \land EQVTRACE(tr_b, slots_b)$
$\Rightarrow$
$\exists\, tr_1 \bullet tr_1 \leq tr_0 \frown tr_a \frown tr_b$
$\land EQVTRC(tr_1, slot)$
$\land EQVTRACE(tr_0 \frown tr_a \frown tr_b - tr_1, slots_a \frown slots_b)$

$\equiv$ " $P \Rightarrow (\exists\, x \bullet Q(x))$ equivales $(\exists\, x \bullet P \Rightarrow Q(x))$ "

$\exists\, tr_1 \bullet$
$tr_0 \leq tr_0 \frown tr_a \land EQVTRC(tr_0, slot)$
$\land EQVTRACE(tr_a, slots_a) \land EQVTRACE(tr_b, slots_b)$
$\Rightarrow$
$tr_1 \leq tr_0 \frown tr_a \frown tr_b$
$\land EQVTRC(tr_1, slot)$
$\land EQVTRACE(tr_0 \frown tr_a \frown tr_b - tr_1, slots_a \frown slots_b)$

At this point we introduce:

$$
\begin{aligned}
P(tr_1, tr_0) \;\widehat{=}\;\; & tr_0 \leq tr_0 \frown tr_a \land EQVTRC(tr_0, slot) \\
& \land EQVTRACE(tr_a, slots_a) \land EQVTRACE(tr_b, slots_b) \\
& \quad \Rightarrow \\
& \quad\; tr_1 \leq tr_0 \frown tr_a \frown tr_b \\
& \quad\; \land EQVTRC(tr_1, slot) \\
& \quad\; \land EQVTRACE(tr_0 \frown tr_a \frown tr_b - tr_1, slots_a \frown slots_b)
\end{aligned}
$$

We now continue our proof as:

$$
\begin{aligned}
& \exists\, tr_1 \bullet P(tr_1, tr_0) \qquad \text{by definition just given} \\
\equiv\; & \quad \text{“ case split on } tr_1 = tr_0 \text{ ”} \\
& \exists\, tr_1 \bullet tr_1 = tr_0 \land P(tr_1, tr_0) \lor tr_1 \neq tr_0 \land P(tr_1, tr_0) \\
\equiv\; & \quad \text{“ } \lor - \exists \text{ distr. ”} \\
& (\exists\, tr_1 \bullet tr_1 = tr_0 \land P(tr_1, tr_0)) \lor (\exists\, tr_1 \bullet tr_1 \neq tr_0 \land P(tr_1, tr_0)) \\
\equiv\; & \quad \text{“ one-point rule ”} \\
& P(tr_0, tr_0) \lor (\exists\, tr_1 \bullet tr_1 \neq tr_0 \land P(tr_1, tr_0)) \\
\equiv\; & \quad \text{“ [prf:ETs:cat:lemma]:p71 ”} \\
& \textsc{True} \lor (\exists\, tr_1 \bullet tr_1 \neq tr_0 \land P(tr_1, tr_0)) \\
\equiv\; & \quad \text{“ simplify ”} \\
& \textsc{True}
\end{aligned}
$$

□

We need to show the following lemma:

$$P(tr_0, tr_0) \equiv \text{TRUE} \qquad \text{[prf:ETs:cat:lemma]}$$

Proof:

$P(tr_0, tr_0)$

$\equiv$ " defn. of P above "

$tr_0 \le tr_0 \frown tr_a \land EQVTRC(tr_0, slot)$

$\land\ EQVTRACE(tr_a, slots_a) \land EQVTRACE(tr_b, slots_b)$

$\Rightarrow$

$tr_0 \le tr_0 \frown tr_a \frown tr_b$

$\land\ EQVTRC(tr_0, slot)$

$\land\ EQVTRACE(tr_0 \frown tr_a \frown tr_b - tr_0, slots_a \frown slots_b)$

$\equiv$ " $\sigma \le \sigma \frown \ldots, \alpha \frown \beta - \alpha = \beta$ "

$EQVTRC(tr_0, slot) \land EQVTRACE(tr_a, slots_a) \land EQVTRACE(tr_b, slots_b)$

$\Rightarrow$

$EQVTRC(tr_0, slot) \land EQVTRACE(tr_a \frown tr_b, slots_a \frown slots_b)$

$\equiv$ " $A \land B \Rightarrow A \land C$ equivales $A \land B \Rightarrow C$ "

$EQVTRC(tr_0, slot) \land EQVTRACE(tr_a, slots_a) \land EQVTRACE(tr_b, slots_b)$

$\Rightarrow EQVTRACE(tr_a \frown tr_b, slots_a \frown slots_b)$

$\equiv$ " $A \land B \Rightarrow C$ equivales $A \Rightarrow (BimpliesC$ "

$EQVTRC(tr_0, slot) \Rightarrow$

$(\ EQVTRACE(tr_a, slots_a) \land EQVTRACE(tr_b, slots_b)$

$\Rightarrow EQVTRACE(tr_a \frown tr_b, slots_a \frown slots_b)\ )$

$\equiv$ " [prf:ETs:cat:hp]:p**??** "

$EQVTRC(tr_0, slot) \Rightarrow \text{TRUE}$

$\equiv$ " simplify "

TRUE

□

A.1.3 Proof

of [ETs:elems]:p18

$$EqvTraces(tr, slots) \Rightarrow elems(tr) = \bigcup_{i \in 1}^{\#slots} \{acc(slots(i))\}$$

Proof by structural induction on slots.
Base case $slots = \langle\rangle$:

$$
\begin{aligned}
& EqvTraces(tr, \langle\rangle) \Rightarrow elems(tr) = \bigcup_{i \in 1}^{\#\langle\rangle} \{acc(\langle\rangle(i))\} \\
\equiv \quad & \text{“ [ETs:def:nil]:p18, } \#\langle\rangle = 0 \text{ ”} \\
& tr = \langle\rangle \Rightarrow elems(tr) = \bigcup_{i \in 1}^{0} \{acc(\langle\rangle(i))\} \\
\equiv \quad & \text{“ Liebniz, } tr - \langle\rangle \text{, null index range ”} \\
& elems(\langle\rangle) = \bigcup_{i \in 1}^{0} \{\} \\
\equiv \quad & \text{“ defn. } elems, \bigcup \text{ ”} \\
& \{\} = \{\} \\
\equiv \quad & \text{“ reflexivity of = ”} \\
& \text{TRUE}
\end{aligned}
$$

Inductive step, assuming

$$EqvTraces(tr, slots) \Rightarrow elems(tr) = \bigcup_{i \in 1}^{\#slots} \{acc(slots(i))\} \quad \text{[prf:ETs:elems:hyp]}$$

to show

$$EqvTraces(tr', slot \mathbin{⨾} slots) \Rightarrow elems(tr') = \bigcup_{i \in 1}^{\#(slot ⨾ slots)} \{acc((slot \mathbin{⨾} slots)(i))\}$$

$EqvTraces(tr', slot \fatsemi slots)$

$$\Rightarrow elems(tr') = \bigcup_{i \in 1}^{\#(slot \fatsemi slots)} \{acc((slot \fatsemi slots)(i))\}$$

$\equiv$ " defns. of # and list indexing "

$EqvTraces(tr', slot \fatsemi slots)$

$$\Rightarrow elems(tr') = \bigcup_{i \in 1}^{\#(slots)+1} \{acc(slot \lhd i = 1 \rhd slots(i-1))\}$$

$\equiv$ " range split, re-indexing, eval. conditional "

$EqvTraces(tr', slot \fatsemi slots)$

$$\Rightarrow elems(tr') = acc(slot) \cup \bigcup_{i \in 1}^{\#slots} \{acc(slots(i))\}$$

$\equiv$ " [ETs:def:cons]:p18 "

$(\exists tr_0 \bullet$

$tr_0 \leq tr'$

$\land EQVTRC(tr_0, slot) \land EqvTraces(tr' - tr_0, slots)\ )$

$$\Rightarrow elems(tr') = acc(slot) \cup \bigcup_{i \in 1}^{\#slots} \{acc(slots(i))\}$$

$\equiv$ " $(\exists x \bullet P) \Rightarrow Q \equiv \forall x \bullet P \Rightarrow Q$, drop $\forall$ "

$tr_0 \leq tr'$

$\land EQVTRC(tr_0, slot) \land EqvTraces(tr' - tr_0, slots)$

$$\Rightarrow elems(tr') = acc(slot) \cup \bigcup_{i \in 1}^{\#slots} \{acc(slots(i))\}$$

$\equiv$ " introduce $tr = tr' - tr_0$ "

$tr_0 \leq tr'$

$\land EQVTRC(tr_0, slot) \land EqvTraces(tr, slots)$

$$\Rightarrow elems(tr') = acc(slot) \cup \bigcup_{i \in 1}^{\#slots} \{acc(slots(i))\}$$

$\equiv$ " $tr' = tr_0 \frown tr$ "

$tr_0 \leq tr_0 \frown tr$

$\land EQVTRC(tr_0, slot) \land EqvTraces(tr, slots)$

$$\Rightarrow elems(tr_0 \frown tr) = acc(slot) \cup \bigcup_{i \in 1}^{\#slots} \{acc(slots(i))\}$$

$\equiv$ " $\sigma \leq \sigma \frown \tau$, defn. $elems$ "

$EQVTRC(tr_0, slot) \land EqvTraces(tr, slots)$

$\Rightarrow elems(tr_0) \cup elems(tr) = acc(slot) \cup \bigcup_{i \in 1}^{\#slots} \{acc(slots(i))\}$

$\equiv$ “ [ET:elems]:p17 ”

$EQVTRC(tr_0, slot) \land elems(tr_0) = acc(slot) \land EqvTraces(tr, slots)$

$\Rightarrow elems(tr_0) \cup elems(tr) = acc(slot) \cup \bigcup_{i \in 1}^{\#slots} \{acc(slots(i))\}$

$\equiv$ “ Liebniz, $elems(tr_0) = acc(slot)$ ”

$EQVTRC(tr_0, slot) \land EqvTraces(tr, slots)$

$\Rightarrow acc(slot) \cup elems(tr) = acc(slot) \cup \bigcup_{i \in 1}^{\#slots} \{acc(slots(i))\}$

$\equiv$ “ [prf:ETs:elems:hyp]:p72 ”

$EQVTRC(tr_0, slot) \land EqvTraces(tr, slots)$

$\land\ elems(tr) = \bigcup_{i \in 1}^{\#slots} \{acc(slots(i))\}$

$\Rightarrow acc(slot) \cup elems(tr) = acc(slot) \cup \bigcup_{i \in 1}^{\#slots} \{acc(slots(i))\}$

$\equiv$ “ Liebniz, $elems(tr) = \ldots$ ”

$EQVTRC(tr_0, slot) \land EqvTraces(tr, slots)$

$\land\ elems(tr) = \bigcup_{i \in 1}^{\#slots} \{acc(slots(i))\}$

$\Rightarrow acc(slot) \cup \bigcup_{i \in 1}^{\#slots} \{acc(slots(i))\} = acc(slot) \cup \bigcup_{i \in 1}^{\#slots} \{acc(slots(i))\}$

$\equiv$ “ reflexivity of = ”

$EQVTRC(tr_0, slot) \land EqvTraces(tr, slots)$

$\land\ elems(tr) = \bigcup_{i \in 1}^{\#slots} \{acc(slots(i))\}$

$\Rightarrow$ TRUE

$\equiv$ “ simplify ”

TRUE

□

A.1.4 Proof

of [ETs:null]:p18

$$EqvTraces(\langle\rangle, slots) \equiv \forall\, i \in 1 \ldots \#slots \bullet EQVTRC(\langle\rangle, slots(i))$$

Proof by structural induction on *slots*.
Base case $slots = \langle\rangle$

$$\begin{aligned}
& EqvTraces(\langle\rangle, \langle\rangle) \equiv \forall\, i \in 1 \ldots \#\langle\rangle \bullet EQVTRC(\langle\rangle, \langle\rangle(i)) \\
\equiv \quad & \text{“ [ETs:def:nil]:p18, and } \{1 \ldots \#\langle\rangle\} = \emptyset \text{ ”} \\
& \langle\rangle = \langle\rangle \equiv \forall\, i \in \emptyset \bullet EQVTRC(\langle\rangle, \langle\rangle(i)) \\
\equiv \quad & \text{“ refl. of } =, \forall\, x : \emptyset \bullet P = \textsc{True} \text{ ”} \\
& \textsc{True} \equiv \textsc{True} \\
\equiv \quad & \text{“ simplify ”} \\
& \textsc{True}
\end{aligned}$$

Inductive case, assuming

$$EqvTraces(\langle\rangle, slots) \equiv \forall\, i \in 1 \ldots \#slots \bullet EQVTRC(\langle\rangle, slots(i)) \quad \text{[prf:ETs:null:hyp]}$$

to show

$$EqvTraces(\langle\rangle, slot \mathbin{⨾} slots) \equiv \forall\, i \in 1 \ldots \#(slot \mathbin{⨾} slots) \bullet EQVTRC(\langle\rangle, (slot \mathbin{⨾} slots)(i))$$

We start with the lhs, to simplify:

$$\begin{aligned}
& EqvTraces(\langle\rangle, slot \mathbin{⨾} slots) \\
\equiv \quad & \text{“ [ETs:def:cons]:p18 ”} \\
& \exists\, tr_0 \bullet tr_0 \leq \langle\rangle \land EQVTRC(tr_0, slot) \land EQVTRACE(\langle\rangle - tr_0, slots) \\
\equiv \quad & \text{“ } \sigma \leq \langle\rangle \Rightarrow \sigma = \langle\rangle \text{ ”} \\
& \exists\, tr_0 \bullet tr_0 = \langle\rangle \land EQVTRC(tr_0, slot) \land EQVTRACE(\langle\rangle - tr_0, slots) \\
\equiv \quad & \text{“ one-point rule, } tr_0 = \langle\rangle,\ \langle\rangle - \langle\rangle = \langle\rangle \text{ ”} \\
& EQVTRC(\langle\rangle, slot) \land EQVTRACE(\langle\rangle, slots) \\
\equiv \quad & \text{“ [prf:ETs:null:hyp]:p75 ”} \\
& EQVTRC(\langle\rangle, slot) \land \forall\, i \in 1 \ldots \#slots \bullet EQVTRC(\langle\rangle, slots(i))
\end{aligned}$$

We now look at the rhs:

$$
\begin{array}{ll}
& \forall\, i \in 1 \ldots \#(slot \mathbin{⨾} slots) \bullet EQVTRC(\langle\rangle, (slot \mathbin{⨾} slots)(i)) \\
\equiv & \quad \text{“ defn. \#,sequence indexing ”} \\
& \forall\, i \in 1 \ldots \#slots + 1 \bullet EQVTRC(\langle\rangle, (slot \lhd i = 1 \rhd slots(i-1))) \\
\equiv & \quad \text{“ range split } 1 \mid 2 \ldots \#slots + 1 \text{ ”} \\
& \forall\, i \in \{1\} \bullet EQVTRC(\langle\rangle, (slot \lhd i = 1 \rhd slots(i-1))) \\
& \wedge \\
& \forall\, i \in 2 \ldots \#slots + 1 \bullet EQVTRC(\langle\rangle, (slot \lhd i = 1 \rhd slots(i-1))) \\
\equiv & \quad \text{“ simplify for } i = 1\text{, and let } j = i - 1 \text{ ”} \\
& EQVTRC(\langle\rangle, slot) \wedge \\
& \forall\, j - 1 \in 2 \ldots \#slots + 1 \bullet EQVTRC(\langle\rangle, (slot \lhd j + 1 = 1 \rhd slots((j + 1 - 1))) \\
\equiv & \quad \text{“ simplify, noting } j + 1 \neq 1 \text{ ”} \\
& EQVTRC(\langle\rangle, slot) \wedge \forall\, j \in 1 \ldots \#slots \bullet EQVTRC(\langle\rangle, slots(j))
\end{array}
$$

The lhs and rhs are the same, modulo α-substitution.

□

A.1.5 Proof

of [EX:subseq]:p19

$$slots_a \leq slots_b \Rightarrow slots_a \preccurlyeq slots_b$$

Proof by induction on rules for $\leq$, noting that *slots* are non-empty

Base case $slots_a = \langle slot \rangle$

$$\begin{aligned}
& \langle slot \rangle \leq slot \mathbin{⨾} slots_b' \Rightarrow \langle slot \rangle \preccurlyeq (slot \mathbin{⨾} slots_b) \\
\equiv \quad & \text{“ defn. } \leq, \textsc{True} \Rightarrow A = A \text{ ”} \\
& \langle slot \rangle \preccurlyeq (slot \mathbin{⨾} slots_b) \\
\equiv \quad & \text{“ [EX:def]:p19 ”} \\
& front(\langle slot \rangle) < (slot \mathbin{⨾} slots_b) \wedge last(\langle slot \rangle) \preceq (slot \mathbin{⨾} slots_b)(\#\langle slot \rangle) \\
\equiv \quad & \text{“ defns: } front,\ last\ \# \text{ ”} \\
& \langle\rangle < (slot \mathbin{⨾} slots_b) \wedge slot \preceq (slot \mathbin{⨾} slots_b)(1) \\
\equiv \quad & \text{“ } \langle\rangle < x \mathbin{⨾} \ldots \text{, defn. indexing ”} \\
& \textsc{True} \wedge slot \preceq slot \\
\equiv \quad & \text{“ [pfx:refl]:p17 ”} \\
& \textsc{True} \wedge \textsc{True} \\
\equiv \quad & \text{“ simplify ”} \\
& \textsc{True}
\end{aligned}$$

Inductive step, assuming [prf:EX:subseq:hyp], $slots_a \leq slots_b \Rightarrow slots_a \preccurlyeq slots_b$ to show

$$slot \mathbin{⨾} slots_a \leq slot \mathbin{⨾} slots_b \Rightarrow slot \mathbin{⨾} slots_a \preccurlyeq slot \mathbin{⨾} slots_b$$

$$\begin{aligned}
& slot \mathbin{⨾} slots_a \leq slot \mathbin{⨾} slots_b \Rightarrow slot \mathbin{⨾} slots_a \preccurlyeq slot \mathbin{⨾} slots_b \\
\equiv \quad & \text{“ defn } \leq, \preccurlyeq \text{ ”} \\
& slots_a \leq slots_b \Rightarrow \\
& front(slot \mathbin{⨾} slots_a) < slot \mathbin{⨾} slots_b \\
& \quad \wedge last(slot \mathbin{⨾} slots_a) \preceq (slot \mathbin{⨾} slots_b)(\#(slot \mathbin{⨾} slots_a)) \\
\equiv \quad & \text{“ defn } front,\ last,\ \# \text{ ”} \\
& slots_a \leq slots_b \Rightarrow \\
& slot \mathbin{⨾} front(slots_a) < slot \mathbin{⨾} slots_b \\
& \quad \wedge last(slots_a) \preceq (slot \mathbin{⨾} slots_b)(\#(slots_a) + 1) \\
\equiv \quad & \text{“ defn. } <\text{, indexing ”} \\
& slots_a \leq slots_b \Rightarrow \\
& front(slots_a) < slots_b \wedge last(slots_a) \preceq slots_b(\#slots_a) \\
\equiv \quad & \text{“ [EX:def]:p19,backwards ”} \\
& slots_a < slots_b \Rightarrow slots_a \preccurlyeq slots_b \\
\equiv \quad & \text{“ [prf:EX:subseq:hyp]:p77 ”} \\
& \textsc{True}
\end{aligned}$$

□

A.1.6 Proof

of [EX:refl]:p19

$$slots \preccurlyeq slots$$

$$
\begin{aligned}
& slots \preccurlyeq slots \\
\equiv \quad & \text{“ [EX:def]:p19 ”} \\
& front(slots) < slots \land last(slots) \preceq slots(\#slots) \\
\equiv \quad & \text{“ property of } front \text{ ”} \\
& \textsc{True} \land last(slots) \preceq slots(\#slots) \\
\equiv \quad & \text{“ } last(\sigma) = \sigma(\#\sigma) \text{ ”} \\
& \textsc{True} \land last(slots) \preceq last(slots) \\
\equiv \quad & \text{“ [pfx:refl]:p17 ”} \\
& \textsc{True} \land \textsc{True} \\
\equiv \quad & \text{“ simplify ”} \\
& \textsc{True}
\end{aligned}
$$

□

A.1.7 Proof

of [EX:trans]:p19

$$slots_a \preccurlyeq slots_b \land slots_b \preccurlyeq slots_c \Rightarrow slots_a \preccurlyeq slots_c$$

First expand the antecedent out:

$$\begin{array}{ll} & slots_a \preccurlyeq slots_b \land slots_b \preccurlyeq slots_c \\ \equiv & \quad \text{“ [EX:def]:p19 ”} \\ & front(slots_a) < slots_b \land last(slots_a) \preceq slots_b(\#slots_a) \\ & \land \\ & front(slots_b) < slots_c \land last(slots_b) \preceq slots_c(\#slots_b) \\ \equiv & \quad \text{“ reorganise ”} \\ & front(slots_a) < slots_b \land front(slots_b) < slots_c \\ & \land \\ & last(slots_a) \preceq slots_b(\#slots_a) \land last(slots_b) \preceq slots_c(\#slots_b) \\ \equiv & \quad \text{“ [Seq:FrontLT:trans]:p215 ”} \\ & front(slots_a) < slots_b \land front(slots_b) < slots_c \land front(slots_a) < slots_c \\ & \land \\ & last(slots_a) \preceq slots_b(\#slots_a) \land last(slots_b) \preceq slots_c(\#slots_b) \end{array}$$

At this point we do a case-split on $\#slots_a = \#slots_b$.
Case 1 $\#slots_a = \#slots_b$.

$$\begin{array}{ll} & front(slots_a) < slots_b \land front(slots_b) < slots_c \land front(slots_a) < slots_c \\ & \land \\ & last(slots_a) \preceq slots_b(\#slots_a) \land last(slots_b) \preceq slots_c(\#slots_b) \\ \equiv & \quad \text{“ Liebniz } \#slots_a = \#slots_b \text{ ”} \\ & front(slots_a) < slots_b \land front(slots_b) < slots_c \land front(slots_a) < slots_c \\ & \land \\ & last(slots_a) \preceq slots_b(\#slots_b) \land last(slots_b) \preceq slots_c(\#slots_a) \\ \equiv & \quad \text{“ [Seq:Last:index]:p\textbf{??} ”} \\ & front(slots_a) < slots_b \land front(slots_b) < slots_c \land front(slots_a) < slots_c \\ & \land \\ & last(slots_a) \preceq last(slots_b) \land last(slots_b) \preceq slots_c(\#slots_a) \\ \Rightarrow & \quad \text{“ [pfx:trans]:p17, weaken ”} \\ & front(slots_a) < slots_c \\ & \land \\ & last(slots_a) \preceq slots_c(\#slots_a) \\ \equiv & \quad \text{“ [EX:def]:p19, backwards ”} \\ & slots_a \preccurlyeq slots_c \end{array}$$

Case 2 $\#slots_a \neq \#slots_b$.

First, we note that $front(slots_a) < slots_b \Rightarrow \#slots_a \leq \#slots_b$ (by [Seq:FrontLT:len]:p214), so we can strengthen the case to $\#slots_a < \#slots_b$.

$$
\begin{array}{ll}
& front(slots_a) < slots_b \land front(slots_b) < slots_c \land front(slots_a) < slots_c \\
& \land \\
& last(slots_a) \preceq slots_b(\#slots_a) \land last(slots_b) \preceq slots_c(\#slots_b) \\
\equiv & \quad \text{“ [prf:EX:trans:lemma1]:p81 ”} \\
& front(slots_a) < slots_b \land front(slots_b) < slots_c \land front(slots_a) < slots_c \\
& \land \\
& last(slots_a) \preceq slots_b(\#slots_a) \land last(slots_b) \preceq slots_c(\#slots_b) \\
& \land\ slots_b(\#slots_a) = slots_c(\#slots_a) \\
\equiv & \quad \text{“ Liebniz } slots_b(\#slots_a) = slots_c(\#slots_a) \text{ ”} \\
& front(slots_a) < slots_b \land front(slots_b) < slots_c \land front(slots_a) < slots_c \\
& \land \\
& last(slots_a) \preceq slots_c(\#slots_a) \land last(slots_b) \preceq slots_c(\#slots_b) \\
& \land\ slots_b(\#slots_a) = slots_c(\#slots_a) \\
\Rightarrow & \quad \text{“ weaken ”} \\
& front(slots_a) < slots_c \land last(slots_a) \preceq slots_c(\#slots_a) \\
\equiv & \quad \text{“ [EX:def]:p19, backwards ”} \\
& slots_a \preccurlyeq slots_c
\end{array}
$$

□

Lemma [prf:EX:trans:lemma1]

$$\#a < \#b \wedge \mathit{front}(a) \leq b \wedge \mathit{front}(b) \leq c \Rightarrow b(\#a) = c(\#a)$$

Start with antecedent:

$$
\begin{array}{ll}
 & \#a < \#b \wedge \mathit{front}(a) \leq b \wedge \mathit{front}(b) \leq c \\
\equiv & \quad \text{“ [Seq:LE:prefix]:p214 ”} \\
 & \#a < \#b \wedge \mathit{front}(a) \leq b \wedge \mathit{front}(b) \leq c \\
 & \wedge \forall\, i : 1 \ldots \#(\mathit{front}(b)) \bullet (\mathit{front}(b))(i) = c(i) \\
\equiv & \quad \text{“ [Seq:Front:index]:p214 ”} \\
 & \#a < \#b \wedge \mathit{front}(a) \leq b \wedge \mathit{front}(b) \leq c \\
 & \wedge \forall\, i : 1 \ldots \#(\mathit{front}(b)) \bullet (\mathit{front}(b))(i) = c(i) \\
 & \wedge \forall\, j : 1 \ldots \#(\mathit{front}(b)) \bullet (\mathit{front}(b))(j) = b(j) \\
\equiv & \quad \text{“ [Seq:Front:len]:p214 ”} \\
 & \#a < \#b \wedge \mathit{front}(a) \leq b \wedge \mathit{front}(b) \leq c \\
 & \wedge \forall\, i : 1 \ldots (\#b) - 1 \bullet (\mathit{front}(b))(i) = c(i) \\
 & \wedge \forall\, j : 1 \ldots (\#b) - 1 \bullet (\mathit{front}(b))(j) = b(j) \\
\equiv & \quad \text{“ } \#b \leq \#c \text{, by [Seq:FrontLT:len]:p214 ”} \\
 & \#a < \#b \wedge \#b \leq \#c \wedge \mathit{front}(a) \leq b \wedge \mathit{front}(b) \leq c \\
 & \wedge \forall\, i : 1 \ldots (\#b) - 1 \bullet (\mathit{front}(b))(i) = c(i) \\
 & \wedge \forall\, j : 1 \ldots (\#b) - 1 \bullet (\mathit{front}(b))(j) = b(j) \\
\Rightarrow & \quad \text{“ } \#a \in 1 \ldots (\#b) - 1 \text{, instantiate ”} \\
 & \#a < \#b \wedge \#b \leq \#c \wedge \mathit{front}(a) \leq b \wedge \mathit{front}(b) \leq c \\
 & \wedge (\mathit{front}(b))(\#a) = c(\#a) \wedge (\mathit{front}(b))(\#a) = b(\#a) \\
\Rightarrow & \quad \text{“ trans. of =, weaken ”} \\
 & c(\#a) = b(\#a)
\end{array}
$$

□

A.1.8 Proof

of [EX:anti]:p19

$$(\forall\, slot_a, slot_b \bullet slot_a \preceq slot_b \land slot_b \preceq slot_a \Rightarrow slot_a = slot_b)$$
$$\Rightarrow$$
$$(slots_a \preccurlyeq slots_b \land slots_b \preccurlyeq slots_a \Rightarrow slots_a = slots_b)$$

We assume

[EX:anti:hyp1]	$slot_a \preceq slot_b \land slot_b \preceq slot_a \Rightarrow slot_a = slot_b$
[EX:anti:hyp2]	$slots_a \preccurlyeq slots_b$
[EX:anti:hyp3]	$slots_b \preccurlyeq slots_a$

in order to show

$$slots_a = slots_b$$

We first expand the definition of $\preccurlyeq$ in the 2nd and 3rd hypotheses:

[EX:anti:hyp2']	$front(slots_a) < slots_b \land last(slots_a) \preceq slots_b(\#slots_a)$
[EX:anti:hyp3']	$front(slots_b) < slots_a \land last(slots_b) \preceq slots_a(\#slots_b)$

We start with this:

$$front(slots_a) < slots_b \land last(slots_a) \preceq slots_b(\#slots_a)$$
$$\quad \land front(slots_b) < slots_a \land last(slots_b) \preceq slots_a(\#slots_b)$$
$$\Rightarrow \quad \text{“ [Seq:FrontLT:anti]:p215 ”}$$
$$front(slots_a) = front(slots_b)$$
$$\quad \land last(slots_a) \preceq slots_b(\#slots_a) \land last(slots_b) \preceq slots_a(\#slots_b)$$
$$\equiv \quad \text{“ [Seq:Front:len]:p214 ”}$$
$$front(slots_a) = front(slots_b)$$
$$\quad \land last(slots_a) \preceq slots_b(\#slots_b) \land last(slots_b) \preceq slots_a(\#slots_a)$$
$$\equiv \quad \text{“ [Seq:Last:index]:p}\textbf{??}\text{, } slots_a,\ slots_b \neq \langle\rangle \text{ ”}$$
$$front(slots_a) = front(slots_b)$$
$$\quad \land last(slots_a) \preceq last(slots_b) \land last(slots_b) \preceq last(slots_a)$$
$$\Rightarrow \quad \text{“ [EX:anti:hyp1]:p82 ”}$$
$$front(slots_a) = front(slots_b) \land last(slots_a) = last(slots_b)$$
$$\equiv \quad \text{“ [Seq:Front-Last:eq]:p214 ”}$$
$$slots_a = slots_b$$

□

A.1.9 Proof

of [EX:null]:p19

$$\langle snull(r) \rangle \preccurlyeq slots$$

$$\begin{array}{ll} & \langle snull(r) \rangle \preccurlyeq slots \\ \equiv & \quad \text{“ [EX:def]:p19 ”} \\ & front\langle snull(r) \rangle < slots \land last\langle snull(r) \rangle \preceq slots(\#\langle snull(r) \rangle) \\ \equiv & \quad \text{“ defn. } front,\ last,\ \# \text{ ”} \\ & \langle\rangle < slots \land snull(r) \preceq slots(1) \\ \equiv & \quad \text{“ [Seq:SPfx:def:sngl]:p?? noting } slots \neq \langle\rangle, \text{[SN:pfx]:p17 ”} \\ & \text{True} \land \text{True} \\ \equiv & \quad \text{“ simplify ”} \\ & \text{True} \end{array}$$

□

A.1.10 Proof

of [EX:dif]:p22

$$slots \preccurlyeq slots' \equiv \langle snull(r) \rangle \preccurlyeq dif(slots', slots)$$

The left-to-right implication is true, via [EX:null]:p19, as it asserts that [DF:pre]:p21 is satisfied in the antecedent.

So we focus on the right-to-left implication:

$$\langle snull(r) \rangle \preccurlyeq dif(slots', slots) \Rightarrow slots \preccurlyeq slots'$$

The antecedent is only well-defined if the consequent holds, so we can proceed as follows:

$$\begin{array}{ll} & \langle snull(r) \rangle \preccurlyeq dif(slots', slots) \\ \equiv & \quad \text{“ defined if [DF:pre]:p21 holds ”} \\ & \langle snull(r) \rangle \preccurlyeq dif(slots', slots) \land slots \preccurlyeq slots' \\ \Rightarrow & \quad \text{“ } A \land B \Rightarrow B \text{ ”} \\ & slots \preccurlyeq slots' \end{array}$$

□

A.1.11 Proof

of [DF:ER:first]:p22

$$eqvref(sl_1 \setminus\!\!\setminus sl_2) = eqvref(sl_1)$$

$$
\begin{array}{ll}
 & eqvref(sl_1 \setminus\!\!\setminus sl_2) \\
= & \quad \text{“ [ER:def]:p18 ”} \\
 & sref(last(sl_1 \setminus\!\!\setminus sl_2)) \\
= & \quad \text{“ [DF:def]:p21 ”} \\
 & sref(last((s_1 \setminus s_2) \fatsemi sfx)) \\
 & \textbf{where } s_2 = last(sl_2) \\
 & \qquad (s_1 \fatsemi sfx) = sl_1 - front(sl_2)
\end{array}
$$

We do a case split on $sfx = \langle\rangle$

Case Split $sfx = \langle\rangle$

$$
\begin{array}{ll}
 & sref(last((s_1 \setminus s_2) \fatsemi sfx)) \\
 & \textbf{where } s_2 = last(sl_2) \\
 & \qquad (s_1 \fatsemi sfx) = sl_1 - front(sl_2) \\
= & \quad \text{“ } sfx = \langle\rangle \text{ ”} \\
 & sref(last(\langle (s_1 \setminus s_2) \rangle)) \\
 & \textbf{where } s_2 = last(sl_2) \\
 & \qquad (s_1 \fatsemi \langle\rangle) = sl_1 - front(sl_2) \\
= & \quad \text{“ } last(\langle x \rangle) = x \text{ ”} \\
 & sref(s_1 \setminus s_2) \\
 & \textbf{where } s_2 = last(sl_2) \\
 & \qquad (s_1 \fatsemi \langle\rangle) = sl_1 - front(sl_2) \\
= & \quad \text{“ [SSub:ref]:p14, discard } s_2 \text{ ”} \\
 & sref(s_1) \\
 & \textbf{where } (s_1 \fatsemi \langle\rangle) = sl_1 - front(sl_2) \\
= & \quad \text{“ } s_1 = head(sl_1 - front(sl_2)) = last(sl_1 - front(sl_2)) \text{, as tail is nil ”} \\
 & sref(last(sl_1 - front(sl_2)))
\end{array}
$$

Case Split $sfx \neq \langle\rangle$

$$
\begin{array}{ll}
 & sref(last((s_1 \setminus s_2) ⨾ sfx)) \\
 & \textbf{where}\ s_2 = last(sl_2) \\
 & \qquad (s_1 ⨾ sfx) = sl_1 - front(sl_2) \\
= & \text{“ } last(x ⨾ \sigma) = last(\sigma), \text{ when } \sigma \neq \langle\rangle \text{ ”} \\
 & sref(last(sfx)) \\
 & \textbf{where}\ s_2 = last(sl_2) \\
 & \qquad (s_1 ⨾ sfx) = sl_1 - front(sl_2) \\
= & \text{“ } last(\sigma) = last(x ⨾ \sigma), \text{if } \sigma \neq \langle\rangle \text{ ”} \\
 & sref(last(s_1 ⨾ sfx)) \\
 & \textbf{where}\ s_2 = last(sl_2) \\
 & \qquad (s_1 ⨾ sfx) = sl_1 - front(sl_2) \\
= & \text{“ Apply } \textbf{where}\text{-clause ”} \\
 & sref(last(sl_1 - front(sl_2)))
\end{array}
$$

End Case Split

$$
\begin{array}{ll}
 & sref(last(sl_1 - front(sl_2))) \\
= & \text{“ } last(\sigma - \tau) = last(\sigma), \text{ when } \sigma - \tau \neq \langle\rangle \text{ ”} \\
 & sref(last(sl_1)) \\
= & \text{“ [ER:def]:p18 backwards ”} \\
 & eqvref(sl_1)
\end{array}
$$

□

A.1.12 Proof

of [DF:len]:p22

$$\#(sl_1 \ddots sl_2) = 1 + \#sl_1 - \#sl_2$$

$$
\begin{array}{ll}
 & \#(sl_1 \ddots sl_2) \\
= & \quad \text{“ [DF:def]:p21 ”} \\
 & \#(ssub(slot', slot) \fatsemi sfx) \\
 & \textbf{where}\ slot = last(sl_2) \land (slot' \fatsemi sfx) = sl_1 - front(sl_2) \\
= & \quad \text{“ } \#(x \fatsemi \sigma) = 1 + \#\sigma \text{ ”} \\
 & 1 + \#sfx \\
 & \textbf{where}\ slot = last(sl_2) \land (slot' \fatsemi sfx) = sl_1 - front(sl_2) \\
= & \quad \text{“ drop } slot, slot' \text{ defns, and express } sfx \text{ as } tail(\ldots) \text{ ”} \\
 & 1 + \#(tail(sl_1 - front(sl_2))) \\
= & \quad \text{“ } tail \text{ reduces by 1, cancelling first one. ”} \\
 & \#(sl_1 - front(sl_2)) \\
= & \quad \text{“ } \#(\sigma - \tau) = \#\sigma - \#\tau \text{ ”} \\
 & \#sl_1 - \#(front(sl_1)) \\
= & \quad \text{“ } front \text{ reduces by 1 ”} \\
 & \#sl_1 - (\#(sl_1) - 1) \\
= & \quad \text{“ arithmetic ”} \\
 & 1 + \#sl_1 - \#sl_2
\end{array}
$$

□

A.1.13 Proof

of [EX:prefix]:p19

$$ss_1 \frown ss_2 \preccurlyeq ss_1 \frown ss_3 \equiv ss_2 \preccurlyeq ss_3$$

Proof by induction on ss_1.

Case 1 : $ss_1 = \langle s \rangle$

$$\begin{aligned}
& s \mathbin{⨾} ss_2 \preccurlyeq s \mathbin{⨾} ss_3 \\
\equiv \quad & \text{“ [EX:def]:p19 ”} \\
& front(s \mathbin{⨾} ss_2) < s \mathbin{⨾} ss_3 \land last(s \mathbin{⨾} ss_2) \preceq (s \mathbin{⨾} ss_3)(\#(s \mathbin{⨾} ss_2)) \\
\equiv \quad & \text{“ defn. } front, last, \# \text{ ”} \\
& s \mathbin{⨾} front(ss_2) < s \mathbin{⨾} ss_3 \land last(ss_2) \preceq (s \mathbin{⨾} ss_3)(\#ss_2 + 1) \\
\equiv \quad & \text{“ defn. } <\text{, indexing ”} \\
& front(ss_2) < ss_3 \land last(ss_2) \preceq (ss_3)(\#ss_2) \\
\equiv \quad & \text{“ [EX:def]:p19, backwards ”} \\
& ss_2 \preccurlyeq ss_3
\end{aligned}$$

Case 2 : Assume

$$ss_1 \frown ss_2 \preccurlyeq ss_1 \frown ss_3 \equiv ss_2 \preccurlyeq ss_3 \qquad \text{[EX:prefix:hyp1]}$$

to show

$$s \mathbin{⨾} ss_1 \frown ss_2 \preccurlyeq s \mathbin{⨾} ss_1 \frown ss_3 \equiv ss_2 \preccurlyeq ss_3$$

$$\begin{aligned}
& s \mathbin{⨾} ss_1 \frown ss_2 \preccurlyeq s \mathbin{⨾} ss_1 \frown ss_3 \\
\equiv \quad & \text{“ [EX:def]:p19 ”} \\
& front(s \mathbin{⨾} ss_1 \frown ss_2) < s \mathbin{⨾} ss_1 \frown ss_3 \land last(s \mathbin{⨾} ss_1 \frown ss_2) \preceq (s \mathbin{⨾} ss_1 \frown ss_3)(\#(s \mathbin{⨾} ss_1 \frown ss_2)) \\
\equiv \quad & \text{“ defns. } front, last \text{ and } \#\text{, using } (x \mathbin{⨾} \sigma) \frown \tau = x \mathbin{⨾} (\sigma \frown \tau)\text{. ”} \\
& s \mathbin{⨾} front(ss_1 \frown ss_2) < s \mathbin{⨾} ss_1 \frown ss_3 \land last(ss_1 \frown ss_2) \preceq (s \mathbin{⨾} ss_1 \frown ss_3)(\#(ss_1 \frown ss_2) + 1) \\
\equiv \quad & \text{“ defn. } <\text{, indexing ”} \\
& front(ss_1 \frown ss_2) < ss_1 \frown ss_3 \land last(ss_1 \frown ss_2) \preceq (ss_1 \frown ss_3)(\#(ss_1 \frown ss_2)) \\
\equiv \quad & \text{“ [EX:def]:p19, backwards ”} \\
& ss_1 \frown ss_2 \preccurlyeq ss_1 \frown ss_3 \\
\equiv \quad & \text{“ [EX:prefix:hyp1]:p87 ”} \\
& ss_2 \preccurlyeq ss_3
\end{aligned}$$

□

A.1.14 Proof

of [EX:sngl]:p19

$$\langle s_1 \rangle \preccurlyeq s_2 \mathbin{⨾} ss \equiv s_1 \preceq s_2$$

$$\begin{array}{ll}
 & \langle s_1 \rangle \preccurlyeq s_2 \mathbin{⨾} ss \\
\equiv & \quad \text{“ [EX:def]:p19 ”} \\
 & front(\langle s_1 \rangle) < s_2 \mathbin{⨾} ss \land last(\langle s_1 \rangle) \preceq (s_2 \mathbin{⨾} ss)(\#\langle s_1 \rangle) \\
\equiv & \quad \text{“ defn. } front, \# \text{ ”} \\
 & \langle\rangle < s_2 \mathbin{⨾} ss \land s_1 \preceq (s_2 \mathbin{⨾} ss)(1) \\
\equiv & \quad \text{“ seq. ordering, seq. indexing ”} \\
 & \textsc{True} \land s_1 \preceq s_2 \\
\equiv & \quad \text{“ logic ”} \\
 & s_1 \preceq s_2 \\
\Box &
\end{array}$$

A.1.15 Proof

of [EX;EX]:p20

[EX;EX] $\quad (slots \preccurlyeq slots'); (slots \preccurlyeq slots') = (slots \preccurlyeq slots')$

$((slots \preccurlyeq slots'); (slots \preccurlyeq slots'))$

$\equiv$ " [Seq:def]:p32, drop un-used obs. vars. "

$\exists slots_0 \bullet slots \preccurlyeq slots_0 \wedge slots_0 \preccurlyeq slots'$

$\equiv$ " [EX:trans]:p19 "

$\exists slots_0 \bullet slots \preccurlyeq slots_0 \wedge slots_0 \preccurlyeq slots' \wedge slots \preccurlyeq slots'$

$\equiv$ " last conjunct free for bound vars "

$slots \preccurlyeq slots' \wedge \exists slots_0 \bullet slots \preccurlyeq slots_0 \wedge slots_0 \preccurlyeq slots'$

$\equiv$ " [EX:refl]:p19 "

$slots \preccurlyeq slots \wedge slots \preccurlyeq slots' \wedge \exists slots_0 \bullet slots \preccurlyeq slots_0 \wedge slots_0 \preccurlyeq slots'$

$\equiv$ " $(P(c) \wedge \exists x \bullet P(c)) \equiv P(c)$, here x is $slots_0$, c is $slots$ "

$slots \preccurlyeq slots \wedge slots \preccurlyeq slots'$

$\equiv$ " [EX:refl]:p19 "

$slots \preccurlyeq slots'$

□

A.1.16 Proof

of [SSEQV:expand]:p20

$$slots_1 \cong slots_2 = front(slots_1) = front(slots_2) \land last(slots_1) \approx last(slots_2)$$

$slots_1 \cong slots_2$

$\equiv$ “ [SEQV:def]:p20 ”

$slots_1 \preccurlyeq slots_2 \land slots_2 \preccurlyeq slots_1$

$\equiv$ “ [EX:def]:p19 ”

$front(slots_1) < slots_2 \land last(slots_1) \preceq slots_2(\#slots_1)$
$\land front(slots_2) < slots_1 \land last(slots_2) \preceq slots_1(\#slots_2)$

$\equiv$ “ [Seq:FrontLT:eqv] ”

$front(slots_1) = front(slots_2)$
$\land last(slots_1) \preceq slots_2(\#slots_1) \land last(slots_2) \preceq slots_1(\#slots_2)$

$\equiv$ “ [Seq:Front:len] ”

$front(slots_1) = front(slots_2) \land \#slots_1 = \#slots_2$
$\land last(slots_1) \preceq slots_2(\#slots_1) \land last(slots_2) \preceq slots_1(\#slots_2)$

$\equiv$ “ Liebniz ”

$front(slots_1) = front(slots_2) \land \#slots_1 = \#slots_2$
$\land last(slots_1) \preceq slots_2(\#slots_2) \land last(slots_2) \preceq slots_1(\#slots_1)$

$\equiv$ “ [Seq:Last:index], noting sequences are non-empty ”

$front(slots_1) = front(slots_2) \land \#slots_1 = \#slots_2$
$\land last(slots_1) \preceq last(slots_2) \land last(slots_2) \preceq last(slots_1)$

$\equiv$ “ [SEQV:def] ”

$front(slots_1) = front(slots_2) \land \#slots_1 = \#slots_2 \land last(slots_1) \approx last(slots_2)$

$\equiv$ “ [Seq:Front:len], reversed. ”

$front(slots_1) = front(slots_2) \land last(slots_1) \approx last(slots_2)$

□

A.2 Useful Sequence Shorthands

[F:s-hand]	$F = front$
[L:s-hand]	$L = last$
[H:s-hand]	$H = head$
[T:s-hand]	$T = tail$

A.2.1 Proof

of [CAT:assoc]:p21

$$sl_1 \sharp\!\sharp (sl_2 \sharp\!\sharp sl_3) \quad = \quad (sl_1 \sharp\!\sharp sl_2) \sharp\!\sharp sl_3$$

We use the shorthands on p91.
We let $sl_i = s_i \between ss_i$, noting that

$$\begin{array}{ll} & (s_i \between ss_i) \sharp\!\sharp (s_j \between ss_j) \\ = & \quad \text{“ [CAT:def]:p21 ”} \\ & (s_i \sharp s_j) : ss_j \lhd ss_i = \langle\rangle \rhd s_i \between (rr_i \frown (r_i \sharp s_j) \between ss_j) \\ & \textbf{where} \quad rr_i \fatsemi r_i = ss_i \\ & \text{[CAT:assoc:def']} \end{array}$$

We reformulate the goal:

$$(s_1 \between ss_1) \sharp\!\sharp ((s_2 \between ss_2) \sharp\!\sharp (s_3 \between ss_3)) \quad = \quad ((s_1 \between ss_1) \sharp\!\sharp (s_2 \between ss_2)) \sharp\!\sharp (s_3 \between ss_3)$$

Looking at the expanded definition of $\sharp\!\sharp$ it is clear we need a case split on both $ss_1 = \langle\rangle$ and $ss_2 = \langle\rangle$.
We use the following definitions:

$$\begin{array}{rl} \text{[CAT:sngl]} & (s_i \between \langle\rangle) \sharp\!\sharp (s_j \between ss_j) = (s_i \sharp s_j) \between ss_j \\ \text{[CAT:many]} & (s_i \between (rr_i \fatsemi r_i)) \sharp\!\sharp (s_j \between ss_j) = s_i : (rr_i \frown ((r_i \sharp s_j) \between ss_j)) \\ \text{[CAT:click]} & (rr_i \fatsemi r_i) \sharp\!\sharp (s_j \between ss_j) = rr_i \frown ((r_i \sharp s_j) \between ss_j) \\ & = (rr_i \fatsemi (r_i \sharp s_j)) \frown ss_j \end{array}$$

Case 1 $ss_1 = \langle\rangle \land ss_2 = \langle\rangle$

Lhs:

$$\begin{aligned}
& (s_1 \fatsemi \langle\rangle) \sharp\!\sharp ((s_2 \fatsemi \langle\rangle) \sharp\!\sharp (s_3 \fatsemi ss_3)) \\
= & \quad \text{“ [CAT:sngl]:p92 ”} \\
& (s_1 \fatsemi \langle\rangle) \sharp\!\sharp (s_2 \sharp s_3) \fatsemi ss_3
\end{aligned}$$

Rhs:

$$\begin{aligned}
& ((s_1 \fatsemi \langle\rangle) \sharp\!\sharp (s_2 \fatsemi \langle\rangle)) \sharp\!\sharp (s_3 \fatsemi ss_3) \\
= & \quad \text{“ [CAT:sngl]:p92 ”} \\
& (s_1 \sharp s_2) \fatsemi \langle\rangle \sharp\!\sharp (s_3 \fatsemi ss_3) \\
= & \quad \text{“ [CAT:sngl]:p92 ”} \\
& ((s_1 \sharp s_2) \sharp s_3) \fatsemi ss_3 \\
= & \quad \text{“ [sadd:assoc]:p17 ”} \\
& (s_1 \sharp (s_2 \sharp s_3)) \fatsemi ss_3 \\
= & \quad \text{“ [CAT:sngl]:p92, backwards ”} \\
& (s_1 \fatsemi \langle\rangle) \sharp\!\sharp (s_2 \sharp s_3) \fatsemi ss_3
\end{aligned}$$

Case 1 is OK.

Case 2 $ss_1 = \langle\rangle \land ss_2 = rr_2 \fatsemi r_2$

Lhs:

$$\begin{aligned}
& (s_1 \fatsemi \langle\rangle) \sharp\!\sharp ((s_2 \fatsemi rr_2 \fatsemi r_2) \sharp\!\sharp (s_3 \fatsemi ss_3)) \\
= & \quad \text{“ [CAT:many]:p92 ”} \\
& (s_1 \fatsemi \langle\rangle) \sharp\!\sharp (s_2 \fatsemi (rr_2 \frown (r_2 \sharp s_3) \fatsemi ss_3) \\
= & \quad \text{“ [CAT:sngl]:p92 ”} \\
& (s_1 \sharp s_2) \fatsemi (rr_2 \frown (r_2 \sharp s_3) \fatsemi ss_3)
\end{aligned}$$

Rhs:

$$\begin{aligned}
& ((s_1 \fatsemi \langle\rangle 1) \sharp\!\sharp (s_2 \fatsemi rr_2 \fatsemi r_2)) \sharp\!\sharp (s_3 \fatsemi ss_3) \\
= & \quad \text{“ [CAT:sngl]:p92 ”} \\
& ((s_1 \sharp s_2) \fatsemi rr_2 \fatsemi r_2) \sharp\!\sharp (s_3 \fatsemi ss_3) \\
= & \quad \text{“ [CAT:many]:p92 ”} \\
& (s_1 \sharp s_2) \fatsemi (rr_2 \frown (r_2 \sharp\!\sharp s_3) \fatsemi ss_3)
\end{aligned}$$

Case 2 is OK

Case 3 $ss_1 = rr_1 \fatsemi r_1 \land ss_2 = \langle\rangle$
Lhs:

$$\begin{aligned}
& (s_1 \circ rr_1 \fatsemi r_1) \sharp ((s_2 \circ \langle\rangle) \sharp (s_3 \circ ss_3)) \\
= & \quad \text{“ [CAT:sngl]:p92 ”} \\
& (s_1 \circ rr_1 \fatsemi r_1) \sharp ((s_2 \sharp s_3) \circ ss_3) \\
= & \quad \text{“ [CAT:many]:p92 ”} \\
& s_1 \circ (rr_1 \frown (r_1 \sharp (s_2 \sharp s_3)) \circ ss_3) \\
= & \quad \text{“ [sadd:assoc]:p17 ”} \\
& s_1 \circ (rr_1 \frown ((r_1 \sharp s_2) \sharp s_3) \circ ss_3)
\end{aligned}$$

Rhs:

$$\begin{aligned}
& ((s_1 \circ rr_1 \fatsemi r_1) \sharp (s_2 \circ \langle\rangle)) \sharp (s_3 \circ ss_3) \\
= & \quad \text{“ [CAT:many]:p92 ”} \\
& (s_1 \circ (rr_1 \frown (r_1 \sharp s_2 (\circ \langle\rangle)))) \sharp (s_3 \circ ss_3) \\
= & \quad \text{“ [CAT:sngl]:p92 ”} \\
& s_1 \circ (rr_1 \frown ((r_1 \sharp s_2) \sharp s_3) \circ ss_3)
\end{aligned}$$

Case3 is OK

Case 4 $ss_1 = rr_1 \fatsemi r_1 \land ss_2 = rr_2 \fatsemi r_2$
Lhs:

$$\begin{aligned}
& (s_1 \circ rr_1 \fatsemi r_1) \sharp ((s_2 \circ rr_2 \fatsemi r_2) \sharp (s_3 \circ ss_3)) \\
= & \quad \text{“ [CAT:many]:p92 ”} \\
& (s_1 \circ rr_1 \fatsemi r_1) \sharp (s_2 \circ (rr_2 \frown ((r_2 \sharp s_3) \circ ss_3))) \\
= & \quad \text{“ [CAT:many]:p92 ”} \\
& s_1 \circ (rr_1 \frown (r_1 \sharp s_2) \circ (rr_2 \frown ((r_2 \sharp s_3) \circ ss_3)))
\end{aligned}$$

Rhs:

$$\begin{aligned}
& ((s_1 \circ rr_1 \fatsemi r_1) \sharp (s_2 \circ rr_2 \fatsemi r_2)) \sharp (s_3 \circ ss_3) \\
= & \quad \text{“ [CAT:many]:p92 ”} \\
& ((s_1 \circ (rr_1 \frown (r_1 \sharp s_2) \circ ((rr_2 \fatsemi r_2)) \sharp (s_3 \circ ss_3) \\
= & \quad \text{“ [CAT:click]:p92 ”} \\
& s_1 \circ (rr_1 \frown (r_1 \sharp s_2) \circ (rr_2 \frown ((r_2 \sharp s_3) \circ ss_3)))
\end{aligned}$$

Case 4 is OK □

A.2.2 Proof

of [CAT:PFX]:p21

$$ss \preccurlyeq ss \,\sharp\!\sharp\, tt$$

$$
\begin{array}{ll}
 & ss \preccurlyeq ss \,\sharp\!\sharp\, tt \\
= & \quad \text{“ [CAT:def]:p21 ”} \\
 & ss \preccurlyeq front(ss) \frown \langle last(ss) \sharp head(tt) \rangle \frown tail(tt) \\
= & \quad \text{“ [EX:def]:p19 ”} \\
 & front(ss) < front(ss) \frown \langle last(ss) \sharp head(tt) \rangle \frown tail(tt) \\
 & \wedge\ last(ss) \preceq (front(ss) \frown \langle last(ss) \sharp head(tt) \rangle \frown tail(tt))(\# ss) \\
= & \quad \text{“ } \sigma < \sigma \frown \tau, \text{ when } \tau \neq \langle\rangle \text{ ”} \\
 & last(ss) \preceq (front(ss) \frown \langle last(ss) \sharp head(tt) \rangle \frown tail(tt))(\# ss) \\
= & \quad \text{“ [Seq:Front:len-index]:p214 ”} \\
 & last(ss) \preceq head(\langle last(ss) \sharp head(tt) \rangle \frown tail(tt)) \\
= & \quad \text{“ defn. of } head \text{ ”} \\
 & last(ss) \preceq last(ss) \sharp head(tt) \\
= & \quad \text{“ [sadd:prefix]:p17 ”} \\
 & \mathbf{true} \\
\square &
\end{array}
$$

A.2.3 Proof

of [CAT:ER:last]:p21

$$eqvref(sl_1 \sharp\sharp\, sl_2) = eqvref(sl_2)$$

$$
\begin{array}{ll}
 & eqvref(sl_1 \sharp\sharp\, sl_2) \\
= & \quad \text{“ [CAT:def]:p21 ”} \\
 & eqvref(front(sl_1) \frown \langle last(sl_1) \sharp\, head(sl_2)\rangle \frown tail(sl_2)) \\
= & \quad \text{“ [ER:def]:p18 ”} \\
 & sref(last(front(sl_1) \frown \langle last(sl_1) \sharp\, head(sl_2)\rangle \frown tail(sl_2)))
\end{array}
$$

We do a case split on $tail(sl_2) = \langle\rangle$.

Case Split $tail(sl_2) \neq \langle\rangle$

$$
\begin{array}{ll}
 & sref(last(front(sl_1) \frown \langle last(sl_1) \sharp\, head(sl_2)\rangle \frown tail(sl_2))) \\
= & \quad \text{“ } last(\sigma \frown \tau) = last(\tau), \text{ if } \tau \neq \langle\rangle \text{ ”} \\
 & sref(last(tail(sl_2))) \\
= & \quad \text{“ } last(\sigma) = last(x : \sigma), \text{ if } \sigma \neq \langle\rangle \text{ ”} \\
 & sref(last((sl_2)) \\
= & \quad \text{“ [ER:def]:p18 backwards ”} \\
 & eqvref(sl_2)
\end{array}
$$

Case Split $tail(sl_2) = \langle\rangle$

$$
\begin{array}{ll}
 & sref(last(front(sl_1) \frown \langle last(sl_1) \sharp\, head(sl_2)\rangle \frown tail(sl_2))) \\
= & \quad \text{“ } last(\sigma \frown \tau) = last(\sigma), \text{ if } \tau = \langle\rangle \text{ ”} \\
 & sref(last(front(sl_1) \frown \langle last(sl_1) \sharp\, head(sl_2)\rangle)) \\
= & \quad \text{“ } last(\sigma \frown \langle x\rangle) = x \text{ ”} \\
 & sref(\langle last(sl_1) \sharp\, head(sl_2)\rangle) \\
= & \quad \text{“ [sadd:ref]:p13 ”} \\
 & sref(head(sl_2)) \\
= & \quad \text{“ } tail(\sigma) = \langle\rangle \Rightarrow head(\sigma) = last(\sigma) \text{ ”} \\
 & sref(last(sl_2)) \\
= & \quad \text{“ [ER:def]:p18 backwards ”} \\
 & eqvref(sl_2)
\end{array}
$$

□

A.2.4 Proof

of [CAT:eqv]:p21

$$sl_1 \cong sl_1 \sharp sl_2 \equiv \exists\, r_2 \bullet sl_2 = \langle snull(r_2) \rangle$$

$sl_1 \cong sl_1 \sharp sl_2$

$\equiv$ “ [CAT:def]:p21 ”

$sl_1 \cong front(sl_1) \frown \langle last(sl_1) \sharp head(sl_2) \rangle \frown tail(sl_2)$

$\equiv$ “ [SSEQV:def]:p20 ”

$sl_1 \preccurlyeq front(sl_1) \frown \langle last(sl_1) \sharp head(sl_2) \rangle \frown tail(sl_2)$

$\wedge\, front(sl_1) \frown \langle last(sl_1) \sharp head(sl_2) \rangle \frown tail(sl_2) \preccurlyeq sl_1$

$\equiv$ “ [EX:def]:p19 ”

$front(sl_1) < ss \wedge last(sl_1) \preceq ss(\# sl_1)$

$\wedge\, front(ss) < sl_1 \wedge last(ss) \preceq sl_1(\# ss)$

where $ss = front(sl_1) \frown \langle last(sl_1) \sharp head(sl_2) \rangle \frown tail(sl_2)$

$\equiv$ “ [Seq:Front:Cat:Le]:p215, $xx = sl_1$, $tt = \langle last(sl_1) \sharp head(sl_2) \rangle \frown tail(sl_2)$ ”

$front(sl_1) < ss \wedge last(sl_1) \preceq ss(\# sl_1)$

$\wedge\, front(ss) < sl_1 \wedge last(ss) \preceq sl_1(\# ss)$

where $ss = front(sl_1) \frown \langle last(sl_1) \sharp head(sl_2) \rangle \wedge tail(sl_2) = \langle\rangle$

$\equiv$ “ $front(\sigma \frown \langle x \rangle) = \sigma$, $last(\sigma \frown \langle x \rangle) = x$ ”

$front(sl_1) < ss \wedge last(sl_1) \preceq ss(\# sl_1)$

$\wedge\, front(sl_1) < sl_1 \wedge last(sl_1) \sharp head(sl_2) \preceq sl_1(\# ss)$

where $ss = front(sl_1) \frown \langle last(sl_1) \sharp head(sl_2) \rangle \wedge tail(sl_2) = \langle\rangle$

$\equiv$ “ $xx < xx \frown tt$, $tt \neq \langle\rangle$, and $front(xx) < xx$, $xx \neq \langle\rangle$ ”

$last(sl_1) \preceq ss(\# sl_1)$

$\wedge\, last(sl_1) \sharp head(sl_2) \preceq sl_1(\# ss)$

where $ss = front(sl_1) \frown \langle last(sl_1) \sharp head(sl_2) \rangle \wedge tail(sl_2) = \langle\rangle$

$\equiv$ “ $ss = front(xx) \frown \langle t \rangle \Rightarrow \# ss = \# xx$ ”

$last(sl_1) \preceq ss(\# ss) \wedge last(sl_1) \sharp head(sl_2) \preceq sl_1(\# sl_1)$

where $ss = front(sl_1) \frown \langle last(sl_1) \sharp head(sl_2) \rangle \wedge tail(sl_2) = \langle\rangle$

$\equiv$ “ $\sigma(\# \sigma) = last(\sigma)$ ”

$last(sl_1) \preceq last(ss) \wedge last(sl_1) \sharp head(sl_2) \preceq last(sl_1)$

where $ss = front(sl_1) \frown \langle last(sl_1) \sharp head(sl_2) \rangle \wedge tail(sl_2) = \langle\rangle$

$\equiv$ “ [Seq:Last:def:alt]:p213 ”

$last(sl_1) \preceq last(sl_1) \sharp head(sl_2) \wedge last(sl_1) \sharp head(sl_2) \preceq last(sl_1) \wedge tail(sl_2) = \langle\rangle$

$\equiv$ “ [SEQV:def]:p20 ”

$last(sl_1) \approx last(sl_1) \sharp head(sl_2) \wedge tail(sl_2) = \langle\rangle$

$\equiv$ “ [sadd:eqv:unit]:p13 ”

$(\exists\, r_2 \bullet head(sl_2) = snull(r_2)) \wedge tail(sl_2) = \langle\rangle$

$\equiv$ “ expand scope, list property ”

$\exists\, r_2 \bullet sl_2 = \langle snull(r_2) \rangle \Box$

A.2.5 Proof

of [CAT:equal]:p21

$$sl_1 = sl_1 \sharp\!\sharp\, sl_2 \equiv sl_2 = \langle snull(eqvref(sl_1))\rangle$$

$$
\begin{array}{ll}
 & sl_1 = sl_1 \sharp\!\sharp\, sl_2 \\
\equiv & \quad \text{“ [CAT:def]:p21 ”} \\
 & sl_1 = front(sl_1) \frown \langle last(sl_1) \sharp\, head(sl_2)\rangle \frown tail(sl_2) \\
\equiv & \quad \text{“ } \sigma = front(\sigma) \frown \tau \Rightarrow \tau = \langle last(\sigma)\rangle \text{ ”} \\
 & last(sl_1) = last(sl_1) \sharp\, head(sl_2) \wedge tail(sl_2) = \langle\rangle \\
\equiv & \quad \text{“ [sadd:unit]:p13 ”} \\
 & head(sl_2) = snull(sref(last(sl_1))) \wedge tail(sl_2) = \langle\rangle \\
\equiv & \quad \text{“ list property ”} \\
 & sl_2 = \langle snull(sref(last(sl_1)))\rangle \\
\equiv & \quad \text{“ [ER:def]:p18, backwards ”} \\
 & sl_2 = \langle snull(eqvref(sl_1))\rangle \\
\Box &
\end{array}
$$

A.2.6 Proof

of [CAT:len]:p21

$$\#(sl_1 \sharp\!\!\sharp\ sl_2) = \#(sl_1) + \#(sl_2) - 1$$

$$
\begin{aligned}
& \#(sl_1 \sharp\!\!\sharp\ sl_2) \\
= \quad & \text{“ [CAT:def]:p21 ”} \\
& \#(front(sl_1) \frown \langle last(sl_1) \sharp head(sl_2) \rangle \frown tail(sl_2)) \\
= \quad & \text{“ } \#(\sigma \frown \tau) = \#\sigma + \#\tau \text{ ”} \\
& \#front(sl_1) + \#\langle last(sl_1) \sharp head(sl_2) \rangle + \#tail(sl_2) \\
= \quad & \text{“ } \mathit{front} \text{ and } \mathit{tail} \text{ shrink by one, } \#\langle x \rangle = 1 \text{ ”} \\
& \#(sl_1) - 1 + 1 + \#(sl_2) - 1 \\
= \quad & \text{“ arithmetic ”} \\
& \#(sl_1) + \#(sl_2) - 1
\end{aligned}
$$

□

A.2.7 Proof

of [DF:equal]:p22

$$slots = dif(slots, sln) \equiv \exists\, rn \bullet sln = \langle snull(rn) \rangle$$

$slots = dif(slots, sln)$

$\equiv$ “ [DF:def]:p21 ”

$slots = ssub(slot', slot) \fatsemi sfx$

where $slot = last(sln) \land (slot' \fatsemi sfx) = slots - front(sln)$

$\equiv$ “ pattern-match $\sigma = x \fatsemi \tau$ ”

$head(slots) = ssub(slot', slot) \land tail(slots) = sfx$

where $slot = last(sln) \land slot' = head(slots - front(sln)) \land sfx = tail(slots - front(sln))$

$\equiv$ “ rewrite $slot$, sfx ”

$head(slots) = ssub(slot', last(sln)) \land tail(slots) = tail(slots - front(sln))$

where $slot' = head(slots - front(sln))$

$\equiv$ “ [Seq:TailSub]:p215 ”

$head(slots) = ssub(slot', last(sln)) \land front(sln) = \langle\rangle$

where $slot' = head(slots - front(sln))$

$\equiv$ “ $\sigma - \langle\rangle = \sigma$ ”

$head(slots) = ssub(slot', last(sln)) \land front(sln) = \langle\rangle$

where $slot' = head(slots)$

$\equiv$ “ rewrite $slot'$ ”

$head(slots) = ssub(head(slots), last(sln)) \land front(sln) = \langle\rangle$

$\equiv$ “ [SSub:equal]:p14 ”

$(\exists\, rn \bullet last(sln) = snull(rn)) \land front(sln) = \langle\rangle$

$\equiv$ “ extend $\exists$ scope ”

$\exists\, rn \bullet last(sln) = snull(rn) \land front(sln) = \langle\rangle$

$\equiv$ “ $last(\sigma) = x \land front(\sigma) = \langle\rangle \equiv \sigma = \langle x \rangle$ ”

$\exists\, rn \bullet sln = \langle snull(rn) \rangle$

$\square$

A.2.8 Proof

of [DF:self]:p22

$$dif(slots, slots) = \langle snull(eqvref(slots)) \rangle$$

$$
\begin{array}{ll}
 & dif(slots, slots) \\
= & \quad \text{“ [DF:def]:p21 ”} \\
 & ssub(slot', slot) \mathbin{⨾} sfx \\
 & \land\ slot = last(slots) \\
 & \land\ (slot' \mathbin{⨾} sfx) = slots - front(slots) \\
= & \quad \text{“ } \sigma - front(\sigma) = \langle last(\sigma) \rangle \text{ ”} \\
 & ssub(slot', slot) \mathbin{⨾} sfx \\
 & \land\ slot = last(slots) \\
 & \land\ (slot' \mathbin{⨾} sfx) = \langle last(slots) \rangle \\
= & \quad \text{“ pattern match ”} \\
 & ssub(slot', slot) \mathbin{⨾} sfx \\
 & \land\ slot = last(slots) \\
 & \land\ slot' = last(slots) \land sfx = \langle\rangle \\
= & \quad \text{“ Liebniz } slot, slot', sfx \text{ ”} \\
 & SSub(last(slot), last(slot)) \mathbin{⨾} \langle\rangle \\
= & \quad \text{“ [SSub:self]:p17, } x \mathbin{⨾} \langle\rangle = \langle x \rangle \text{ ”} \\
 & \langle snull(sref(last(slot))) \rangle \\
= & \quad \text{“ [ER:def]:p18, backwards ”} \\
 & \langle snull(eqvref(slots)) \rangle
\end{array}
$$

□

A.2.9 Proof

of [DF:nil]:p22

$$dif(slots, \langle snull(r) \rangle) = slots$$

$$
\begin{array}{ll}
 & dif(slots, \langle snull(r) \rangle) \\
= & \quad \text{“ [DF:def]:p21 ”} \\
 & ssub(slot', slot) \mathbin{⨾} sfx \\
 & \land\ slot = last(\langle snull(r) \rangle) \\
 & \land\ (slot' \mathbin{⨾} sfx) = slots - front(\langle snull(r) \rangle) \\
= & \quad \text{“ defn. } last,\ front \text{ ”} \\
 & ssub(slot', slot) \mathbin{⨾} sfx \land slot = snull(r) \land (slot' \mathbin{⨾} sfx) = slots - \langle\rangle \\
= & \quad \text{“ } \sigma - \langle\rangle = \sigma \text{ ”} \\
 & ssub(slot', slot) \mathbin{⨾} sfx \land slot = snull(r) \land (slot' \mathbin{⨾} sfx) = slots \\
= & \quad \text{“ pattern match ”} \\
 & ssub(slot', slot) \mathbin{⨾} sfx \land slot = snull(r) \land slot' = head(slots) \land sfx = tail(slots) \\
= & \quad \text{“ Liebniz } slot, slots', sfx \text{ ”} \\
 & ssub(head(slots), snull(r)) \mathbin{⨾} tail(slots) \\
= & \quad \text{“ [SSub:nil]:p17 ”} \\
 & head(slots) \mathbin{⨾} tail(slots) \\
= & \quad \text{“ property of } head \text{ and } tail \text{ ”} \\
 & slots \\
\Box &
\end{array}
$$

A.2.10 Proof

of [DF:Null:equal]:p22

$$slots' \fallingdotseq slots = \langle snull(r') \rangle \equiv slots' \cong slots \land eqvref(slots') = r'$$

$$
\begin{array}{lll}
 & & slots' \fallingdotseq slots = \langle snull(r') \rangle \\
 & \equiv & \text{“ [DF:def]:p21 ”} \\
 & & ssub(slot', slot) \mathbin{⨾} sfx = \langle snull(r') \rangle \\
\textbf{where} & & slot = last(slots) \\
 & & (slot' \mathbin{⨾} sfx) = slots' - front(slots) \\
 & \equiv & \text{“ list properties ”} \\
 & & ssub(slot', slot) = snull(r') \land sfx = \langle\rangle \\
\textbf{where} & & slot = last(slots) \\
 & & slot' = head(slots' - front(slots)) \\
 & & sfx = tail(slots' - front(slots)) \\
 & \equiv & \text{“ eliminate } sfx\text{, more list properties ”} \\
 & & ssub(slot', slot) = snull(r') \\
\textbf{where} & & slot = last(slots) \\
 & & slots' - front(slots) = \langle slot' \rangle \\
 & & front(slots) = front(slots') \\
 & \equiv & \text{“ [SSub:eqv]:p14, given } r' = sref(slots') \text{ ”} \\
 & & slot' \approx slot \land sref(slot') = r' \\
\textbf{where} & & slot = last(slots) \\
 & & slots' - front(slots) = \langle slot' \rangle \\
 & & front(slots) = front(slots') \\
 & \equiv & \text{“ Liebniz, last line ”} \\
 & & slot' \approx slot \land sref(slot') = r' \\
\textbf{where} & & slot = last(slots) \\
 & & slots' - front(slots') = \langle slot' \rangle \\
 & & front(slots) = front(slots') \\
 & \equiv & \text{“ list properties ”} \\
 & & slot' \approx slot \land sref(slot') = r' \\
\textbf{where} & & slot = last(slots) \\
 & & slot' = last(slots') \\
 & & front(slots) = front(slots') \\
 & \equiv & \text{“ [SSEQV:expand]:p20 ”} \\
 & & slots' \cong slots \\
 & & \land\ sref(slot') = r' \land slot' = last(slots') \\
 & \equiv & \text{“ [ER:def]:p18 ”} \\
 & & slots' \cong slots \land eqvref(slots') = r' \\
 & \Box &
\end{array}
$$

A.2.11 Proof

of [DF:Null:eqv]:p22

$$(\exists r' \bullet slots' \setminus\!\setminus slots = \langle snull(r') \rangle) \equiv slots' \cong slots$$

$$\begin{array}{ll}
 & (\exists r' \bullet slots' \setminus\!\setminus slots = \langle snull(r') \rangle) \\
\equiv & \quad \text{“ [DF:def]:p21 ”} \\
 & (\exists r' \bullet ssub(slot', slot) \mathbin{⨾} sfx = \langle snull(r') \rangle) \\
 & \textbf{where}\ slot = last(slots) \\
 & \qquad (slot' \mathbin{⨾} sfx) = slots' - front(slots) \\
\equiv & \quad \text{“ list properties ”} \\
 & (\exists r' \bullet ssub(slot', slot) = snull(r') \land sfx = \langle\rangle) \\
 & \textbf{where}\ slot = last(slots) \\
 & \qquad slot' = head(slots' - front(slots)) \\
 & \qquad sfx = tail(slots' - front(slots)) \\
\equiv & \quad \text{“ eliminate } sfx\text{, more list properties ”} \\
 & (\exists r' \bullet ssub(slot', slot) = snull(r')) \\
 & \textbf{where}\ slot = last(slots) \\
 & \qquad slots' - front(slots) = \langle slot' \rangle \\
 & \qquad front(slots) = front(slots') \\
\equiv & \quad \text{“ [SSub:eqv]:p14, given } r' = sref(slots') \text{ ”} \\
 & (\exists r' \bullet slot' \approx slot \land sref(slot') = r') \\
 & \textbf{where}\ slot = last(slots) \\
 & \qquad slots' - front(slots) = \langle slot' \rangle \\
 & \qquad front(slots) = front(slots') \\
\equiv & \quad \text{“ narrow scope; Liebniz, last line ”} \\
 & slot' \approx slot \land (\exists r' \bullet sref(slot') = r') \\
 & \textbf{where}\ slot = last(slots) \\
 & \qquad slots' - front(slots') = \langle slot' \rangle \\
 & \qquad front(slots) = front(slots') \\
\equiv & \quad \text{“ list properties, one-point rule ”} \\
 & last(slots') \approx last(slots) \land front(slots) = front(slots') \\
\equiv & \quad \text{“ [SSEQV:expand]:p20 ”} \\
 & slots' \cong slots \\
\Box &
\end{array}$$

A.2.12 Proof

of [DF:same]:p22

$$slots_b \preccurlyeq slots_a \land slots_b \preccurlyeq slots_c \Rightarrow \\ dif(slots_a, slots_b) = dif(slots_c, slots_b) \equiv slots_a = slots_c$$

$dif(slots_a, slots_b) = dif(slots_c, slots_b)$

$\equiv$ " [DF:def]:p21 "

$ssub(slot'_a, slot_a) \fatsemi sfx_a$

where $slot_a = last(slots_b)$

and $(slot'_a \fatsemi sfx_a) = slots_a - front(slots_b)$

$=$

$ssub(slot'_c, slot_c) \fatsemi sfx_c$

where $slot_c = last(slots_b)$

and $(slot'_c \fatsemi sfx_c) = slots_c - front(slots_b)$

$\equiv$ " re-organise "

$ssub(slot'_a, slot_a) \fatsemi sfx_a = ssub(slot'_c, slot_c) \fatsemi sfx_c$

$\land\ slot_a = last(slots_b)$

$\land\ (slot'_a \fatsemi sfx_a) = slots_a - front(slots_b)$

$\land\ slot_c = last(slots_b)$

$\land\ (slot'_c \fatsemi sfx_c) = slots_c - front(slots_b)$

$\equiv$ " pattern matching "

$ssub(slot'_a, slot_a) \fatsemi sfx_a = ssub(slot'_c, slot_c) \fatsemi sfx_c$

$\land\ slot_a = last(slots_b)$

$\land\ slot'_a = head(slots_a - front(slots_b))$

$\land\ sfx_a = tail(slots_a - front(slots_b))$

$\land\ slot_c = last(slots_b)$

$\land\ slot'_c = head(slots_c - front(slots_b))$

$\land\ sfx_c = tail(slots_c - front(slots_b))$

$\equiv$ " defn. of list equality "

$ssub(slot'_a, slot_a) = ssub(slot'_c, slot_c)$

$\land\ sfx_a = sfx_c$

$\land\ slot_a = last(slots_b)$

$\land\ slot'_a = head(slots_a - front(slots_b))$

$\land\ sfx_a = tail(slots_a - front(slots_b))$

$\land\ slot_c = last(slots_b)$

$\land\ slot'_c = head(slots_c - front(slots_b))$

$\land\ sfx_c = tail(slots_c - front(slots_b))$

$\equiv$ " rearrange "

$ssub(slot'_a, slot_a) = ssub(slot'_c, slot_c)$
$\land\ sfx_a = sfx_c$
$\land\ sfx_a = tail(slots_a - front(slots_b))$
$\land\ sfx_c = tail(slots_c - front(slots_b))$
$\land\ slot_a = last(slots_b)$
$\land\ slot'_a = head(slots_a - front(slots_b))$
$\land\ slot_c = last(slots_b)$
$\land\ slot'_c = head(slots_c - front(slots_b))$
$\equiv$ “ Liebniz $sfx_a, sfx_c, slot_a, slot_c$ ”
$ssub(slot'_a, last(slots_a)) = ssub(slot'_c, last(slots_c))$
$\land\ tail(slots_a - front(slots_b)) = tail(slots_c - front(slots_b))$
$\land\ slot'_a = head(slots_a - front(slots_b))$
$\land\ slot'_c = head(slots_c - front(slots_b))$
$\equiv$ “ $tail(\sigma - \tau) = tail(\sigma) - \tau$ ”
$ssub(slot'_a, last(slots_a)) = ssub(slot'_c, last(slots_c))$
$\land\ tail(slots_a) - front(slots_b) = tail(slots_c) - front(slots_b)$
$\land\ slot'_a = head(slots_a - front(slots_b))$
$\land\ slot'_c = head(slots_c - front(slots_b))$
$\equiv$ “ $\sigma - \tau = \nu - \tau \equiv \sigma = \nu$ ”
$ssub(slot'_a, last(slots_a)) = ssub(slot'_c, last(slots_c))$
$\land\ tail(slots_a) = tail(slots_c)$
$\land\ slot'_a = head(slots_a - front(slots_b))$
$\land\ slot'_c = head(slots_c - front(slots_b))$
$\equiv$ “ $tail(\sigma) = tail(\tau) \Rightarrow last(\sigma) = last(\tau)$ ”
$ssub(slot'_a, last(slots_a)) = ssub(slot'_c, last(slots_c))$
$\land\ tail(slots_a) = tail(slots_c)$
$\land\ last(slots_a) = last(slots_c)$
$\land\ slot'_a = head(slots_a - front(slots_b))$
$\land\ slot'_c = head(slots_c - front(slots_b))$
$\equiv$ “ [SSub:same]:p17, given pre-condition ”
$slot'_a = slot'_c$
$\land\ last(slots_a) = last(slots_c)$
$\land\ tail(slots_a) = tail(slots_c)$
$\land\ slot'_a = head(slots_a - front(slots_b))$
$\land\ slot'_c = head(slots_c - front(slots_b))$
$\equiv$ “ Liebnix, $slot'_a, slot'_c$ ”

$$
\begin{aligned}
& last(slots_a) = last(slots_c) \\
& \land\ tail(slots_a) = tail(slots_c) \\
& \land\ head(slots_a - front(slots_b)) = head(slots_c - front(slots_b)) \\
\equiv\quad & \text{“ [Seq:HdSub:index']:p215, drop } last \text{ ”} \\
& tail(slots_a) = tail(slots_c) \land slots_a(\#slots_b) = slots_c(\#slots_b)
\end{aligned}
$$

We now do case analysis on $\#slots_b$.
Case 1 $\#slots_b = 1$:

$$
\begin{aligned}
& tail(slots_a) = tail(slots_c) \land slots_a(1) = slots_c(1) \\
\equiv\quad & \text{“ } head(\sigma) = \sigma(1) \text{ ”} \\
& tail(slots_a) = tail(slots_c) \land head(slots_a) = head(slots_c) \\
\equiv\quad & \text{“ defn. of list equality ”} \\
& slots_a = slots_c
\end{aligned}
$$

Case 2 $\#slots_b > 1$.

$$
\begin{aligned}
& tail(slots_a) = tail(slots_c) \land slots_a(\#slots_b) = slots_c(\#slots_b) \\
& \text{“ pre-condition implies } front(slots_b) \leq slots_a \text{, etc. ”} \\
\equiv\quad & \text{“ Case 2 implies } front(slots_b) \neq \langle\rangle \text{ ”} \\
& \text{“ So, } head(slots_a) = head(slots_b) \text{ etc. ”} \\
& tail(slots_a) = tail(slots_c) \\
& \land\ head(slots_a) = head(slots_b) \land head(slots_c) = head(slots_b) \\
& \land\ slots_a(\#slots_b) = slots_c(\#slots_b) \\
\equiv\quad & \text{“ Liebniz } head(slots_b) \text{ ”} \\
& tail(slots_a) = tail(slots_c) \\
& \land\ head(slots_a) = head(slots_c) \\
& \land\ slots_a(\#slots_b) = slots_c(\#slots_b) \\
\equiv\quad & \text{“ defn. list equality ”} \\
& slots_a = slots_c \land slots_a(\#slots_b) = slots_c(\#slots_b) \\
\equiv\quad & \text{“ } A \land B \land A \Rightarrow B \equiv A \text{ ”} \\
& slots_a = slots_c
\end{aligned}
$$

□

A.2.13 Proof

of [DF:ref]:p22

$$srefs(dif(slots', slots)) = srefs(slots' - front(slots))$$

$$
\begin{array}{lll}
 & & srefs(dif(slots', slots)) \\
 & = & \text{“ [DF:def]:p21 ”} \\
 & & srefs(ssub(slot', slot) \frown sfx) \\
\textbf{where} & & slot = last(slots) \\
 & & (slot' \frown sfx) = slots' - front(slots) \\
 & = & \text{“ meaning of } \textbf{where}\text{, pattern match ”} \\
 & & srefs(ssub(slot', slot) \frown sfx) \\
 & & \land\ slot = last(slots) \\
 & & \land\ slot' = head(slots' - front(slots)) \\
 & & \land\ sfx = tail(slots' - front(slots)) \\
 & = & \text{“ Liebniz } slot', slot, sfx \text{ ”} \\
 & & srefs(ssub(head(slots' - front(slots)), last(slots)) \frown tail(slots' - front(slots))) \\
 & = & \text{“ [RFS:def]:p18, defn. } map \text{ ”} \\
 & & sref(ssub(head(slots' - front(slots)), last(slots))) \frown srefs(tail(slots' - front(slots)) \\
 & = & \text{“ [ssub:ref]:p\textbf{??} ”} \\
 & & sref(head(slots' - front(slots)) \frown srefs(tail(slots' - front(slots)) \\
 & = & \text{“ defn. } map \text{ ”} \\
 & & srefs(slots' - front(slots)) \\
 & & \Box
\end{array}
$$

A.2.14 Proof

of [DF:subsub]:p22

$$slots_c \preccurlyeq slots_a \land slots_c \preccurlyeq slots_b \land slots_b \preccurlyeq slots_a$$
$$\Rightarrow dif(dif(slots_a, slots_c), dif(slots_b, slots_c)) = dif(slots_a, slots_b)$$

We expand the antecedents using [EX:def]:p19 so we know what we can assume:

[prf:DF:subsub:ante1]	$front(slots_c) \le slots_a$
[prf:DF:subsub:ante2]	$last(slots_c) \preceq slots_a(\#slots_c)$
[prf:DF:subsub:ante3]	$front(slots_c) \le slots_b$
[prf:DF:subsub:ante4]	$last(slots_c) \preceq slots_b(\#slots_c)$
[prf:DF:subsub:ante5]	$front(slots_b) \le slots_a$
[prf:DF:subsub:ante6]	$last(slots_b) \preceq slots_a(\#slots_b)$

We can re-code this with [EX:pfx]:p19 as follows:

[prf:DF:subsub:ante7]	$slots_c = pfx_c \frown \langle slot_c \rangle$
[prf:DF:subsub:ante8]	$slots_b = pfx_c \frown \langle slot_{b1} \rangle \frown sfx_b$
[prf:DF:subsub:ante9]	$slots_b = pfx_b \frown \langle slot_{b2} \rangle$
[prf:DF:subsub:ante10]	$slot_{b2} = slot_{b1} \lhd sfx_b = \langle\rangle \rhd last(sfx_b)$
[prf:DF:subsub:ante11]	$slots_a = pfx_b \frown \langle slot_a \rangle \frown sfx_a$
[prf:DF:subsub:ante12]	$slot_c \preceq slot_{b1}$
[prf:DF:subsub:ante12]	$slot_{b2} \preceq slot_a$

The rhs:

$$dif(slots_a, slots_b)$$
$$= \quad \text{“ [prf:DF:subsub:ante9,11] ”}$$
$$dif(pfx_b \frown \langle slot_a \rangle \frown sfx_a, pfx_b \fatsemi \langle slot_{b2} \rangle)$$
$$= \quad \text{“ [DF:pfx]:p21 ”}$$
$$ssub(slot_a, slot_{b2}) \fatsemi sfx_a$$

Now, the lhs:

$$dif(dif(slots_a, slots_c), dif(slots_b, slots_c))$$
$$= \quad \text{“ [prf:DF:subsub:ante7,8,11] ”}$$
$$dif($$
$$\quad dif(pfx_b \frown \langle slot_a \rangle \frown sfx_a, pfx_c \frown \langle slot_c \rangle),$$
$$\quad dif(pfx_c \frown \langle slot_{b1} \rangle \frown sfx_b, pfx_c \frown \langle slot_c \rangle)\ )$$
$$= \quad \text{“ [DF:pfx] ”}$$
$$dif($$
$$\quad dif(pfx_b \frown \langle slot_a \rangle \frown sfx_a, pfx_c \frown \langle slot_c \rangle),$$
$$\quad ssub(slot_{b1}, slot_c) \fatsemi sfx_b\ )$$

At this point we do case analysis on $sfx_b = \langle\rangle$
Case 1 $sfx_b = \langle\rangle$.
An immediate consequence of this, by [prf:DF:subsub:ante10,8,9] is that $slot_{b1} = slot_{b2}$ and $pfx_c = pfx_b$, so by Liebniz:

$$
\begin{array}{ll}
 & dif(\\
 & \quad dif(pfx_b \frown \langle slot_a\rangle \frown sfx_a, pfx_b \frown \langle slot_c\rangle), \\
 & \quad ssub(slot_{b2}, slot_c) \mathbin{⨾} \langle\rangle\ ) \\
= & \quad \text{“ [DF:pfx]:p21 ”} \\
 & dif(\ ssub(slot_a, slot_c) \mathbin{⨾} sfx_a, \\
 & \qquad \langle ssub(slot_{b2}, slot_c)\rangle\ ) \\
= & \quad \text{“ [DF:pfx]:p21 ”} \\
 & ssub(ssub(slot_a, slot_c), ssub(slot_{b2}, slot_c)) \mathbin{⨾} sfx_a \\
= & \quad \text{“ [SSub:subsub]:p17 ”} \\
 & ssub(slot_a, slot_{b2}) \mathbin{⨾} sfx_a
\end{array}
$$

Case 2 $sfx_b \neq \langle\rangle$
A consequence of this is that $slot_{b2} = last(sfx_b)$ and

$$pfx_b = pfx_c \frown \langle slot_{b1}\rangle \frown front(sfx_b)$$

so by Liebniz for pfx_b:

$$
\begin{array}{ll}
 & dif(dif(pfx_c \frown \langle slot_{b1}\rangle \frown front(sfx_b) \frown \langle slot_a\rangle \frown sfx_a, pfx_c \frown \langle slot_c\rangle\ ), \\
 & \qquad ssub(slot_{b1}, slot_c) \mathbin{⨾} sfx_b\ ) \\
= & \quad \text{“ [DF:def]:p21 ”} \\
 & dif(SSsub(slot_{b1}, slot_c) \mathbin{⨾} (front(sfx_b) \frown \langle slot_a\rangle \frown sfx_a), \\
 & \qquad ssub(slot_{b1}, slot_c) \mathbin{⨾} sfx_b\ ) \\
= & \quad \text{“ } \sigma \neq \langle\rangle \Rightarrow \sigma = front(\sigma) \frown \langle last(\sigma)\rangle \text{ ”} \\
 & dif(ssub(slot_{b1}, slot_c) \mathbin{⨾} (front(sfx_b) \frown \langle slot_a\rangle \frown sfx_a), \\
 & \qquad ssub(slot_{b1}, slot_c) \mathbin{⨾} (front(sfx_b) \frown \langle last(sfx_b)\rangle)\ ) \\
= & \quad \text{“ Liebniz for } last(sfx_b) \text{ ”} \\
 & dif(ssub(slot_{b1}, slot_c) \mathbin{⨾} (front(sfx_b) \frown \langle slot_a\rangle \frown sfx_a), \\
 & \qquad ssub(slot_{b1}, slot_c) \mathbin{⨾} (front(sfx_b) \frown \langle slot_{b2}\rangle)\ ) \\
= & \quad \text{“ [DF:def]:p21 ”} \\
 & ssub(slot_a, slot_{b2}) \mathbin{⨾} sfx_a
\end{array}
$$

□

A.2.15 Proof

of [CAT:DF:id]:p22

$$(ss \sharp tt) \fallingdotseq ss \quad = \quad tt$$

We use the shorthands of p91

$$
\begin{array}{ll}
 & (ss \sharp tt) \fallingdotseq ss \\
= & \quad \text{“ [CAT:def]:p21 ”} \\
 & (F(ss) \frown \langle L(ss) \sharp H(tt) \rangle \frown T(tt)) \fallingdotseq ss \\
= & \quad \text{“ [DF:def]:p21 ”} \\
 & \mathbf{let}\ \ rest = (F(ss) \frown \langle L(ss) \sharp H(tt) \rangle \frown T(tt)) - F(ss) \\
 & \mathbf{in}\ \ (H(rest) \setminus L(ss)) \fatsemi T(rest) \\
= & \quad \text{“ } (\sigma \frown \tau) - \sigma = \tau \text{, for sequences ”} \\
 & \mathbf{let}\ \ rest = \langle L(ss) \sharp H(tt) \rangle \frown T(tt) \\
 & \mathbf{in}\ \ (H(rest) \setminus L(ss)) \fatsemi T(rest) \\
= & \quad \text{“ substitute for } H(rest),\ T(rest) \text{ ”} \\
 & ((L(ss) \sharp H(tt)) \setminus L(ss)) \fatsemi T(tt) \\
= & \quad \text{“ [ssub:sadd]:p17 ”} \\
 & H(tt) \fatsemi T(tt) \\
= & \quad \text{“ Sequence law ”} \\
 & tt \\
\Box &
\end{array}
$$

A.2.16 Proof

of [CAT:DF:pfx]:p22

$$uu \preccurlyeq tt \Rightarrow (ss \sharp tt) \fallingdotseq (ss \sharp uu) \quad = \quad tt \fallingdotseq uu$$

We use the shorthands of p91.
Given the antecedent, we can write uu and tt as follows:

[prf:CAT:DF:pfx:uu]	$uu = pfx \frown \langle u \rangle$
[prf:CAT:DF:pfx:tt]	$tt = pfx \frown \langle t \rangle \frown sfx$
[prf:CAT:DF:pfx:u-le-t]	$u \preceq t$
[prf:CAT:DF:pfx:ss]	$ss = rr \fatsemi r$

Rhs:

$$\begin{aligned}
& tt \fallingdotseq uu \\
= & \quad \text{“ [prf:CAT:DF:pfx:uu],[prf:CAT:DF:pfx:tt] ”} \\
& (pfx \frown \langle t \rangle \frown sfx) \fallingdotseq (pfx \frown \langle u \rangle) \\
= & \quad \text{“ [DF:pfx]:p21 ”} \\
& (t \setminus u) \fatsemi sfx
\end{aligned}$$

Lhs:

$$\begin{aligned}
& (ss \sharp tt) \fallingdotseq (ss \sharp uu) \\
= & \quad \text{“ [prf:CAT:DF:pfx:uu],[prf:CAT:DF:pfx:tt],[prf:CAT:DF:pfx:ss] ”} \\
& ((rr \fatsemi r) \sharp (pfx \frown \langle t \rangle \frown sfx)) \fallingdotseq ((rr \fatsemi r) \sharp (pfx \frown \langle u \rangle)) \\
= & \quad \text{“ [CAT:def]:p21, twice ”} \\
& (rr \frown \langle r \sharp H(pfx) \rangle \frown T(pfx) \frown \langle t \rangle \frown sfx) \\
& \quad \fallingdotseq (rr \frown \langle r \sharp H(pfx) \rangle \frown T(pfx) \frown \langle u \rangle) \\
= & \quad \text{“ [DF:pfx]:p21, common prefix is } rr \frown \langle r \sharp H(pfx) \rangle \frown T(pfx) \text{ ”} \\
& (t \setminus u) \fatsemi sfx
\end{aligned}$$

□

A.3 Proofs for Healthiness Conditions

A.3.1 Proof

of [R1:idem]:p23

$$\mathbf{R1} \circ \mathbf{R1} = \mathbf{R1}$$

$$
\begin{array}{ll}
 & (\mathbf{R1} \circ \mathbf{R1})(P) \\
\equiv & \quad \text{“ [R1:def]:p23 ”} \\
 & \mathbf{R1}(P \land tr \preccurlyeq tr') \\
\equiv & \quad \text{“ [R1:def]:p23 ”} \\
 & (P \land tr \preccurlyeq tr') \land tr \preccurlyeq tr') \\
\equiv & \quad \text{“ prop. calc. ”} \\
 & P \land tr \preccurlyeq tr' \\
\equiv & \quad \text{“ [R1:def]:p23 ”} \\
 & \mathbf{R1}(P) \\
\square &
\end{array}
$$

A.3.2 Proof

of [Shift:obsolete]:p**??**

$$ss \frown Shift(s, slots, slots') \quad = \quad (ss \frown \langle s \rangle) \sharp\!\sharp\, (slots' \diagdown\!\diagdown\, slots))$$

Lhs:

$$
\begin{array}{ll}
 & ss \frown Shift(s, slots, slots') \\
= & \quad \text{“ [R2:shift]:p\textbf{??} ”} \\
 & ss \frown \langle s \sharp (slot' \diagdown slot) \rangle \frown sfx \\
 & \textbf{where } slot = last(slots), (slot' \fatsemi sfx) = slots' - front(slots) \\
= & \quad \text{“ intro } rest \text{ ”} \\
 & ss \frown \langle s \sharp (slot' \diagdown slot) \rangle \frown sfx \\
 & \textbf{where } slot = last(slots), (slot' \fatsemi sfx) = rest, rest = slots' - front(slots) \\
= & \quad \text{“ substitute, use p91 shorthand ”} \\
 & ss \frown \langle s \sharp (H(rest) \diagdown L(slots)) \rangle \frown T(rest) \\
 & \textbf{where } rest = slots' = F(slots)
\end{array}
$$

Rhs:

$$
\begin{array}{ll}
 & (ss \frown \langle s \rangle) \sharp\!\sharp\, (slots' \diagdown\!\diagdown\, slots)) \\
= & \quad \text{“ [CAT:def]:p21, using p91 shorthand ”} \\
 & F(ss \frown \langle s \rangle) \frown \langle L(ss \frown \langle s \rangle) \sharp H(slots' \diagdown\!\diagdown\, slots) \rangle \frown T(slots' \diagdown\!\diagdown\, slots) \\
= & \quad \text{“ } L(\sigma \frown \langle x \rangle) = x \text{ ”} \\
 & F(ss \frown \langle s \rangle) \frown \langle s \sharp H(slots' \diagdown\!\diagdown\, slots) \rangle \frown T(slots' \diagdown\!\diagdown\, slots) \\
= & \quad \text{“ [DF:def]:p21 ”} \\
 & F(ss \frown \langle s \rangle) \frown \langle s \sharp H(diff) \rangle \frown T(rest) \\
 & \textbf{where } diff = (H(rest) \diagdown L(slots)) \fatsemi T(rest) \\
 & \qquad\qquad rest = slots' - F(slots) \\
= & \quad \text{“ } H(x \fatsemi \sigma) = x \text{ ”} \\
 & F(ss \frown \langle s \rangle) \frown \langle s \sharp (H(rest) \diagdown L(slots)) \rangle \frown T(rest) \\
 & \textbf{where } rest = slots' - F(slots)
\end{array}
$$

If $ss = \langle\rangle$ then $F(ss \frown \langle s \rangle) = \langle\rangle$ and both Lhs and Rhs reduce to $\langle s \sharp (H(rest) \diagdown L(slots)) \rangle \frown T(rest)$. If $ss \neq \langle\rangle$, then $F(ss \frown \langle s \rangle) = ss$, and so Rhs becomes Lhs. $\square$

A.3.3 Proof

of [R2:subs-idem]:p**??**

$$R2_{ss}(R2_{tt}(P)) \equiv R2_{tt}(P)$$

We work with P explicit in *slots*, *slots*′:

$$
\begin{array}{ll}
 & R2_{ss}(R2_{tt}(P(slots, slots'))) \\
\equiv & \quad \text{“ [R2:subs]:p24 ”} \\
 & R2_{ss}(P(tt, tt \sharp (slots' \setminus\!\setminus slots))) \\
\equiv & \quad \text{“ [R2:subs]:p24 ”} \\
 & P(tt, tt \sharp ((ss \sharp (slots' \setminus\!\setminus slots)) \setminus\!\setminus ss)) \\
\equiv & \quad \text{“ [CAT:DF:id]:p22 ”} \\
 & P(tt, tt \sharp (slots' \setminus\!\setminus slots)) \\
\equiv & \quad \text{“ [R2:subs]:p24, backwards ”} \\
 & R2_{tt}(P(slots, slots')) \\
\Box &
\end{array}
$$

A.3.4 Proof

of [R2a:idem]:p**??**

$$\mathbf{R2a} \circ \mathbf{R2a} = \mathbf{R2a}$$

$$
\begin{array}{ll}
 & \mathbf{R2a}(\mathbf{R2a}(P)) \\
\equiv & \quad \text{“ [R2a:def]:p\textbf{??} ”} \\
 & \mathbf{R2a}(\exists\, tt \bullet R2_{tt}(P)) \\
\equiv & \quad \text{“ [R2a:def]:p\textbf{??} ”} \\
 & \exists\, ss \bullet R2_{ss}(\exists\, tt \bullet R2_{tt}(P)) \\
\equiv & \quad \text{“ } R2 \text{ distributes through } \exists\, tt \text{ ”} \\
 & \exists\, ss \bullet \exists\, tt \bullet R2_{ss}(R2_{tt}(P)) \\
\equiv & \quad \text{“ [R2:subs-idem]:p\textbf{??} ”} \\
 & \exists\, ss \bullet \exists\, tt \bullet R2_{tt}(P) \\
\equiv & \quad \text{“ } ss \text{ not free ”} \\
 & \exists\, tt \bullet R2_{tt}(P) \\
\equiv & \quad \text{“ [R2a:def]:p\textbf{??}, backwards ”} \\
 & \mathbf{R2a}(P) \\
\Box &
\end{array}
$$

A.3.5 Proof

of [R2a:almost]:p??

$$\mathbf{R2a}(slots' = slots) \equiv slots' \cong slots$$

$$
\begin{array}{ll}
 & \mathbf{R2a}(slots' = slots) \\
\equiv & \quad \text{“ [R2a:def]:p?? ”} \\
 & \exists\, tt \bullet R2_{tt}(slots' = slots) \\
\equiv & \quad \text{“ [R2:subs]:p24 ”} \\
 & \exists\, tt \bullet (tt \sharp (slots' \setminus\setminus slots) = tt)
\end{array}
$$

A.3.6 Proof

of [R2:idem]:p24

$$\mathbf{R2} \circ \mathbf{R2} = \mathbf{R2}$$

$$
\begin{array}{ll}
 & \mathbf{R2}(\mathbf{R2}(P)) \\
\equiv & \text{“ [R2:alt]:p24 ”} \\
 & \mathbf{R2}(\exists\, tt \bullet R2_{tt}(P) \land ER(tt, slots)) \\
\equiv & \text{“ [R2:alt]:p24 ”} \\
 & \exists\, ss \bullet R2_{ss}(\exists\, tt \bullet R2_{tt}(P) \land ER(tt, slots)) \land ER(ss, slots) \\
\equiv & \text{“ } R2_{ss} \text{ distributes through } \exists\, tt \text{ ”} \\
 & \exists\, ss \bullet (\exists\, tt \bullet R2_{ss}(R2_{tt}(P)) \land R2_{ss}(ER(tt, slots))) \land ER(ss, slots) \\
\equiv & \text{“ [R2:subs-idem]:p}\mathbf{??}\text{, [R2:subs]:p24 ”} \\
 & \exists\, ss \bullet (\exists\, tt \bullet R2_{tt}(P) \land ER(tt, ss)) \land ER(ss, slots) \\
\equiv & \text{“ expand } \exists\, tt \text{ scope, [EQRF:def]:p24 ”} \\
 & \exists\, ss, tt \bullet R2_{tt}(P) \land eqvref(tt) = eqvref(ss) \land eqvref(ss) = eqvref(slots) \\
\equiv & \text{“ properties of =, Liebniz ”} \\
 & \exists\, ss, tt \bullet R2_{tt}(P) \land eqvref(ss) = eqvref(slots) \land eqvref(tt) = eqvref(slots) \\
\equiv & \text{“ rearrange, shrink } \exists \text{ scopes ”} \\
 & (\,\exists\, tt \bullet R2_{tt}(P) \land eqvref(tt) = eqvref(slots)\,) \land \exists\, ss \bullet eqvref(ss) = eqvref(slots) \\
\equiv & \text{“ [R2:subs]:p24 backwards, [R2:alt]:p24 ”} \\
 & \mathbf{R2}(P) \land \exists\, ss \bullet ER(ss, slots) \\
\equiv & \text{“ witness } ss = slots \text{ ”} \\
 & \mathbf{R2}(P) \\
\Box &
\end{array}
$$

A.3.7 Proof

of [R2a:subsumes-arg]:p**??**

$$\mathbf{R2a}(P) \equiv \mathbf{R2a}(P) \lor P$$

$$
\begin{array}{ll}
 & \mathbf{R2a}(P) \\
\equiv & \quad \text{“ [R2a:def]:p??,[R2:subs]:p24 ”} \\
 & \exists\, ss \bullet P[ss, ss \sharp (slots' \setminus slots)/slots, slots'] \\
\equiv & \quad \text{“ } P \equiv (P \lhd c \rhd P) \text{ ”} \\
 & \exists\, ss \bullet P[ss, ss \sharp (slots' \setminus slots)/slots, slots'] \\
 & \qquad \lhd ss = slots \rhd P[ss, ss \sharp (slots' \setminus slots)/slots, slots'] \\
\equiv & \quad \text{“ defn. conditional ”} \\
 & \exists\, ss \bullet ss = slots \land P[ss, ss \sharp (slots' \setminus slots)/slots, slots'] \\
 & \qquad \lor ss \neq slots \land P[ss, ss \sharp (slots' \setminus slots)/slots, slots'] \\
\equiv & \quad \text{“ } \lor\text{-idem., } \exists\text{–}\lor \text{ distr. ”} \\
 & \quad (\exists\, ss \bullet ss = slots \land P[ss, ss \sharp (slots' \setminus slots)/slots, slots']) \\
 & \lor (\exists\, ss \bullet ss = slots \land P[ss, ss \sharp (slots' \setminus slots)/slots, slots'] \\
 & \lor ss \neq slots \land P[ss, ss \sharp (slots' \setminus slots)/slots, slots']) \\
\equiv & \quad \text{“ one-point rule, } ss\text{, defn. conditional backwards ”} \\
 & \quad P[slots, slots \sharp (slots' \setminus slots)/slots, slots'] \\
 & \lor (\exists\, ss \bullet P[ss, ss \sharp (slots' \setminus slots)/slots, slots'] \\
 & \qquad \lhd ss = slots \rhd P[ss, ss \sharp (slots' \setminus slots)/slots, slots']) \\
\equiv & \quad \text{“ [CAT:DF:id]:p22, } (P \lhd c \rhd P) \equiv P \text{ ”} \\
 & \quad P[slots, slots'/slots, slots'] \\
 & \lor \exists\, ss \bullet P[ss, ss \sharp (slots' \setminus slots)/slots, slots'] \\
\equiv & \quad \text{“ identity subs., [R2:subs]:p24 and [R2a:def]:p?? backwards ”} \\
 & P \lor \mathbf{R2a}(P)
\end{array}
$$

□

A.3.8 Proof

of [R2s:idem]:p**??**

$$\mathbf{R2s} \circ \mathbf{R2s} = \mathbf{R2s}$$

$(\mathbf{R2s} \circ \mathbf{R2s})(P(\mathit{pfx} \frown \langle \mathit{slot} \rangle, \mathit{pfx} \frown \langle \mathit{slot}' \rangle \frown \mathit{sfx})$

$\equiv$ “ [R2s:pfx]:p**??** ”

$\mathbf{R2s}(\sqcap_{ss} P(ss \frown \langle \mathit{slot} \rangle, ss \frown \langle \mathit{slot}' \rangle \frown \mathit{sfx}))$

$\equiv$ “ [R2s:pfx]:p**??**, using tt and noting ss above plays pfx role ”

$\sqcap_{tt}(\sqcap_{ss} P(tt \frown \langle \mathit{slot} \rangle, tt \frown \langle \mathit{slot}' \rangle \frown \mathit{sfx})))$

$\equiv$ “ $\sqcap_x P = P$, if x not free in P ”

$\mathbf{R2s}(\sqcap_{tt} P(tt \frown \langle \mathit{slot} \rangle, tt \frown \langle \mathit{slot}' \rangle \frown \mathit{sfx}))$

$\equiv$ “ [R2s:pfx]:p**??** backwards, using tt instead of ss ”

$\mathbf{R2s}(P)$

$\Box$

A.3.9 Proof

of [R2':idem]:p**??**

$$\mathbf{R2'} \circ \mathbf{R2'} = \mathbf{R2'}$$

$(\mathbf{R2'} \circ \mathbf{R2'})(P(pfx \frown \langle slot \rangle, pfx \frown \langle slot' \rangle \frown sfx)$

$\equiv$ " [R2':pfx]:p**??** "

$\mathbf{R2'}(P(\langle snull(\emptyset) \rangle, ssub(slot', slot) \mathbin{⨾} sfx))$

$\equiv$ " [R2':pfx]:p**??**, with $pfx = \langle \rangle$, $slot = snull(\emptyset)$ and $slot' = ssub(..)$ "

$P(\langle snull(\emptyset) \rangle, SSub(ssub(slot', slot), snull(\emptyset)) \mathbin{⨾} sfx))$

$\equiv$ " [SSub:nil]:p17 "

$P(\langle snull(\emptyset) \rangle, ssub(slot', slot) \mathbin{⨾} sfx)$

$\equiv$ " [R2':pfx]:p**??** backwards "

$\mathbf{R2'}(P)$

$\square$

A.3.10 Proof

of [R2:distr:and]:p24

$$\mathbf{R2}(P \wedge Q) = \mathbf{R2}(P) \wedge Q, \qquad \mathit{slots}, \mathit{slots}' \text{ not free in } Q$$

$$\begin{aligned}
& \mathbf{R2}(P \wedge Q) \\
\equiv \quad & \text{“ [R2:alt]:p24 ”} \\
& \exists\, ss \bullet R2_{ss}(P \wedge Q) \wedge ER(ss, \mathit{slots}) \\
\equiv \quad & \text{“ [R2:subs]:p24, } \mathit{slots}, \mathit{slots}' \text{ not free in } Q \text{ ”} \\
& \exists\, ss \bullet R2_{ss}(P) \wedge Q \wedge ER(ss, \mathit{slots}) \\
\equiv \quad & \text{“ narrow } \exists \text{ scope ”} \\
& (\exists\, ss \bullet R2_{ss}(P) \wedge ER(ss, \mathit{slots})) \wedge Q \\
\equiv \quad & \text{“ [R2:alt]:p24 backwards ”} \\
& \mathbf{R2}(P) \wedge Q
\end{aligned}$$

□

A.3.11 Proof

of [R2:distr:or]:p24

$$\mathbf{R2}(P \vee Q) = \mathbf{R2}(P) \vee \mathbf{R2}(Q)$$

$$\begin{aligned}
& \mathbf{R2}(P \vee Q) \\
\equiv \quad & \text{“ [R2:alt]:p24 ”} \\
& \exists\, ss \bullet R2_{ss}(P \vee Q) \wedge RE(ss, \mathit{slots}) \\
\equiv \quad & \text{“ [R2:subs]:p24 ”} \\
& \exists\, ss \bullet (R2_{ss}(P) \vee R2_{ss}(Q)) \wedge RE(ss, \mathit{slots}) \\
\equiv \quad & \text{“ } \wedge\text{–}\vee \text{ distr. ”} \\
& \exists\, ss \bullet R2_{ss}(P) \wedge RE(ss, \mathit{slots}) \vee R2_{ss}(Q) \wedge RE(ss, \mathit{slots}) \\
\equiv \quad & \text{“ } \exists\text{–}\vee \text{ distr. ”} \\
& (\exists\, ss \bullet R2_{ss}(P) \wedge RE(ss, \mathit{slots})) \vee (\exists\, ss \bullet R2_{ss}(Q) \wedge RE(ss, \mathit{slots})) \\
\equiv \quad & \text{“ [R2:alt]:p24 backwards ”} \\
& \mathbf{R2}(P) \vee \mathbf{R2}(Q)
\end{aligned}$$

□

A.3.12 Proof

of [R2:distr:cond]:p24

$$\mathbf{R2}(P \lhd c \rhd Q) \equiv \mathbf{R2}(P) \lhd c \rhd \mathbf{R2}(Q), \quad slots, slots' \text{ not free in } c$$

$\mathbf{R2}(P \lhd c \rhd Q)$

$\equiv$ “ defn. of $\lhd c \rhd$ ”

$\mathbf{R2}(c \land P \lor \neg\, c \land Q)$

$\equiv$ “ [R2:distr:or]:p24 ”

$\mathbf{R2}(c \land P) \lor \mathbf{R2}(\neg\, c \land Q)$

$\equiv$ “ [R2:distr:and]:p24, $slots, slots'$ not free in c ”

$c \land \mathbf{R2}(P) \lor \neg\, c \land \mathbf{R2}(Q)$

$\equiv$ “ defn. $\lhd c \rhd$ ”

$\mathbf{R2}(P) \lhd c \rhd \mathbf{R2}(Q)$

$\square$

A.3.13 Proof

of [R3:idem]:p24

$$\mathbf{R3} \circ \mathbf{R3} = \mathbf{R3}$$

$(\mathbf{R3} \circ \mathbf{R3})(P)$

$\equiv$ “ [R3:def]:p24 ”

$\mathbf{R3}(I\!I_R \lhd wait \rhd P)$

$\equiv$ “ [R3:def]:p24 ”

$I\!I_R \lhd wait \rhd (I\!I_R \lhd wait \rhd P)$

$\equiv$ “ $Q \lhd c \rhd \_$ is idempotent ”

$I\!I_R \lhd wait \rhd P$

$\equiv$ “ [R3:def]:p24, backwards ”

$\mathbf{R3}(P)$

$\Box$

A.3.14 Proof

of [R1:is:R2]:p26

$$\mathbf{R2}(\mathbf{R1}(\mathbf{true})) \equiv \mathbf{R1}(\mathbf{true})$$

$$
\begin{array}{ll}
 & \mathbf{R2}(\mathbf{R1}(\mathbf{true})) \\
\equiv & \quad \text{“ [R1:def]:p23, } \wedge\text{-unit ”} \\
 & \mathbf{R2}(slots \preccurlyeq slots') \\
\equiv & \quad \text{“ [R2:def]:p23 ”} \\
 & \exists\, ss \bullet (ss \preccurlyeq ss \sharp (slots' \setminus\!\!\setminus slots)) \wedge ER(ss, slots) \\
\equiv & \quad \text{“ [CAT:PFX]:p21, given } slots' \setminus\!\!\setminus slots \text{ is defined ”} \\
 & \exists\, ss \bullet slots \preccurlyeq slots' \wedge ER(ss, slots) \\
\equiv & \quad \text{“ witness } ss = \text{slots ”} \\
 & slots \preccurlyeq slots' \\
\equiv & \quad \text{“ [R1:def]:p23 backwards ”} \\
 & \mathbf{R1}(\mathbf{true}) \\
\square &
\end{array}
$$

A.3.15 Proof

of [R2a:almost]:p??

$$\mathbf{R2a}(slots = slots') \equiv slots \cong slots'$$

$$
\begin{aligned}
& \mathbf{R2a}(slots = slots') \\
\equiv\ & \quad \text{“ front and last of both must be equal ”} \\
& \mathbf{R2a}(front(slots) = front(slots') \land last(slots) = last(slots')) \\
\equiv\ & \quad \text{“ [R2a:def]:p?? ”} \\
& \exists\, ss, s \bullet \\
& \quad front(ss \frown \langle s \rangle) = front(ss \frown \langle sadd(s, ssub(slot', slot)) \rangle \frown sfx) \\
& \quad \land\ last(ss \frown \langle s \rangle) = last(ss \frown \langle sadd(s, ssub(slot', slot)) \rangle \frown sfx) \\
& \mathbf{where} \\
& \quad slot = last(slots) \\
& \quad (slot' \mathbin{⨾} sfx) = slots' - front(slots) \\
\equiv\ & \quad \text{“ From [Seq:FrontEQ:end]:p215, we get } sfx = \langle\rangle\text{, so\ldots ”} \\
& \exists\, ss, s \bullet \\
& \quad front(ss \frown \langle s \rangle) = front(ss \frown \langle sadd(s, ssub(slot', slot)) \rangle) \\
& \quad \land\ last(ss \frown \langle s \rangle) = last(ss \frown \langle sadd(s, ssub(slot', slot)) \rangle) \\
& \mathbf{where} \\
& \quad slot = last(slots) \\
& \quad slot' = head(slots' - front(slots)) \\
& \quad tail(slots' - front(slots)) = \langle\rangle \\
\equiv\ & \quad \text{“ [Seq:Front:def:alt]:p212,[Seq:Last:def:alt]:p213, simplify ”} \\
& \exists\, ss, s \bullet \\
& \quad s = sadd(s, ssub(slot', slot)) \\
& \mathbf{where} \ldots \\
\equiv\ & \quad \text{“ drop } ss\text{, expand } \mathbf{where} \text{ ”} \\
& (\exists\, s \bullet s = sadd(s, ssub(head(slots' - front(slots)), last(slots)))) \\
& \quad \land\ tail(slots' - front(slots)) = \langle\rangle \\
\equiv\ & \quad \text{“ [Seq:FrontEQ:end']:p215 ”} \\
& (\exists\, s \bullet s = sadd(s, ssub(head(slots' - front(slots)), last(slots)))) \\
& \quad \land\ front(slots') = front(slots) \\
\equiv\ & \quad \text{“ Liebniz ”} \\
& (\exists\, s \bullet s = sadd(s, ssub(head(slots' - front(slots')), last(slots)))) \\
& \quad \land\ front(slots') = front(slots) \\
\equiv\ & \quad \text{“ [Seq:Last:def:alt']:p213 ”} \\
& (\exists\, s \bullet s = sadd(s, ssub(last(slots'), last(slots)))) \\
& \quad \land\ front(slots') = front(slots) \\
\equiv\ & \quad \text{“ [sadd:unit]:p13 ”}
\end{aligned}
$$

$$
\begin{array}{ll}
& (\exists s \bullet ssub(last(slots'), last(slots)) = snull(sref(s)) \\
& \quad \land front(slots') = front(slots) \\
\equiv & \quad \text{“ [SSub:eqv]:p14 ”} \\
& (\exists s \bullet ssub(last(slots'), last(slots)) = snull(sref(s)) \\
& \quad \land front(slots') = front(slots) \\
\equiv & \quad \text{“ pair equality, [sref:def]:p11 ”} \\
& (\exists s \bullet \pi_1(ssub(last(slots'), last(slots))) = \pi_1(snull(sref(s))) \\
& \qquad \land sref(ssub(last(slots'), last(slots))) = sref(snull(sref(s)))) \\
& \quad \land front(slots') = front(slots) \\
\equiv & \quad \text{“ } \land\text{-idem., [SSub:ref]:p14,[SN:ref]:p12 ”} \\
& (\exists s \bullet \pi_1(ssub(last(slots'), last(slots))) = \pi_1(snull(sref(s))) \\
& \qquad \land sref(last(slots')) = sref(s) \\
& \qquad \land sref(ssub(last(slots'), last(slots))) = sref(snull(sref(s)))) \\
& \quad \land front(slots') = front(slots) \\
\equiv & \quad \text{“ Libeniz ”} \\
& (\exists s \bullet \pi_1(ssub(last(slots'), last(slots))) = \pi_1(snull(sref(last(slots')))) \\
& \qquad \land sref(ssub(last(slots'), last(slots))) = sref(snull(sref(last(slots'))))) \\
& \quad \land front(slots') = front(slots) \\
\equiv & \quad \text{“ [SN:ref]:p12, [SN:def]:p12 ”} \\
& (\exists s \bullet \pi_1(ssub(last(slots'), last(slots))) = \pi_1(hnull, sref(last(slots'))) \\
& \qquad \land sref(ssub(last(slots'), last(slots))) = sref(last(slots'))))) \\
& \quad \land front(slots') = front(slots) \\
\equiv & \quad \text{“ defn. of } \pi_1 \text{ ”} \\
& (\exists s \bullet \pi_1(ssub(last(slots'), last(slots))) = hnull \\
& \qquad \land sref(ssub(last(slots'), last(slots))) = sref(last(slots'))))) \\
& \quad \land front(slots') = front(slots) \\
\equiv & \quad \text{“ pair equality, ”} \\
& (\exists s \bullet ssub(last(slots'), last(slots)) = (hnull, sref(last(slots')))) \\
& \quad \land front(slots') = front(slots) \\
\equiv & \quad \text{“ drop } s\text{, [SN:def]:p12, backwards ”} \\
& ssub(last(slots'), last(slots)) = snull(last(slots')) \land front(slots') = front(slots) \\
\equiv & \quad \text{“ [SSub:eqv]:p14 ”} \\
& last(slots') \approx last(slots) \land front(slots') = front(slots) \\
\equiv & \quad \text{“ [SSEQV:expand]:p20, backwards ”} \\
& slots \cong slots' \\
\Box &
\end{array}
$$

A.3.16 Proof

of [SSEQV:is:R2a]

$$\text{[SSEQV:is:R2a]} \qquad \mathbf{R2a}(slots \cong slots') \equiv slots \cong slots'$$

$$
\begin{array}{ll}
 & \mathbf{R2a}(slots \cong slots') \\
\equiv & \quad \text{“ [SSEQV:expand]:p20 ”} \\
 & \mathbf{R2a}(front(slots) = front(slots') \land last(slots) \approx last(slots')) \\
\equiv & \quad \text{“ [R2a:def]:p\textbf{??} ”} \\
 & \exists\, ss, s \bullet \\
 & \quad front(ss \frown \langle s \rangle) = front(ss \frown \langle sadd(s, ssub(slot', slot)) \rangle \frown sfx) \\
 & \quad \land\ last(ss \frown \langle s \rangle) \approx last(ss \frown \langle sadd(s, ssub(slot', slot)) \rangle \frown sfx) \\
 & \mathbf{where} \\
 & \quad slot = last(slots) \\
 & \quad (slot' \mathbin{;} sfx) = slots' - front(slots) \\
\equiv & \quad \text{“ From [Seq:FrontEQ:end]:p215, we get } sfx = \langle\rangle \text{, so... ”} \\
 & \exists\, ss, s \bullet \\
 & \quad front(ss \frown \langle s \rangle) = front(ss \frown \langle sadd(s, ssub(slot', slot)) \rangle) \\
 & \quad \land\ last(ss \frown \langle s \rangle) \approx last(ss \frown \langle sadd(s, ssub(slot', slot)) \rangle) \\
 & \mathbf{where} \\
 & \quad slot = last(slots) \\
 & \quad slot' = head(slots' - front(slots)) \\
 & \quad tail(slots' - front(slots)) = \langle\rangle \\
\equiv & \quad \text{“ [Seq:Front:def:alt]:p212,[Seq:Last:def:alt]:p213, simplify ”} \\
 & \exists\, ss, s \bullet \\
 & \quad s \approx sadd(s, ssub(slot', slot)) \\
 & \mathbf{where} \ldots \\
\equiv & \quad \text{“ drop } ss \text{, expand } \mathbf{where} \text{ ”} \\
 & (\exists\, s \bullet s \approx sadd(s, ssub(head(slots' - front(slots)), last(slots)))) \\
 & \quad \land\ tail(slots' - front(slots)) = \langle\rangle \\
\equiv & \quad \text{“ [Seq:FrontEQ:end']:p215 ”} \\
 & (\exists\, s \bullet s \approx sadd(s, ssub(head(slots' - front(slots)), last(slots)))) \\
 & \quad \land\ front(slots') = front(slots) \\
\equiv & \quad \text{“ Liebniz ”} \\
 & (\exists\, s \bullet s \approx sadd(s, ssub(head(slots' - front(slots')), last(slots)))) \\
 & \quad \land\ front(slots') = front(slots) \\
\equiv & \quad \text{“ [Seq:Last:def:alt']:p213 ”} \\
 & (\exists\, s \bullet s \approx sadd(s, ssub(last(slots'), last(slots)))) \\
 & \quad \land\ front(slots') = front(slots) \\
\equiv & \quad \text{“ [sadd:eqv:unit]:p13 ”}
\end{array}
$$

$(\exists\, s \bullet \exists\, r \bullet ssub(last(slots'), last(slots)) = snull(r))$
$\quad \wedge\, front(slots') = front(slots)$

$\equiv$ “ [SSub:ref]:p14,[SN:ref]:p12 ”

$(\exists\, s \bullet \exists\, r \bullet ssub(last(slots'), last(slots)) = snull(sref(slot')))$
$\quad \wedge\, front(slots') = front(slots)$

$\equiv$ “ [SSub:eqv]:p14 ”

$(\exists\, s \bullet \exists\, r \bullet last(slots') \approx last(slots)) \wedge front(slots') = front(slots)$

$\equiv$ “ drop quantifiers ”

$front(slots) = front(slots') \wedge last(slots) \approx last(slots')$

$\equiv$ “ [SSEQV:expand]:p20, backwards ”

$slots \cong slots'$

$\square$

A.3.17 Proof

of [SSEQV:is:R2]

$$\text{[SSEQV:is:R2]} \qquad \mathbf{R2}(slots \cong slots') \equiv slots \cong slots'$$

REDO PROOF: new R2

$\mathbf{R2}(slots \cong slots')$

$\equiv$ “ [SSEQV:expand]:p20 ”

$\mathbf{R2}(front(slots) = front(slots') \land last(slots) \approx last(slots'))$

$\equiv$ “ [R2:def]:p23 ”

$\exists\, ss, s \bullet$

$front(ss \frown \langle s \rangle) = front(ss \frown \langle sadd(s, ssub(slot', slot)) \rangle \frown sfx)$

$\land\, last(ss \frown \langle s \rangle) \approx last(ss \frown \langle sadd(s, ssub(slot', slot)) \rangle \frown sfx)$

where

$slot = last(slots)$

$(slot' \,;\, sfx) = slots' - front(slots)$

$\equiv$ “ From [Seq:FrontEQ:end]:p215, we get $sfx = \langle\rangle$, so. . . ”

$\exists\, ss, s \bullet$

$front(ss \frown \langle s \rangle) = front(ss \frown \langle sadd(s, ssub(slot', slot)) \rangle)$

$\land\, last(ss \frown \langle s \rangle) \approx last(ss \frown \langle sadd(s, ssub(slot', slot)) \rangle)$

where

$slot = last(slots)$

$slot' = head(slots' - front(slots))$

$tail(slots' - front(slots)) = \langle\rangle$

$\equiv$ “ [Seq:Front:def:alt]:p212,[Seq:Last:def:alt]:p213, simplify ”

$\exists\, ss, s \bullet$

$s \approx sadd(s, ssub(slot', slot))$

where . . .

$\equiv$ “ drop ss, expand **where** ”

$(\exists\, s \bullet s \approx sadd(s, ssub(head(slots' - front(slots)), last(slots))))$

$\land\, tail(slots' - front(slots)) = \langle\rangle$

$\equiv$ “ [Seq:FrontEQ:end']:p215 ”

$(\exists\, s \bullet s \approx sadd(s, ssub(head(slots' - front(slots)), last(slots))))$

$\land\, front(slots') = front(slots)$

$\equiv$ “ Liebniz ”

$(\exists\, s \bullet s \approx sadd(s, ssub(head(slots' - front(slots')), last(slots))))$

$\land\, front(slots') = front(slots)$

$\equiv$ “ [Seq:Last:def:alt']:p213 ”

$(\exists\, s \bullet s \approx sadd(s, ssub(last(slots'), last(slots))))$

$\land\, front(slots') = front(slots)$

$\equiv$ “ [sadd:eqv:unit]:p13 ”

$(\exists s \bullet \exists r \bullet ssub(last(slots'), last(slots)) = snull(r))$
$\quad \land front(slots') = front(slots)$
$\equiv$ " [SSub:ref]:p14,[SN:ref]:p12 "
$(\exists s \bullet \exists r \bullet ssub(last(slots'), last(slots)) = snull(sref(slot')))$
$\quad \land front(slots') = front(slots)$
$\equiv$ " [SSub:eqv]:p14 "
$(\exists s \bullet \exists r \bullet last(slots') \approx last(slots)) \land front(slots') = front(slots)$
$\equiv$ " drop quantifiers "
$front(slots) = front(slots') \land last(slots) \approx last(slots')$
$\equiv$ " [SSEQV:expand]:p20, backwards "
$slots \cong slots'$

INCOMPLETE

A.3.18 Proof

of [SSEQ:is:R2]

$$\text{[SSEQ:is:R2]} \qquad \mathbf{R2}(slots = slots') \equiv slots = slots'$$

$\mathbf{R2}(slots = slots')$

$\equiv$ “ [R2:def]:p23 ”

$\exists\, ss \bullet ss = ss \sharp (slots' \setminus\!\setminus slots) \land ER(ss, slots)$

$\equiv$ “ [CAT:equal]:p21 ”

$\exists\, ss \bullet (slots' \setminus\!\setminus slots) = \langle snull(eqvref(ss))\rangle \land ER(ss, slots)$

$\equiv$ “ [DF:Null:equal]:p22 ”

$\exists\, ss \bullet (slots' \cong slots) \land eqvref(slots') = eqvref(ss) \land ER(ss, slots)$

$\equiv$ “ [EQRF:def]:p24 ”

$\exists\, ss \bullet slots' \cong slots$
$\quad \land\ eqvref(slots') = eqvref(ss) \land eqvref(ss) = eqvref(slots)$

$\equiv$ “ Liebniz ”

$\exists\, ss \bullet slots' \cong slots$
$\quad \land\ eqvref(slots') = eqvref(slots) \land eqvref(ss) = eqvref(slots)$

$\equiv$ “ [SSEQV:expand]:p20, [ER:def]:p18, re-arrange ”

$\exists\, ss \bullet eqvref(ss) = eqvref(slots)$
$\quad \land\ front(slots') = front(slots)$
$\quad \land\ last(slots') \approx last(slots) \land sref(last(slots')) = sref(last(slots))$

$\equiv$ “ [SEQV:equal-h]:p20 ”

$\exists\, ss \bullet eqvref(ss) = eqvref(slots)$
$\quad \land\ front(slots') = front(slots)$
$\quad \land\ first(last(slots')) = first(last(slots)) \land sref(last(slots')) = sref(last(slots))$

$\equiv$ “ structural equality, shrink quantifier scope ”

$slots' = slots \land \exists\, ss \bullet eqvref(ss) = eqvref(slots)$

$\equiv$ “ witness $ss = slots$ ”

$slots' = slots$

$\square$

A.3.19 Proof

of [IIr:is:R1]:p26

$$\mathbf{R1}(\mathit{I\!I}_R) \equiv \mathit{I\!I}_R$$

$$
\begin{array}{ll}
 & \mathbf{R1}(\mathit{I\!I}_R) \\
\equiv & \quad \text{“ [IIr:def]:p24 ”} \\
 & \mathbf{R1}(\neg\, ok \wedge slots \preccurlyeq slots' \\
 & \qquad \vee\, ok' \wedge wait' = wait \wedge slots' = slots) \\
\equiv & \quad \text{“ [R1:def]:p23 ”} \\
 & (\neg\, ok \wedge slots \preccurlyeq slots' \\
 & \ \vee\, ok' \wedge wait' = wait \wedge slots' = slots) \\
 & \ \wedge\, slots \preccurlyeq slots' \\
\equiv & \quad \text{“ $\wedge$–$\vee$ distr. ”} \\
 & \neg\, ok \wedge slots \preccurlyeq slots' \wedge slots \preccurlyeq slots' \\
 & \ \vee\, ok' \wedge wait' = wait \wedge slots' = slots \wedge slots \preccurlyeq slots' \\
\equiv & \quad \text{“ $\wedge$-idem. ”} \\
 & \neg\, ok \wedge slots \preccurlyeq slots' \\
 & \ \vee\, ok' \wedge wait' = wait \wedge slots' = slots \wedge slots \preccurlyeq slots' \\
\equiv & \quad \text{“ $s = s' \wedge s \preccurlyeq s' \equiv s = s'$, see [SSEQV:def]:p20 ”} \\
 & \neg\, ok \wedge slots \preccurlyeq slots' \\
 & \ \vee\, ok' \wedge wait' = wait \wedge slots' = slots \\
\equiv & \quad \text{“ [IIr:def]:p24, backwards ”} \\
 & \mathit{I\!I}_R \\
\Box &
\end{array}
$$

A.3.20 Proof

of [R3:is:R1]:p26

$$\mathbf{R1}(P) \equiv P \Rightarrow \mathbf{R1}(\mathbf{R3}(P)) \equiv \mathbf{R3}(P)$$

Assume

$$\mathbf{R1}(P) \equiv P \qquad \text{[R3:is:R1:hyp1]}$$

$\mathbf{R1}(\mathbf{R3}(P))$

$\equiv$ “ [R3:def]:p24 ”

$\mathbf{R1}(I\!I_R \lhd wait \rhd P)$

$\equiv$ “ [R1:distr:cond]:p23 ”

$\mathbf{R1}(I\!I_R) \lhd wait \rhd \mathbf{R1}(P)$

$\equiv$ “ [IIr:is:R1]:p26, [R3:is:R1:hyp1]:p134 ”

$I\!I_R \lhd wait \rhd P$

$\equiv$ “ [R3:def]:p24, backwards ”

$\mathbf{R3}(P)$

□

A.3.21 Proof

of [IIr:is:R2]:p26

$$\mathbf{R2}(\mathbb{I}_R) \equiv \mathbb{I}_R$$

$$
\begin{array}{ll}
 & \mathbf{R2}(\mathbb{I}_R) \\
\equiv & \text{“ [IIr:def]:p24 ”} \\
 & \mathbf{R2}(\neg\, ok \land slots \preccurlyeq slots' \\
 & \quad \lor ok' \land wait' = wait \land slots' = slots) \\
\equiv & \text{“ [R2:distr:or]:p24 ”} \\
 & \mathbf{R2}(\neg\, ok \land slots \preccurlyeq slots') \\
 & \quad \lor \mathbf{R2}(ok' \land wait' = wait \land slots' = slots) \\
\equiv & \text{“ [R2:distr:and]:p24, } slots, slots' \text{ not free in either } \neg\, ok \text{ or } ok' \text{ ”} \\
 & \neg\, ok \land \mathbf{R2}(slots \preccurlyeq slots') \\
 & \quad \lor ok' \land wait' = wait \land \mathbf{R2}(slots' = slots) \\
\equiv & \text{“ [SSEQ:is:R2]:p132 ”} \\
 & \neg\, ok \land slots \preccurlyeq slots' \\
 & \quad \lor ok' \land wait' = wait \land slots' = slots \\
\equiv & \text{“ [IIr:def]:p24, backwards ”} \\
 & \mathbb{I}_R \\
\Box &
\end{array}
$$

A.3.22 Proof

of [IIr:is:R3]:p26

$$\mathbf{R3}(I\!I_R) \equiv I\!I_R$$

$\mathbf{R3}(I\!I_R)$

$\equiv$ “ [R3:def]:p24 ”

$I\!I_R \lhd wait \rhd I\!I_R$

$\equiv$ “ $P \lhd c \rhd P \equiv P$ ”

$I\!I_R$

□

A.3.23 Proof

of [R1:R2:comm]:p26

$$\mathbf{R1} \circ \mathbf{R2} = \mathbf{R2} \circ \mathbf{R1}$$

Lhs:

$$
\begin{array}{ll}
 & \mathbf{R1}(\mathbf{R2}(P)) \\
\equiv & \quad \text{“ [R2:alt]:p24 ”} \\
 & \mathbf{R1}(\exists\, ss \bullet P[ss, ss \sharp (slots' \setminus\!\!\setminus slots)/slots, slots'] \land ER(ss, slots)) \\
\equiv & \quad \text{“ [R1:def]:p23 ”} \\
 & (\exists\, ss \bullet P[ss, ss \sharp (slots' \setminus\!\!\setminus slots)/slots, slots'] \land ER(ss, slots)) \land slots \preccurlyeq slots' \\
\equiv & \quad \text{“ expand quantifier scope ”} \\
 & \exists\, ss \bullet P[ss, ss \sharp (slots' \setminus\!\!\setminus slots)/slots, slots'] \land ER(ss, slots)) \land slots \preccurlyeq slots'
\end{array}
$$

Rhs:

$$
\begin{array}{ll}
 & \mathbf{R2}(\mathbf{R1}(P)) \\
\equiv & \quad \text{“ [R1:def]:p23 ”} \\
 & \mathbf{R2}(P \land slots \preccurlyeq slots') \\
\equiv & \quad \text{“ [R2:alt]:p24 ”} \\
 & \exists\, ss \bullet P[ss, ss \sharp (slots' \setminus\!\!\setminus slots)/slots, slots'] \land ss \preccurlyeq ss \sharp (slots' \setminus\!\!\setminus slots) \land ER(ss, slots) \\
\equiv & \quad \text{“ } ss \preccurlyeq tt \text{ provided } tt \text{ is defined, i.e here that } slots \preccurlyeq slots' \text{ ”} \\
 & \exists\, ss \bullet P[ss, ss \sharp (slots' \setminus\!\!\setminus slots)/slots, slots'] \land slots \preccurlyeq slots' \land ER(ss, slots)
\end{array}
$$

Both sides are identical □

A.3.24 Proof

of [R1:R3:comm]:p26

$$\mathbf{R1} \circ \mathbf{R3} = \mathbf{R3} \circ \mathbf{R1}$$

Lhs:

$$
\begin{array}{ll}
 & \mathbf{R1}(\mathbf{R3}(P)) \\
\equiv & \quad \text{“ [R3:def]:p24 ”} \\
 & \mathbf{R1}(I\!I_R \lhd wait \rhd P) \\
\equiv & \quad \text{“ [R1:distr:cond]:p23 ”} \\
 & \mathbf{R1}(I\!I_R) \lhd wait \rhd \mathbf{R1}(P) \\
\equiv & \quad \text{“ [IIr:is:R1]:p26 ”} \\
 & I\!I_R \lhd wait \rhd \mathbf{R1}(P) \\
\equiv & \quad \text{“ [R3:def]:p24, backwards ”} \\
 & \mathbf{R3}(\mathbf{R1})(P)) \\
\Box &
\end{array}
$$

A.3.25 Proof

of [R2:R3:comm]:p26

$$\mathbf{R2} \circ \mathbf{R3} = \mathbf{R3} \circ \mathbf{R2}$$

$$
\begin{array}{ll}
 & \mathbf{R2}(\mathbf{R3}(P) \\
\equiv & \quad \text{“ [R3:def]:p24 ”} \\
 & \mathbf{R2}(\mathit{I\!I}_R \lhd wait \rhd P) \\
\equiv & \quad \text{“ [R2:distr:cond]:p24 ”} \\
 & \mathbf{R2}(\mathit{I\!I}_R) \lhd wait \rhd \mathbf{R2}(P) \\
\equiv & \quad \text{“ [IIr:is:R2]:p26 ”} \\
 & \mathit{I\!I}_R \lhd wait \rhd \mathbf{R2}(P) \\
\equiv & \quad \text{“ [R3:def]:p24, backwards ”} \\
 & \mathbf{R3}(\mathbf{R2}(P))
\end{array}
$$

A.3.26 Proof

of [R:idem]:p26

$$\mathbf{R} \circ \mathbf{R} = \mathbf{R}$$

$$
\begin{array}{ll}
 & \mathbf{R} \circ \mathbf{R} \\
= & \quad \text{“ [R:def]:p25 ”} \\
 & \mathbf{R} \circ \mathbf{R3} \circ \mathbf{R2} \circ \mathbf{R1} \\
= & \quad \text{“ [R:def]:p25 ”} \\
 & \mathbf{R3} \circ \mathbf{R2} \circ \mathbf{R1} \circ \mathbf{R3} \circ \mathbf{R2} \circ \mathbf{R1} \\
= & \quad \text{“ [R1:R3:comm]:p26,[R2:R3:comm]:p26,[R1:R2:comm]:p26 ”} \\
 & \mathbf{R3} \circ \mathbf{R3} \circ \mathbf{R2} \circ \mathbf{R2} \circ \mathbf{R1} \circ \mathbf{R1} \\
= & \quad \text{“ [R1:idem]:p23,[R2:idem]:p24,[R3:idem]:p24 ”} \\
 & \mathbf{R3} \circ \mathbf{R2} \circ \mathbf{R1} \\
= & \quad \text{“ [R:def]:p25, backwards ”} \\
 & \mathbf{R}
\end{array}
$$

A.3.27 Proof

of [P-GROW:Q-GROW:eq:PQ-GROW].

The following law (referred to in [She06] as "relational calculus"), is generally quite useful:

[P-GROW:Q-GROW:eq:PQ-GROW] $(P \land GROW);\ (Q \land GROW)$
$\equiv ((P \land GROW);\ (Q \land GROW)) \land GROW$

Proof, reduce Lhs to Rhs:

$(P \land GROW);\ (Q \land GROW)$

$\equiv$ " [Seq:def]:p32 "

$\exists\, obs_m \bullet P[seq'] \land GROW[seq'] \land Q[seq] \land GROW[seq]$

$\equiv$ " [pfx:trans]:p17 "

$\exists\, obs_m \bullet P[seq'] \land GROW[seq'] \land Q[seq] \land GROW[seq] \land GROW$

$\equiv$ " shrink scope "

$(\exists\, obs_m \bullet P[seq'] \land GROW[seq'] \land Q[seq] \land GROW[seq]) \land GROW$

$\equiv$ " [Seq:def]:p32, backwards "

$(P \land GROW;\ Q \land GROW) \land GROW$

$\square$

This can also be written as $\mathbf{R1}(P);\ \mathbf{R1}(Q) \equiv \mathbf{R1}(\mathbf{R1}(P);\ \mathbf{R1}(Q))$

A.3.28 Proof

of [expand-R:eq:expand]:p26

$$P \equiv \mathbf{R}(P) \quad \Rightarrow \quad GROW;\ P = GROW$$

We assume the antecedent and proceed:

$$
\begin{array}{ll}
& GROW;\ P \\
\equiv & \quad \text{“ assumption, [R:expand:1]:p26, ignoring } \mathbf{R2} \text{ part ”} \\
& GROW;\ (I\!I_R \lhd wait \rhd P \land GROW) \\
\equiv & \quad \text{“ [Cond:def]:p32 ”} \\
& GROW;\ (wait \land I\!I_R \lor \neg\, wait \land P \land GROW) \\
\equiv & \quad \text{“ ; –}\lor\text{ distributivity ”} \\
& (GROW;\ wait \land I\!I_R) \lor (GROW;\ \neg\, wait \land P \land GROW) \\
\equiv & \quad \text{“ [IIr:def]:p24 ”} \\
& \quad (GROW;\ wait \land (DIV \lor ok' \land RSTET)) \lor (GROW;\ \neg\, wait \land P \land GROW) \\
\equiv & \quad \text{“ ; –}\lor\text{ distributivity ”} \\
& \quad (GROW;\ wait \land DIV) \lor (GROW;\ wait \land ok' \land RSTET) \\
& \lor (GROW;\ \neg\, wait \land P \land GROW) \\
\equiv & \quad \text{“ [DIV:def]:p24 ”} \\
& \quad (GROW;\ wait \land \neg\, ok \land GROW) \\
& \lor (GROW;\ wait \land ok' \land RSTET) \\
& \lor (GROW;\ \neg\, wait \land P \land GROW) \\
\equiv & \quad \text{“ Line 1:[Seq:def]:p32; Line 2: [comp:RSTET]:p25; Line 3:[comp:GROW:closed]:p147 ”} \\
& \quad (\exists\, obs_0 \bullet GROW[obs_0/obs'] \land wait_0 \land \neg\, ok_0 \land GROW[obs_0, obs]) \\
& \lor (\exists\, ok_0, state_0 \bullet GROW[ok_0, state_0/ok', state'] \land (wait \land ok')[ok_0, state_0, rest'/ok, state, rest]) \\
& \lor GROW \land (GROW;\ \neg\, wait \land P \land GROW) \\
\equiv & \quad \text{“ Line 1: factor } slots_0 \text{ apart from } ok_0 \text{ and } wait_0\text{; Line 2: subst., drop } \exists\, state_0 \text{ ”} \\
& \quad (\exists\, slots_0 \bullet GROW[slots_0/slots'] \land GROW[slots_0, slots]) \land (\exists\, wait_0, ok_0 \bullet wait_0 \land \neg\, ok_0) \\
& \lor (\exists\, ok_0 \bullet GROW \land wait' \land ok') \lor GROW \land (GROW;\ \neg\, wait \land P \land GROW) \\
\equiv & \quad \text{“ Line 1, } wait_0 = true \land ok_0 = false\text{; Line 2: } ok_0 \text{ not mentioned. ”} \\
& \quad (\exists\, slots_0 \bullet GROW[slots_0/slots'] \land GROW[slots_0, slots]) \\
& \lor GROW \land wait' \land ok' \lor GROW \land (GROW;\ \neg\, wait \land P \land GROW) \\
\equiv & \quad \text{“ [Seq:def]:p32, backwards, noting only } slots \text{ mentioned, distributivity ”} \\
& (GROW;\ GROW) \lor GROW \land (wait' \land ok' \lor (GROW;\ \neg\, wait \land P \land GROW)) \\
\equiv & \quad \text{“ [GROW-GROW:eq:GROW]:p23 ”} \\
& GROW \lor GROW \land \ldots \\
\equiv & \quad \text{“ absorption ”} \\
& GROW
\end{array}
$$

□

A.3.29 Proof

of [DIV-R:eq:DIV]:p26

$$P \equiv \mathbf{R}(P) \quad \Rightarrow \quad DIV;\ P = DIV$$

We assume the antecedent and proceed:

$$
\begin{array}{ll}
 & DIV;\ P \\
\equiv & \text{" assumption, [R:expand:1]:p26, ignoring } \mathbf{R2} \text{ part "} \\
 & DIV;\ (I\!I_R \lhd wait \rhd P \land GROW) \\
\equiv & \text{" [Cond:def]:p32 "} \\
 & DIV;\ (wait \land I\!I_R \lor \neg\, wait \land P \land GROW) \\
\equiv & \text{" ; -}\lor\text{ distributivity "} \\
 & (DIV;\ wait \land I\!I_R) \lor (DIV;\ \neg\, wait \land P \land GROW) \\
\equiv & \text{" [IIr:def]:p24 "} \\
 & \quad (DIV;\ wait \land DIV \lor ok' \land RSTET) \lor (DIV;\ \neg\, wait \land P \land GROW) \\
\equiv & \text{" ; -}\lor\text{ distributivity "} \\
 & \quad (DIV;\ wait \land DIV) \lor (DIV;\ wait \land ok' \land RSTET) \\
 & \lor (DIV;\ \neg\, wait \land P \land GROW) \\
\equiv & \text{" [DIV:def]:p24 "} \\
 & \quad (\neg\, ok \land GROW;\ wait \land \neg\, ok \land GROW) \\
 & \lor (\neg\, ok \land GROW;\ wait \land ok' \land RSTET) \\
 & \lor (\neg\, ok \land GROW;\ \neg\, wait \land P \land GROW) \\
\equiv & \text{" Line 1:[Seq:def]:p32; Line 2: [comp:RSTET]:p25; Line 3:[comp:GROW:closed]:p147 "} \\
 & \quad (\exists\, obs_0 \bullet \neg\, ok \land GROW[obs_0/obs'] \land wait_0 \land \neg\, ok_0 \land GROW[obs_0, obs]) \\
 & \lor (\exists\, ok_0, state_0 \bullet \neg\, ok \land GROW[ok_0, state_0/ok', state'] \land (wait \land ok')[ok_0, state_0, rest'/ok, state, rest]) \\
 & \lor GROW \land (\neg\, ok \land GROW;\ \neg\, wait \land P \land GROW) \\
\equiv & \text{" Line 1: factor } slots_0 \text{ apart from } ok_0 \text{ and } wait_0\text{; Line 2: subst., drop } \exists\, state_0 \text{ "} \\
 & \quad (\exists\, slots_0 \bullet \neg\, ok \land GROW[slots_0/slots'] \land GROW[slots_0, slots]) \land (\exists\, wait_0, ok_0 \bullet wait_0 \land \neg\, ok_0) \\
 & \lor (\exists\, ok_0 \bullet \neg\, ok \land GROW \land wait' \land ok') \lor GROW \land (\neg\, ok \land GROW;\ \neg\, wait \land P \land GROW) \\
\equiv & \text{" Line 1, reduce } slots_0 \text{ scope, } wait_0 = true \land ok_0 = false\text{; Line 2: } ok_0 \text{ not mentioned. "} \\
 & \quad \neg\, ok \land (\exists\, slots_0 \bullet GROW[slots_0/slots'] \land GROW[slots_0, slots]) \\
 & \lor \neg\, ok \land GROW \land wait' \land ok' \lor \neg\, ok \land GROW \land (GROW;\ \neg\, wait \land P \land GROW) \\
\equiv & \text{" [Seq:def]:p32, backwards, noting only } slots \text{ mentioned, distributivity "} \\
 & \neg\, ok \land (GROW;\ GROW) \lor \neg\, ok \land GROW \land (wait' \land ok' \lor (GROW;\ \neg\, wait \land P \land GROW)) \\
\equiv & \text{" [GROW-GROW:eq:GROW]:p23 "} \\
 & \neg\, ok \land GROW \lor \neg\, ok \land GROW \land \ldots \\
\equiv & \text{" absorption, [DIV:def]:p24 backwards "} \\
 & DIV
\end{array}
$$

□

A.3.30 Proof

of [IIr:is:CSP1]:p27

$$\mathbf{CSP1}(I\!I_R) \quad = \quad I\!I_R$$

$$
\begin{array}{ll}
 & \mathbf{CSP1}(I\!I_R) \\
\equiv & \quad \text{“ [CSP1:def]:p27 ”} \\
 & I\!I_R \lor \neg\, ok \land slots \preccurlyeq slots' \\
\equiv & \quad \text{“ [IIr:def]:p24 ”} \\
 & (DIV \lor ok' \land RSTET) \lor DIV \\
\equiv & \quad \text{“ logic ”} \\
 & DIV \lor ok' \land RSTET \\
\equiv & \quad \text{“ [IIr:def]:p24, backwards ”} \\
 & I\!I_R \\
\Box &
\end{array}
$$

A.3.31 Proof

of [R1:CSP1:comm]:p27

$$\mathbf{R1} \circ \mathbf{CSP1} \quad = \quad \mathbf{CSP1} \circ \mathbf{R1}$$

$\mathbf{R1}(\mathbf{CSP1}(P))$
$\equiv$ “ [CSP1:def]:p27 ”
$\mathbf{R1}(P \lor \neg\, ok \land slots \preccurlyeq slots')$
$\equiv$ “ [R1:def]:p23 ”
$(P \lor \neg\, ok \land slots \preccurlyeq slots') \land slots \preccurlyeq slots'$
$\equiv$ “ prop. calc. ”
$P \land slots \preccurlyeq slots' \lor \neg\, ok \land slots \preccurlyeq slots'$
$\equiv$ “ [R1:def]:p23, backwards ”
$\mathbf{R1}(P) \lor \neg\, ok \land slots \preccurlyeq slots'$
$\equiv$ “ [CSP1:def]:p27, backwards ”
$\mathbf{CSP1}(\mathbf{R1}(P))$
$\Box$

A.3.32 Proof

of [R2:CSP1:comm]:p27

$$\mathbf{R2} \circ \mathbf{CSP1} \quad = \quad \mathbf{CSP1} \circ \mathbf{R2}$$

$\mathbf{R2}(\mathbf{CSP1}(P))$
$\equiv$ “ [CSP1:def]:p27 ”
$\mathbf{R2}(P \lor \neg\, ok \land slots \preccurlyeq slots')$
$\equiv$ “ [R2:distr:or]:p24 ”
$\mathbf{R2}(P) \lor \mathbf{R2}(\neg\, ok \land slots \preccurlyeq slots')$
$\equiv$ “ [R2:distr:and]:p24 ”
$\mathbf{R2}(P) \lor \neg\, ok \land \mathbf{R2}(slots \preccurlyeq slots')$
$\equiv$ “ [R1:is:R2]:p26 ”
$\mathbf{R2}(P) \lor \neg\, ok \land slots \preccurlyeq slots'$
$\equiv$ “ [CSP1:def]:p27, backwards ”
$\mathbf{CSP1}(\mathbf{R2}(P))$

A.3.33 Proof

of [ok:and:IIr]

[ok:and:II] $ok \land I\!I_R \equiv ok \land ok' \land RSTET$

This is [She06, Lemma 3.1, p36].

$$
\begin{array}{ll}
 & ok \land I\!I_R \\
\equiv & \text{“ [IIr:def]:p24 ”} \\
 & ok \land (DIV \lor ok' \land RSTET) \\
\equiv & \text{“ distr. ”} \\
 & ok \land DIV \lor ok \land ok' \land RSTET \\
\equiv & \text{“ [DIV:def]:p24 ”} \\
 & ok \land \neg\, ok \land GROW \lor ok \land ok' \land RSTET \\
\equiv & \text{“ logic ”} ok \land ok' \land RSTET \\
\Box &
\end{array}
$$

A.3.34 Proof

of [not-ok:and:IIr]

[not-ok:and:IIr] $\neg\, ok \land I\!I_R \equiv DIV$

$$
\begin{array}{ll}
 & \neg\, ok \land I\!I_R \\
\equiv & \text{“ [IIr:def]:p24 ”} \\
 & \neg\, ok \land (DIV \lor ok' \land RSTET) \\
\equiv & \text{“ [DIV:def]:p24 ”} \\
 & \neg\, ok \land (\neg\, ok \land GROW \lor\land ok' \land RSTET) \\
\equiv & \text{“ [RSTET:def]:p24 ”} \\
 & \neg\, ok \land (\neg\, ok \land GROW \land ok' \land wait' = wait \land slots' = slots) \\
\equiv & \text{“ } slots' = slots \Rightarrow GROW \text{ ”} \\
 & \neg\, ok \land (\neg\, ok \land GROW \land ok' \land wait' = wait \land slots' = slots \land GROW) \\
\equiv & \text{“ distr., idem. ”} \\
 & \neg\, ok \land GROW \lor \neg\, ok \land GROW \land ok' \land wait' = wait \land slots' = slots \\
\equiv & \text{“ absorb. ”} \\
 & \neg\, ok \land GROW \\
\equiv & \text{“ [DIV:def]:p24, backwards ”} \\
 & DIV \\
\Box &
\end{array}
$$

A.3.35 Proof

of [R3:CSP1:comm]:p27

$$\mathbf{CSP1}(\mathbf{R3}(P)) \quad \equiv \quad \mathbf{R3}(\mathbf{CSP1}(P)) \lor wait \land \mathbf{CSP1}(\mathbf{false})$$

$\mathbf{CSP1}(\mathbf{R3}(P))$

$\equiv$ “ [R3:def]:p24 ”

$\mathbf{CSP1}(I\!I_R \lhd wait \rhd P)$

$\equiv$ “ defn conditional ”

$\mathbf{CSP1}(wait \land I\!I_R \lor \neg\, wait \land P)$

$\equiv$ “ [CSP1:alt]:p27 ”

$wait \land I\!I_R \lor \neg\, wait \land P \lor \mathbf{CSP1}(\mathbf{false})$

$\equiv$ “ prop. calc ”

$wait \land I\!I_R \lor \neg\, wait \land P \lor wait \land \mathbf{CSP1}(\mathbf{false}) \lor \neg\, wait \land \mathbf{CSP1}(\mathbf{false})$

$\equiv$ “ rearrange ”

$wait \land I\!I_R \lor wait \land \mathbf{CSP1}(\mathbf{false}) \lor \neg\, wait \land P \lor \neg\, wait \land \mathbf{CSP1}(\mathbf{false})$

$\equiv$ “ $\land$–$\lor$ distributivity ”

$wait \land (I\!I_R \lor \mathbf{CSP1}(\mathbf{false})) \lor \neg\, wait \land (P \lor \mathbf{CSP1}(\mathbf{false}))$

$\equiv$ “ defn. conditional ”

$(I\!I_R \lor \mathbf{CSP1}(\mathbf{false}) \lhd wait \rhd P \lor \mathbf{CSP1}(\mathbf{false}))$

$\equiv$ “ [CSP1:alt]:p27, backwards ”

$\mathbf{CSP1}(I\!I_R) \lhd wait \rhd \mathbf{CSP1}(P)$

$\equiv$ “ [IIr:is:CSP1]:p27 ”

$I\!I_R \lhd wait \rhd \mathbf{CSP1}(P)$

$\equiv$ “ [R3:def]:p24, backwards ”

$\mathbf{R3}(\mathbf{CSP1}(P))$

$\square$

A.3.36 Proof

of [comp:R1:closed]:p23

$$(P \equiv \mathbf{R1}(P)) \land (Q \equiv \mathbf{R1}(Q)) \Rightarrow ((P;\ Q) \equiv \mathbf{R1}(P;\ Q))$$

We assume

[comp:R1:closed:hyp1]	$P \equiv \mathbf{R1}(P)$
[comp:R1:closed:hyp2]	$Q \equiv \mathbf{R1}(Q)$

to show:

$P;\ Q$

$\equiv$ " [comp:R1:closed:hyp1]:p147,[comp:R1:closed:hyp2]:p147 "

$\mathbf{R1}(P);\ \mathbf{R1}(Q)$

$\equiv$ " [R1:def]:p23 "

$(P \land slots \preccurlyeq slots');\ (Q \land slots \preccurlyeq slots')$

$\equiv$ " [Seq:def]:p32 "

$\exists\, obs_0 \bullet P[obs_0/obs'] \land slots \preccurlyeq slots_0 \land Q[obs_0/obs] \land slots_0 \preccurlyeq slots'$

$\equiv$ " [EX:trans]:p19 "

$\exists\, obs_0 \bullet P[obs_0/obs'] \land slots \preccurlyeq slots_0 \land Q[obs_0/obs] \land slots_0 \preccurlyeq slots' \land slots \preccurlyeq slots'$

$\equiv$ " obs_0 not free in last conjunct "

$(\exists\, obs_0 \bullet P[obs_0/obs'] \land slots \preccurlyeq slots_0 \land Q[obs_0/obs] \land slots_0 \preccurlyeq slots') \land slots \preccurlyeq slots'$

$\equiv$ " [Seq:def]:p32, backwards "

$((P \land slots \preccurlyeq slots');\ (Q \land slots \preccurlyeq slots')) \land slots \preccurlyeq slots'$

$\equiv$ " [R1:def]:p23, backwards thrice "

$\mathbf{R1}(\mathbf{R1}(P);\ \mathbf{R1}(Q))$

$\equiv$ " [comp:R1:closed:hyp1]:p147,[comp:R1:closed:hyp2]:p147 "

$\mathbf{R1}(P;\ Q)$

$\Box$

An important result also from the above proof is [comp:GROW:closed]:

$$(P \land GROW);\ (Q \land GROW) \quad \equiv \quad GROW \land ((P \land GROW);\ (Q \land GROW))$$

A.3.37 Proof

of [comp:R2a:closed]:p**??**

$$(P \equiv \mathbf{R2a}(P)) \land (Q \equiv \mathbf{R2a}(Q)) \Rightarrow ((P;\ Q) \equiv \mathbf{R2a}(P;\ Q))$$

We assume the antecedent, and proceed to start with the rhs of the consequent.
Note that as the antecedent is used in the first and last steps only, that we have also shown that

[R2a:comp:closure] $\mathbf{R2a}(\mathbf{R2a}(P);\ \mathbf{R2a}(Q)) \equiv \mathbf{R2a}(P);\ \mathbf{R2a}(Q)$

for arbitrary P and Q
(see overleaf)

$\mathbf{R2a}(P;\ Q)$

$\equiv$ “ P and Q are **R2a**-healthy by assumption ”

$\mathbf{R2a}(\mathbf{R2a}(P);\ \mathbf{R2a}(Q))$

$\equiv$ “ [R2a:def]:p**??** ”

$\mathbf{R2a}((\exists\, uu \bullet P[uu, uu \sharp (slots' \setminus\setminus slots)/slots', slots])\ ;$
$(\exists\, vv \bullet Q[vv, vv \sharp (slots' \setminus\setminus slots)/slots', slots])\ )$

$\equiv$ “ [Seq:def]:p32 ”

$\mathbf{R2a}(\exists\, slots_0 \bullet$
$(\exists\, uu \bullet P[uu, uu \sharp (slots_0 \setminus\setminus slots)/slots', slots]) \land$
$(\exists\, vv \bullet Q[vv, vv \sharp (slots' \setminus\setminus slots_0)/slots', slots])\ )$

$\equiv$ “ [R2a:def]:p**??** ”

$\exists\, ss, slots_0 \bullet$
$(\exists\, uu \bullet P[uu, uu \sharp (slots_0 \setminus\setminus ss)/slots', slots]) \land$
$(\exists\, vv \bullet Q[vv, vv \sharp ((ss \sharp (slots' \setminus\setminus slots)) \setminus\setminus slots_0)/slots', slots])\ )$

$\equiv$ “ take $slots_0 = ss \sharp (slots_1 \setminus\setminus slots)$ ”

$\exists\, ss, slots_1 \bullet$
$(\exists\, uu \bullet P[uu, uu \sharp ((ss \sharp (slots_1 \setminus\setminus slots)) \setminus\setminus ss)/slots', slots]) \land$
$(\exists\, vv \bullet Q[vv, vv \sharp ((ss \sharp (slots' \setminus\setminus slots)) \setminus\setminus (ss \sharp (slots_1 \setminus\setminus slots)))/slots', slots])\ )$

$\equiv$ “ [CAT:DF:id]:p22 ”

$\exists\, ss, slots_1 \bullet$
$(\exists\, uu \bullet P[uu, uu \sharp (slots_1 \setminus\setminus slots)/slots', slots]) \land$
$(\exists\, vv \bullet Q[vv, vv \sharp ((ss \sharp (slots' \setminus\setminus slots)) \setminus\setminus (ss \sharp (slots_1 \setminus\setminus slots)))/slots', slots])\ )$

$\equiv$ “ [CAT:DF:pfx]:p22 ”

$\exists\, ss, slots_1 \bullet$
$(\exists\, uu \bullet P[uu, uu \sharp (slots_1 \setminus\setminus slots)/slots', slots]) \land$
$(\exists\, vv \bullet Q[vv, vv \sharp ((slots' \setminus\setminus slots) \setminus\setminus (slots_1 \setminus\setminus slots))\ /\ slots', slots])\ )$

$\equiv$ “ [DF:subsub]:p22 ”

$\exists\, ss, slots_1 \bullet$
$(\exists\, uu \bullet P[uu, uu \sharp (slots_1 \setminus\setminus slots)/slots', slots]) \land$
$(\exists\, vv \bullet Q[vv, vv \sharp (slots' \setminus\setminus slots_1)\ /\ slots', slots])\ )$

$\equiv$ “ [Seq:def]:p32, backwards, ss not mentioned ”

$(\exists\, uu \bullet P[uu, uu \sharp (slots' \setminus\setminus slots)/slots', slots])\ ;$
$(\exists\, vv \bullet Q[vv, vv \sharp (slots' \setminus\setminus slots)\ /\ slots', slots])$

$\equiv$ “ [R2a:def]:p**??**, backwards, twice ”

$\mathbf{R2a}(P)\ ;\ \mathbf{R2a}(Q)$

$\equiv$ “ P and Q are **R2a**-healthy by assumption ”

$P\ ;\ Q$

$\square$

A.3.38 Proof

of [comp:R2:closed]:p24

$$(P \equiv \mathbf{R2}(P)) \land (Q \equiv \mathbf{R2}(Q)) \Rightarrow ((P;\ Q) \equiv \mathbf{R2}(P;\ Q))$$

$$
\begin{array}{ll}
 & \mathbf{R2}(P;\ Q) \\
\equiv & \quad \text{“ } P \text{ and } Q \text{ are } \mathbf{R2}\text{-healthy ”} \\
 & \mathbf{R2}(\mathbf{R2}(P);\ \mathbf{R2}(Q)) \\
\equiv & \quad \text{“ [R2:alt]:p24 twice ”} \\
 & \mathbf{R2}(\,(\exists\, uu \bullet R2_{uu}(P) \land RE(uu, slots)) \\
 & \qquad ;\ (\exists\, vv \bullet R2_{vv}(Q) \land RE(vv, slots))\,) \\
\equiv & \quad \text{“ [Seq:def]:p32 ”} \\
 & \mathbf{R2}(\exists\, obs_0 \bullet \\
 & \qquad (\exists\, uu \bullet R2_{uu}(P) \land RE(uu, slots))[seq'] \\
 & \qquad \land (\exists\, vv \bullet R2_{vv}(Q) \land RE(vv, slots))[seq]\,) \\
\equiv & \quad \text{“ [Seq:subs]:p32, [Seq:subs']:p32 ”} \\
 & \mathbf{R2}(\exists\, obs_0 \bullet \\
 & \qquad (\exists\, uu \bullet (R2_{uu}(P))[seq'] \land RE(uu, slots)) \\
 & \qquad \land (\exists\, vv \bullet (R2_{vv}(Q))[seq] \land RE(vv, slots_0))\,) \\
\equiv & \quad \text{“ [R2:subs]:p24 twice ”} \\
 & \mathbf{R2}(\exists\, obs_0 \bullet \\
 & \qquad (\exists\, uu \bullet P[uu, uu \sharp (slots' \backslash\backslash slots)/slots, slots'][seq'] \land RE(uu, slots)) \\
 & \qquad \land (\exists\, vv \bullet Q[vv, vv \sharp (slots' \backslash\backslash slots)/slots, slots'][seq] \land RE(vv, slots_0))\,) \\
\equiv & \quad \text{“ applying } [seq] \text{ subs, } P_0, Q_0 = P[rest_0/rest'], Q[rest_0/rest] \text{, for } obs = rest, slots \text{ ”} \\
 & \mathbf{R2}(\exists\, rest_0, slots_0 \bullet \\
 & \qquad (\exists\, uu \bullet P_0[uu, uu \sharp (slots_0 \backslash\backslash slots)/slots, slots'] \land RE(uu, slots)) \\
 & \qquad \land (\exists\, vv \bullet Q_0[vv, vv \sharp (slots' \backslash\backslash slots_0)/slots, slots'] \land RE(vv, slots_0))\,) \\
\equiv & \quad \text{“ [R2:alt]:p24 ”} \\
 & \exists\, ss \bullet RE(ss, slots) \land \\
 & \quad R2_{ss}(\,\exists\, rest_0, slots_0 \bullet \\
 & \qquad (\exists\, uu \bullet P_0[uu, uu \sharp (slots_0 \backslash\backslash slots)/slots, slots'] \land RE(uu, slots)) \\
 & \qquad \land (\exists\, vv \bullet Q_0[vv, vv \sharp (slots' \backslash\backslash slots_0)/slots, slots'] \land RE(vv, slots_0))\,) \\
\equiv & \quad \text{“ [R2:subs]:p24 ”} \\
 & \exists\, ss \bullet RE(ss, slots) \land \\
 & \quad \exists\, rest_0, slots_0 \bullet \\
 & \qquad (\exists\, uu \bullet P_0[uu, uu \sharp (slots_0 \backslash\backslash ss)/slots, slots'] \land RE(uu, ss)) \\
 & \qquad \land (\exists\, vv \bullet Q_0[vv, vv \sharp ((ss \sharp (slots' \backslash\backslash slots)) \backslash\backslash slots_0)/slots, slots'] \land RE(vv, slots_0))
\end{array}
$$

At this point we introduce a change of variable: $slots_0 = ss \sharp (slots_1 \backslash\backslash slots)$. The usual justification is the following bidirectional inference rule:

$$\frac{\exists\, x : A \bullet P(x)}{\exists\, y : B \bullet P(f(y))} \quad f : B \twoheadrightarrow A,$$

The function f giving the old variable x in terms of the new one y has to be surjective, so all possible candidate xs are covered. However, the expression $ss \sharp (slots_1 \setminus\!\!\setminus slots)$, viewed as a function from $slots_1$ to $slots_0$ is not surjective. However, we note that we are in a context where ss is itself existentially quantified, so we apply the following rule:

$$\frac{\exists\, x : A, c : C \bullet P(x, c)}{\exists\, y : B, c : C \bullet P(f(y, c), c)} \quad f : B \times C \twoheadrightarrow A,$$

The expression $ss \sharp (slots_1 \setminus\!\!\setminus slots)$ viewed as a function over ss and $slots_1$ is surjective. Any value of $slots_0$ can be obtained, regardless of the value of $slots$ by choosing ss and $slots_1$ to satisfy the following conditions:

$$\begin{aligned} ss &= slots_0 \\ slots_1 &\cong slots \\ eqvref(slots_1) &= eqvref(slots_0) \end{aligned}$$

We now continue with the proof, having made the variable change:

$$\begin{array}{ll}
& \exists\, ss \bullet RE(ss, slots) \land \\
& \quad \exists\, rest_0, slots_1 \bullet \\
& \qquad (\exists\, uu \bullet P_0[uu, uu \sharp ((ss \sharp (slots_1 \setminus\!\!\setminus slots)) \setminus\!\!\setminus ss)/slots, slots'] \land RE(uu, ss)) \\
& \qquad \land (\exists\, vv \bullet Q_0[vv, vv \sharp ((ss \sharp (slots' \setminus\!\!\setminus slots)) \setminus\!\!\setminus (ss \sharp (slots_1 \setminus\!\!\setminus slots)))/slots, slots'] \\
& \qquad\quad \land RE(vv, ss \sharp (slots_1 \setminus\!\!\setminus slots))) \\
\equiv & \quad \text{“ [CAT:DF:id]:p22 line3, [CAT:DF:pfx]:p22 line 4, [CAT:ER:last]:p21 line 5 ”} \\
& \exists\, ss \bullet RE(ss, slots) \land \\
& \quad \exists\, rest_0, slots_1 \bullet \\
& \qquad (\exists\, uu \bullet P_0[uu, uu \sharp (slots_1 \setminus\!\!\setminus slots)/slots, slots'] \land RE(uu, ss)) \\
& \qquad \land (\exists\, vv \bullet Q_0[vv, vv \sharp ((slots' \setminus\!\!\setminus slots) \setminus\!\!\setminus (slots_1 \setminus\!\!\setminus slots))/slots, slots'] \\
& \qquad\quad \land RE(vv, slots_1 \setminus\!\!\setminus slots) \\
\equiv & \quad \text{“ [DF:subsub]:p22 line 4, [DF:ER:first]:p22 line 5 ”} \\
& \exists\, ss \bullet RE(ss, slots) \land \\
& \quad \exists\, rest_0, slots_1 \bullet \\
& \qquad (\exists\, uu \bullet P_0[uu, uu \sharp (slots_1 \setminus\!\!\setminus slots)/slots, slots'] \land RE(uu, ss)) \\
& \qquad \land (\exists\, vv \bullet Q_0[vv, vv \sharp (slots' \setminus\!\!\setminus slots_1)/slots, slots'] \\
& \qquad\quad \land RE(vv, slots_1)) \\
\equiv & \quad \text{“ } RE(uu, ss) \land RE(ss, slots) \equiv RE(uu, slots) \land RE(ss, slots) \text{ ”} \\
& \exists\, ss \bullet RE(ss, slots) \land \\
& \quad \exists\, rest_0, slots_1 \bullet \\
& \qquad (\exists\, uu \bullet P_0[uu, uu \sharp (slots_1 \setminus\!\!\setminus slots)/slots, slots'] \land RE(uu, slots)) \\
& \qquad \land (\exists\, vv \bullet Q_0[vv, vv \sharp (slots' \setminus\!\!\setminus slots_1)/slots, slots'] \\
& \qquad\quad \land RE(vv, slots_1))
\end{array}$$

$\equiv$ “ narrow scope of $\exists\, ss$ ”

$(\exists\, ss \bullet RE(ss, slots)) \land$

$(\,\exists\, rest_0, slots_1 \bullet$

$\quad (\exists\, uu \bullet P_0[uu, uu \sharp (slots_1 \setminus\!\!\setminus slots)/slots, slots'] \land RE(uu, slots))$

$\quad \land (\exists\, vv \bullet Q_0[vv, vv \sharp (slots' \setminus\!\!\setminus slots_1)/slots, slots']$

$\quad\quad \land RE(vv, slots_1))\,)$

$\equiv$ “ witness $ss = slots$ line 1, α-rename $slots_1$ to $slots_0$ ”

$\exists\, rest_0, slots_0 \bullet$

$\quad (\exists\, uu \bullet P_0[uu, uu \sharp (slots_0 \setminus\!\!\setminus slots)/slots, slots'] \land RE(uu, slots))$

$\quad \land (\exists\, vv \bullet Q_0[vv, vv \sharp (slots' \setminus\!\!\setminus slots_1)/slots, slots']$

$\quad\quad \land RE(vv, slots_0))$

$\equiv$ “ defn. $P_0,\ Q_0$ ”

$\exists\, rest_0, slots_0 \bullet$

$\quad (\exists\, uu \bullet P[rest_0/rest'][uu, uu \sharp (slots_0 \setminus\!\!\setminus slots)/slots, slots'] \land RE(uu, slots))$

$\quad \land (\exists\, vv \bullet Q[rest_0/rest][vv, vv \sharp (slots' \setminus\!\!\setminus slots_1)/slots, slots']$

$\quad\quad \land RE(vv, slots_0))$

$\equiv$ “ [Seq:def]:p32 backwards ”

$(\exists\, uu \bullet P[uu, uu \sharp (slots' \setminus\!\!\setminus slots)/slots, slots'] \land RE(uu, slots))$

$\quad ;\ (\exists\, vv \bullet Q[vv, vv \sharp (slots' \setminus\!\!\setminus slots)/slots, slots'] \land RE(vv, slots))$

$\equiv$ “ [R2:alt]:p24, twice backwards ”

$\mathbf{R2}(P);\ \mathbf{R2}(Q)$

$\equiv$ “ P and Q are $\mathbf{R2}$-healthy by assumption ”

$P;\ Q$

$\Box$

A.3.39 Proof

of [comp:CSP1:closed]

$$(P \equiv \mathbf{CSP1}(P)) \land (Q \equiv \mathbf{CSP1}(Q)) \quad \Rightarrow \quad ((P;\ Q) \equiv \mathbf{CSP1}(P;\ Q))$$

We assume the antecedent:

[comp:CSP1:closed:hyp1] $P \equiv \mathbf{CSP1}(P)$
[comp:CSP1:closed:hyp2] $Q \equiv \mathbf{CSP1}(Q)$

We then convert LHS to RHS:

$$
\begin{array}{ll}
 & P;\ Q \\
\equiv & \quad \text{“ [comp:CSP1:closed:hyp1], [comp:CSP1:closed:hyp2] ”} \\
 & \mathbf{CSP1}(P);\ \mathbf{CSP1}(Q) \\
\equiv & \quad \text{“ [CSP1:def]:p27 ”} \\
 & (P \lor DIV);\ (Q \lor DIV) \\
\equiv & \quad \text{“ Seq:def ”} \\
 & \exists\, obs_0 \bullet (P[obs_0/obs'] \lor DIV[obs_0/obs']) \land (Q[obs_0/obs] \lor DIV[obs_0/obs]) \\
\equiv & \quad \text{“ distributivity ”} \\
 & \exists\, obs_0 \bullet \\
 & \qquad P[obs_0/obs'] \land Q[obs_0/obs] \\
 & \quad \lor DIV[obs_0/obs'] \land Q[obs_0/obs] \\
 & \quad \lor P[obs_0/obs'] \land DIV[obs_0/obs] \\
 & \quad \lor DIV[obs_0/obs'] \land DIV[obs_0/obs] \\
\equiv & \quad \text{“ [DIV:def]:p24, substitution ”} \\
 & \exists\, obs_0 \bullet \\
 & \qquad P[obs_0/obs'] \land Q[obs_0/obs] \\
 & \quad \lor \neg\, ok \land slots \preccurlyeq slots_0 \land Q[obs_0/obs] \\
 & \quad \lor P[obs_0/obs'] \land \neg\, ok_0 \land slots_0 \preccurlyeq slots' \\
 & \quad \lor \neg\, ok \land slots \preccurlyeq slots_0 \land \neg\, ok_0 \land slots_0 \preccurlyeq slots'
\end{array}
$$

A.3.40 Proof

of [comp:prsv:SSEQ]

[comp:prsv:SSEQ] $\quad (P \Rightarrow SSEQ) \land (Q \Rightarrow SSEQ) \equiv ((P;\ Q) \Rightarrow SSEQ)$

We note that $(A \Rightarrow B) \equiv (A \equiv A \land B)$, and so we assume $P = P \land SSEQ$, $Q = Q \land SSEQ$ to show that

$$(P;\ Q) \equiv (P;\ Q) \land SSEQ$$

Proof, start with LHS:

$$
\begin{array}{ll}
& P;\ Q \\
\equiv & \quad \text{“ assumption ”} \\
& (P \land SSEQ);\ (Q \land SSEQ) \\
\equiv & \quad \text{“ [Seq:def]:p32,[SSEQ:def]:p\textbf{??} ”} \\
& \exists\, obs_0 \bullet P[seq'] \land Q[seq] \land slots_0 = slots \land slots' = slots_0 \\
\equiv & \quad \text{“ = is transitive ”} \\
& \exists\, obs_0 \bullet P[seq'] \land Q[seq] \land slots_0 = slots \land slots' = slots_0 \land slots' = slots \\
\equiv & \quad \text{“ [SSEQ:def]:p\textbf{??},} SSEQ \text{ not bound by quantifier ”} \\
& (\exists\, obs_0 \bullet P[seq'] \land Q[seq] \land slots_0 = slots \land slots' = slots_0) \land SSEQ
\end{array}
$$

Now the RHS:

$$
\begin{array}{ll}
& (P;\ Q) \land SSEQ \\
\equiv & \quad \text{“ [Seq:def]:p32 ”} \\
& (\exists\, obs_0 \bullet P[seq'] \land Q[seq]) \land SSEQ \\
\equiv & \quad \text{“ Assumption ”} \\
& (\exists\, obs_0 \bullet (P \land SSEQ)[seq'] \land (Q \land SSEQ)[seq]) \land SSEQ \\
\equiv & \quad \text{“ Substitution ”} \\
& (\exists\, obs_0 \bullet P[seq'] \land SSEQ[seq'] \land Q[seq] \land SSEQ)[seq]) \land SSEQ \\
\equiv & \quad \text{“ [SSEQ:def]:p\textbf{??},[Seq:subs']:p32,[Seq:subs]:p32 ”} \\
& (\exists\, obs_0 \bullet P[seq'] \land slots_0 = slots \land Q[seq] \land slots' = slots_0) \land SSEQ
\end{array}
$$

LHS and RHS are identical, modulo reordering

□

A.3.41 Proof

of [comp:R1:prsv:SSEQ]

[comp:R1:prsv:SSEQ] $\quad (P \equiv \mathbf{R1}(P) \wedge (Q \equiv \mathbf{R1}(Q))$
$\Rightarrow$
$(P \wedge SSEQ);\ (P \wedge SSEQ) \equiv (P;\ Q) \wedge SSEQ$

Proof, LHS first

$$
\begin{array}{ll}
 & (P \wedge SSEQ);\ (P \wedge SSEQ) \\
\equiv & \quad \text{“ [Seq:def]:p32,[SSEQ:def]:p\textbf{??} ”} \\
 & \exists\, obs_0 \bullet P[seq'] \wedge slots = slots_0 \wedge Q[seq] \wedge slots_0 = slots \\
\equiv & \quad \text{“ = is transitive ”} \\
 & \exists\, obs_0 \bullet P[seq'] \wedge slots = slots_0 \wedge Q[seq] \wedge slots_0 = slots \wedge slots = slots'
\end{array}
$$

RHS:

$$
\begin{array}{ll}
 & (P;\ Q) \wedge SSEQ \\
\equiv & \quad \text{“ } P \text{ and } Q \text{ are } \mathbf{R1} \text{ ”} \\
 & ((P \wedge slots \preccurlyeq slots');\ (Q \wedge slots \preccurlyeq slots')) \wedge SSEQ \\
\equiv & \quad \text{“ [Seq:def]:p32 ”} \\
 & (\exists\, obs_0 \bullet P[seq'] \wedge slots \preccurlyeq slots'_0 \wedge Q[seq] \wedge slots_0 \preccurlyeq slots') \wedge SSEQ \\
\equiv & \quad \text{“ [SSEQ:def]:p\textbf{??}, distr, into quantifier ”} \\
 & \exists\, obs_0 \bullet P[seq'] \wedge slots \preccurlyeq slots'_0 \wedge Q[seq] \wedge slots_0 \preccurlyeq slots' \wedge slots' = slots \\
\equiv & \quad \text{“ } s = s' \wedge s \preccurlyeq s_0 \wedge s_0 \preccurlyeq s' \text{ means } s_0 = s = s' \text{ ”} \\
 & \exists\, obs_0 \bullet P[seq'] \wedge slots = slots'_0 \wedge Q[seq] \wedge slots_0 = slots' \wedge slots' = slots
\end{array}
$$

Both sides are equal

□

A.3.42 Proof

of [comp:R3:closed]:p25

$$(P \equiv \mathbf{R3}(P)) \wedge (Q \equiv \mathbf{R3}(Q)) \Rightarrow ((P;\ Q) \equiv \mathbf{R3}(P;\ Q))$$

We assume

[comp:R3:closed:hyp1] $P \equiv \mathbf{R3}P)$

[comp:R3:closed:hyp2] $Q \equiv \mathbf{R3}(Q)$

to show:

$P;\ Q$

$\equiv$ “ [comp:R3:closed:hyp1]:p156,[comp:R3:closed:hyp2]:p156 ”

$\mathbf{R3}(P);\ \mathbf{R3}(Q)$

$\equiv$ “ [R3:def]:p24 ”

$(I\!I_R \lhd wait \rhd P);\ (I\!I_R \lhd wait \rhd Q)$

$\equiv$ “ [Seq:def]:p32 ”

$\exists\, obs_0 \bullet (I\!I_R \lhd wait \rhd P)[seq'] \wedge (I\!I_R \lhd wait \rhd Q)[seq]$

$\equiv$ “ [Cond:def]:p32 ”

$\exists\, obs_0 \bullet (wait \wedge I\!I_R \vee \neg\, wait \wedge P)[seq'] \wedge (wait \wedge I\!I_R \vee \neg\, wait \wedge Q)[seq]$

$\equiv$ “ defn. of substitution, [Seq:subs]:p32,[Seq:subs']:p32 ”

$\exists\, obs_0 \bullet (wait \wedge I\!I_R[seq'] \vee \neg\, wait \wedge P[seq']) \wedge (wait_0 \wedge I\!I_R[seq] \vee \neg\, wait_0 \wedge Q[seq])$

$\equiv$ “ $\wedge$–$\vee$ distributivity ”

$\exists\, obs_0 \bullet$

$wait \wedge I\!I_R[seq'] \wedge wait_0 \wedge I\!I_R[seq]$

$\vee\ wait \wedge I\!I_R[seq'] \wedge \neg\, wait_0 \wedge Q[seq]$

$\vee\ \neg\, wait \wedge P[seq'] \wedge wait_0 \wedge I\!I_R[seq]$

$\vee\ \neg\, wait \wedge P[seq'] \wedge \neg\, wait_0 \wedge Q[seq]$

$\equiv$ “ Collect terms guarded by $wait$ ”

$\exists\, obs_0 \bullet$

$wait \wedge (wait_0 \wedge I\!I_R[seq'] \wedge I\!I_R[seq] \vee \neg\, wait_0 \wedge I\!I_R[seq'] \wedge Q[seq])$

$\vee$

$\neg\, wait \wedge (wait_0 \wedge P[seq'] \wedge I\!I_R[seq] \vee \neg\, wait_0 \wedge P[seq'] \wedge Q[seq])$

$\equiv$ “ Collect terms guarded by $wait_0$ ”

$\exists\, obs_0 \bullet$

$(I\!I_R[seq'] \wedge I\!I_R[seq] \lhd wait_0 \rhd I\!I_R[seq'] \wedge Q[seq])$

$\lhd wait \rhd$

$(P[seq'] \wedge I\!I_R[seq] \lhd wait_0 \rhd P[seq'] \wedge Q[seq])$

$\equiv$ “ $(A \wedge B) \lhd C \rhd (A \wedge D) \equiv A \wedge (B \lhd C \rhd D)$ ”

(see overleaf)

$\exists\, obs_0 \bullet$
$\quad (I\!I_R[seq'] \land (I\!I_R[seq] \lhd wait_0 \rhd Q[seq]))$
$\quad \lhd wait \rhd$
$\quad (P[seq'] \land (I\!I_R[seq] \lhd wait_0 \rhd Q[seq]))$
$\equiv$ “ $\exists\, obs_0$ distributes through $\lhd wait \rhd$ ”
$(\exists\, obs_0 \bullet I\!I_R[seq'] \land (I\!I_R[seq] \lhd wait_0 \rhd Q[seq]))$
$\quad \lhd wait \rhd$
$(\exists\, obs_0 \bullet P[seq'] \land (I\!I_R[seq] \lhd wait_0 \rhd Q[seq]))$
$\equiv$ “ substitution backwards ”
$(\exists\, obs_0 \bullet I\!I_R[seq'] \land (I\!I_R \lhd wait \rhd Q)[seq])$
$\quad \lhd wait \rhd$
$(\exists\, obs_0 \bullet P[seq'] \land (I\!I_R \lhd wait \rhd Q)[seq])$
$\equiv$ “ [Seq:def]:p32, backwards ”
$(I\!I_R;\ (I\!I_R \lhd wait \rhd Q)) \lhd wait \rhd (P;\ I\!I_R \lhd wait \rhd Q))$
$\equiv$ “ Q is **R3** ”
$(I\!I_R;\ Q) \lhd wait \rhd (P;\ Q)$
$\equiv$ “ [R3:wait:Skip]:p25 ”
$(I\!I_R;\ I\!I_R) \lhd wait \rhd (P;\ Q)$
$\equiv$ “ [SkipR-SkipR:eq:SkipR]:p25 ”
$I\!I_R \lhd wait \rhd (P;\ Q)$
$\equiv$ “ [R3:def]:p24, backwards ”
$\mathbf{R3}(P;\ Q)$
$\Box$

A.3.43 Proof

of [DIV:DIV:eq:DIV]:p25

$$DIV;\ DIV \quad \equiv \quad DIV$$

$$
\begin{array}{ll}
 & DIV;\ DIV \\
\equiv & \text{“ [DIV:def]:p24 ”} \\
 & (\neg\ ok \land slots \preccurlyeq slots');\ (\neg\ ok \land slots \preccurlyeq slots') \\
\equiv & \text{“ [Seq:def]:p32, drop unused q. vars. ”} \\
 & \exists\, ok_0, slots_0 \bullet \neg\ ok \land slots \preccurlyeq slots_0 \land \neg\ ok_0 \land slots_0 \preccurlyeq slots' \\
\equiv & \text{“ [EX:trans]:p19 ”} \\
 & \exists\, ok_0, slots_0 \bullet \neg\ ok \land slots \preccurlyeq slots_0 \land \neg\ ok_0 \land slots_0 \preccurlyeq slots' \land slots \preccurlyeq slots' \\
\equiv & \text{“ shrink and split scopes ”} \\
 & \neg\ ok \land slots \preccurlyeq slots' \land (\exists\, ok_0 \bullet \neg\ ok_0) \land (\exists\, slots_0 \bullet slots \preccurlyeq slots_0 \land slots_0 \preccurlyeq slots') \\
\equiv & \text{“ take } ok_0 = \textsc{False}\text{, and } slots_0 = slots \text{ ”} \\
 & \neg\ ok \land slots \preccurlyeq slots' \land (slots \preccurlyeq slots') \\
\equiv & \text{“ idempotence, [DIV:def]:p24 backwards ”} \\
 & DIV \\
\Box &
\end{array}
$$

A.3.44 Proof

of [DIV:A:eq:DIV]:p**??**

$$DIV;\ A \quad \equiv \quad DIV, \qquad A \text{ healthy}$$

$$\begin{aligned} & DIV;\ A \\ \equiv & \quad \text{“ [Seq:def]:p32, [DIV:def]:p24 ”} \\ & \exists\, obs_0 \bullet \neg\, ok \wedge slots \preccurlyeq slots_0 \wedge A[obs_0/obs] \end{aligned}$$

There isn't a simple relationship here - A can influence the overall outcome.

A.3.45 Proof

of [A:DIV:eq:DIV]:p**??**

$$A;\ DIV \quad \equiv \quad DIV, \qquad A \text{ healthy}$$

$$\begin{aligned} & A;\ DIV \\ \equiv & \quad \text{“ [Seq:def]:p32, [DIV:def]:p24 ”} \\ & \exists\, obs_0 \bullet A[obs_0/obs'] \wedge \neg\, ok_0 \wedge slots_0 \preccurlyeq slots' \end{aligned}$$

DIV should mask the outcome of A.

A.3.46 Proof

of [IIr:DIV:eq:DIV]:p25

$$I\!I_R;\ DIV \quad \equiv \quad DIV$$

$$
\begin{array}{ll}
 & I\!I_R;\ DIV \\
\equiv & \quad \text{“ [IIr:def]:p24 ”} \\
 & (DIV \lor ok' \land RSTET);\ DIV \\
\equiv & \quad \text{“ } \lor\text{–;}\ \text{distributivity ”} \\
 & (DIV;\ DIV) \lor (ok' \land RSTET);\ DIV \\
\equiv & \quad \text{“ [DIV:DIV:eq:DIV]:p25, [DIV:def]:p24, [Seq:def]:p32 ”} \\
 & DIV \lor (\exists\, obs_0 \bullet ok_0 \land RSTET[obs_0/obs'] \land \neg\, ok_0 \land slots_0 \preccurlyeq slots') \\
\equiv & \quad \text{“ quantifier body reduces to } \textsc{False} \text{ ”} \\
 & DIV \\
\Box &
\end{array}
$$

A.3.47 Proof

of [DIV:IIr:eq:DIV]:p25

$$DIV;\ I\!I_R \quad \equiv \quad DIV$$

$$
\begin{array}{ll}
 & DIV;\ I\!I_R \\
\equiv & \quad \text{“ [IIr:def]:p24 ”} \\
 & DIV;\ (DIV \lor ok' \land RSTET) \\
\equiv & \quad \text{“ } \lor\text{–;}\ \text{distributivity ”} \\
 & (DIV;\ DIV) \lor (DIV;\ (ok' \land RSTET)) \\
\equiv & \quad \text{“ [DIV:DIV:eq:DIV]:p25 [Seq:def]:p32 ”} \\
 & DIV \lor (\exists\, obs_0 \bullet DIV[obs_0/obs'] \land ok' \land RSTET[obs_0/obs]) \\
\equiv & \quad \text{“ [one-point:RSTET]:p25 ”} \\
 & DIV \lor (\exists\, ok_0 \bullet DIV[ok_0/ok'] \land ok' \\
\equiv & \quad \text{“ } ok' \text{ not free in } DIV\text{, drop quantifier ”} \\
 & DIV \lor (DIV \land ok') \\
\equiv & \quad \text{“ absorption ”} \\
 & DIV \\
\Box &
\end{array}
$$

A.3.48 Proof

of [IIr:is:CSP2]:p28

$$\mathbf{CSP2}(\mathit{II}_R) \quad = \quad \mathit{II}_R$$

$$
\begin{array}{ll}
 & \mathbf{CSP2}(\mathit{II}_R) \\
\equiv & \quad \text{“ [CSP2:def]:p27 ”} \\
 & \mathit{II}_R;\ (ok \Rightarrow ok' \land wait' = wait \land state' = state \land slots' = slots) \\
\equiv & \quad \text{“ [IIr:def]:p24 ”} \\
 & (\neg\ ok \land slots \preccurlyeq slots' \\
 & \ \lor ok' \land wait' = wait \land slots' = slots) \\
 & ;\ (ok \Rightarrow ok' \land wait' = wait \land state' = state \land slots' = slots) \\
\equiv & \quad \text{“ [Seq:def]:p32 ”} \\
 & \exists\, ok_0, wait_0, state_0, slots_0, \bullet \\
 & \quad (\neg\ ok \land slots \preccurlyeq slots_0 \\
 & \qquad \lor ok_0 \land wait_0 = wait \land slots_0 = slots) \\
 & \quad \land (ok_0 \Rightarrow ok' \land wait' = wait_0 \land state' = state_0 \land slots' = slots_0) \\
\equiv & \quad \text{“ $\land$-,$\lor$- and $\exists$-distributivity ”} \\
 & (\exists\, ok_0, wait_0, state_0, slots_0, \bullet \\
 & \quad \neg\ ok \land slots \preccurlyeq slots_0 \\
 & \quad \land ok_0 \Rightarrow ok' \land wait' = wait_0 \land state' = state_0 \land slots' = slots_0) \\
 & \lor \\
 & (\exists\, ok_0, wait_0, state_0, slots_0, \bullet \\
 & \quad ok_0 \land wait_0 = wait \land slots_0 = slots \\
 & \land ok_0 \Rightarrow ok' \land wait' = wait_0 \land state' = state_0 \land slots' = slots_0) \\
\equiv & \quad \text{“ one-point rule for $wait_0$, $state_0$, $slots_0$ ”} \\
 & (\exists\, ok_0 \bullet \neg\ ok \land ok_0 \Rightarrow ok' \land slots \preccurlyeq slots') \\
 & \lor \\
 & (\exists\, ok_0 \bullet ok_0 \land ok_0 \Rightarrow ok' \land wait' = wait \land slots' = slots\,) \\
\equiv & \quad \text{“ $(\,\exists\, b : \mathbb{B} \bullet P(b)\,) \equiv P(\textsc{False}) \lor P(\textsc{True})$ ”} \\
 & (\neg\ ok \land \textsc{False} \Rightarrow ok' \land slots \preccurlyeq slots') \\
 & \lor (\neg\ ok \land \textsc{True} \Rightarrow ok' \land slots \preccurlyeq slots') \\
 & \lor (\textsc{False} \land \textsc{False} \Rightarrow ok' \land wait' = wait \land slots' = slots\,) \\
 & \lor (\textsc{True} \land \textsc{True} \Rightarrow ok' \land wait' = wait \land slots' = slots\,) \\
\equiv & \quad \text{“ prop. calc ”} \\
 & (\neg\ ok \land slots \preccurlyeq slots') \lor (ok' \land wait' = wait \land slots' = slots\,) \\
\equiv & \quad \text{“ [IIr:def]:p24 ”} \\
 & \mathit{II}_R
\end{array}
$$

□

A.3.49 Lemma

[DIV:is:CSP2]

[DIV:is:CSP2] $\quad \mathbf{CSP2}(DIV) = DIV$

Begin with the lhs:

$$
\begin{array}{ll}
 & \mathbf{CSP2}(DIV) \\
\equiv & \text{“ [DIV:def]:p24,[CSP2:def]:p27 ”} \\
 & \neg\, ok \land slots \preccurlyeq slots';\ ok \Rightarrow ok' \land wait = wait' \land state = state' \land slots = slots' \\
\equiv & \text{“ [Seq:def]:p32 ”} \\
 & \exists\, ok_0, wait_0, state_0, slots_0 \bullet \\
 & \qquad \neg\, ok \land slots \preccurlyeq slots_0 \\
 & \quad \land\, ok_0 \Rightarrow ok' \land wait_0 = wait' \land state_0 = state' \land slots_0 = slots' \\
\equiv & \text{“ one-point } wait_0, state_0, slots_0 \text{ ”} \\
 & \exists\, ok_0 \bullet \neg\, ok \land slots \preccurlyeq slots' \land ok_0 \Rightarrow ok' \\
\equiv & \text{“ } (\,\exists\, b : \mathbb{B} \bullet P(b)\,) \equiv P(\textsc{False}) \lor P(\textsc{True}) \text{ ”} \\
 & \neg\, ok \land slots \preccurlyeq slots' \land \textsc{False} \Rightarrow ok' \\
 & \lor \neg\, ok \land slots \preccurlyeq slots' \land \textsc{True} \Rightarrow ok' \\
\equiv & \text{“ prop. calc. ”} \\
 & \neg\, ok \land slots \preccurlyeq slots' \land \textsc{True} \\
 & \lor \neg\, ok \land slots \preccurlyeq slots' \land ok' \\
\equiv & \text{“ more prop. calc. ”} \\
 & \neg\, ok \land slots \preccurlyeq slots' \\
\equiv & \text{“ [DIV:def]:p24 backwards ”} \\
 & DIV
\end{array}
$$

□

A.3.50 Proof

of [R1:CSP2:comm]:p28

$$[\mathsf{R1{:}CSP2{:}comm}] : p28 \qquad \mathbf{R1} \circ \mathbf{CSP2} = \mathbf{CSP2} \circ \mathbf{R1}$$

Start with lhs:

$$
\begin{array}{ll}
 & \mathbf{R1}(\mathbf{CSP2}(P)) \\
\equiv & \quad \text{“ [CSP2:def]:p27 ”} \\
 & \mathbf{R1}(P;\ (ok \Rightarrow ok' \wedge wait' = wait \wedge state' = state \wedge slots' = slots)) \\
\equiv & \quad \text{“ [R1:def]:p23 ”} \\
 & (P;\ (ok \Rightarrow ok' \wedge wait' = wait \wedge state' = state \wedge slots' = slots)) \\
 & \wedge\ slots \preccurlyeq slots' \\
\equiv & \quad \text{“ [Seq:def]:p32 ”} \\
 & (\exists\, ok_0, wait_0, state_0, slots_0 \bullet \\
 & \quad P[seq'] \wedge ok_0 \Rightarrow ok' \wedge wait' = wait_0 \wedge state' = state_0 \wedge slots' = slots_0) \\
 & \wedge\ slots \preccurlyeq slots' \\
\equiv & \quad \text{“ One-point } wait_0, state_0, slots_0, \text{ ”} \\
 & (\exists\, ok_0 \bullet P[ok_0/ok'] \wedge ok_0 \Rightarrow ok') \wedge slots \preccurlyeq slots'
\end{array}
$$

Next, the rhs:

$$
\begin{array}{ll}
 & \mathbf{CSP2}(\mathbf{R1}(P)) \\
\equiv & \quad \text{“ [R1:def]:p23 ”} \\
 & \mathbf{CSP2}(P \wedge slots \preccurlyeq slots') \\
\equiv & \quad \text{“ [CSP2:def]:p27 ”} \\
 & (P \wedge slots \preccurlyeq slots'); \\
 & (ok \Rightarrow ok' \wedge wait' = wait \wedge state' = state \wedge slots' = slots) \\
\equiv & \quad \text{“ [Seq:def]:p32 ”} \\
 & \exists\, ok_0, wait_0, state_0, slots_0 \bullet \\
 & \quad P[seq'] \wedge slots \preccurlyeq slots_0 \\
 & \quad \wedge\ ok_0 \Rightarrow ok' \wedge wait' = wait_0 \wedge state' = state_0 \wedge slots' = slots_0 \\
\equiv & \quad \text{“ One-point } wait_0, state_0, slots_0, \text{ ”} \\
 & \exists\, ok_0 \bullet P[ok_0/ok'] \wedge slots \preccurlyeq slots' \wedge ok_0 \Rightarrow ok' \\
\equiv & \quad \text{“ shrink quantifier scope ”} \\
 & (\exists\, ok_0 \bullet P[ok_0/ok'] \wedge ok_0 \Rightarrow ok') \wedge slots \preccurlyeq slots'
\end{array}
$$

Lhs and Rhs are identical □.

A.3.51 Proof

of [R2:CSP2:comm]:p28

$$\mathbf{R2}(\mathbf{CSP2}(P)) \quad = \quad \mathbf{CSP2}(\mathbf{R2}(P))$$

REDO
We use notation expressing P as a function of its free vars.
Lhs:

$$
\begin{array}{ll}
 & \mathbf{R2}(\mathbf{CSP2}(P(ok, wait, state, slots, ok', wait', state', slots'))) \\
\equiv & \text{“ [CSP2:def]:p27 ”} \\
 & \mathbf{R2}(P(ok, wait, state, slots, ok', wait', state', slots') \\
 & \quad ;\ (ok \Rightarrow ok' \land wait' = wait \land state' = state \land slots' = slots)) \\
\equiv & \text{“ [Seq:def]:p32 ”} \\
 & \mathbf{R2}(\exists\, obs_0 \bullet P(ok, wait, state, slots, ok_0, wait_0, state_0, slots_0) \\
 & \qquad \land ok_0 \Rightarrow ok' \land wait' = wait_0 \land state' = state_0 \land slots' = slots_0) \\
\equiv & \text{“ one-point rule, } wait_0,\ state_0 \text{ and } slots_0 \text{ ”} \\
 & \mathbf{R2}(\exists\, ok_0 \bullet P(ok, wait, state, slots, ok_0, wait', state', slots') \land ok_0 \Rightarrow ok') \\
\equiv & \text{“ [R2:def]:p23 ”} \\
 & \exists\, ss, ok_0 \bullet P(ok, wait, state, ss, ok_0, wait', state', ss \sharp (slots' \setminus\setminus slots)) \land ok_0 \Rightarrow ok')
\end{array}
$$

Rhs:

$$
\begin{array}{ll}
 & \mathbf{CSP2}(\mathbf{R2}(P(ok, wait, state, slots, ok', wait', state', slots'))) \\
\equiv & \text{“ [R2:def]:p23 ”} \\
 & \mathbf{CSP2}(\exists\, ss \bullet P(ok, wait, state, ss, ok', wait', state', ss \sharp (slots' \setminus\setminus slots))) \\
\equiv & \text{“ [CSP2:def]:p27 ”} \\
 & (\exists\, ss \bullet P(ok, wait, state, ss, ok', wait', state', ss \sharp (slots' \setminus\setminus slots))) \\
 & ;\ (ok \Rightarrow ok' \land wait' = wait \land state' = state \land slots' = slots) \\
\equiv & \text{“ [Seq:def]:p32 ”} \\
 & \exists\, obs_0 \bullet \\
 & \quad (\exists\, ss \bullet P(ok, wait, state, ss, ok_0, wait_0, state_0, ss \sharp (slots_0 \setminus\setminus slots))) \\
 & \quad \land ok_0 \Rightarrow ok' \land wait' = wait_0 \land state' = state_0 \land slots' = slots_0) \\
\equiv & \text{“ one-point rule, } wait_0,\ state_0 \text{ and } slots_0 \text{ ”} \\
 & \exists\, ok_0 \bullet \\
 & \quad (\exists\, ss \bullet P(ok, wait, state, ss, ok_0, wait', state', ss \sharp (slots' \setminus\setminus slots))) \land ok_0 \Rightarrow ok') \\
\equiv & \text{“ expand scopeof } \exists\, ss\text{, reorder quantifiers ”} \\
 & \exists\, ss, ok_0 \bullet P(ok, wait, state, ss, ok_0, wait', state', ss \sharp (slots' \setminus\setminus slots)) \land ok_0 \Rightarrow ok')
\end{array}
$$

Lhs and Rhs are equal $\Box$.

A.3.52 Proof

of [R3:CSP2:comm]:p28

[R3:CSP2:comm] $\mathbf{R3} \circ \mathbf{CSP2} = \mathbf{CSP2} \circ \mathbf{R3}$

$$
\begin{array}{ll}
 & \mathbf{CSP2}(\mathbf{R3}(P)) \\
\equiv & \quad \text{“ [R3:def]:p24 ”} \\
 & \mathbf{CSP2}(I\!I_R \lhd wait \rhd P) \\
\equiv & \quad \text{“ [CSP2:distr:cond]:p28 ”} \\
 & \mathbf{CSP2}(I\!I_R) \lhd wait \rhd \mathbf{CSP2}(P) \\
\equiv & \quad \text{“ [IIr:is:CSP2]:p28 ”} \\
 & I\!I_R \lhd wait \rhd \mathbf{CSP2}(P) \\
\equiv & \quad \text{“ [R3:def]:p24, backwards ”} \\
 & \mathbf{R3}(\mathbf{CSP2}(P))
\end{array}
$$

A.3.53 Proof

of [CSP1:CSP2:comm]:p28

[CSP1:CSP2:comm] $\mathbf{CSP1} \circ \mathbf{CSP2} = \mathbf{CSP2} \circ \mathbf{CSP1}$

$\mathbf{CSP2}(\mathbf{CSP1}(P))$

$\equiv$ " [CSP1:def]:p27 "

$\mathbf{CSP2}(P \lor DIV)$

$\equiv$ " [CSP2:distr:or]:p28 "

$\mathbf{CSP2}(P) \lor \mathbf{CSP2}(DIV)$

$\equiv$ " [DIV:is:CSP2]:p28 "

$\mathbf{CSP2}(P) \lor DIV$

$\equiv$ " [CSP1:def]:p27, backwards "

$\mathbf{CSP1}(\mathbf{CSP2}(P))$

$\square$

A.3.54 Proof

of [ETs:slots-diff:R1:R2]

[ETs:slots-diff:R1:R2] $\mathbf{R2}(\mathbf{R1}(EQVTRACE(tt, f(slots' \diagdown\!\!\diagdown slots))))$
$\equiv EQVTRACE(tt, f(slots' \diagdown\!\!\diagdown slots))$

We handle **R1** and **R2** separately. First, **R1**, starting with the rhs:

$$
\begin{array}{ll}
 & EQVTRACE(tt, f(slots' \diagdown\!\!\diagdown slots)) \\
\equiv & \quad \text{“ pre-condition of } \diagdown\!\!\diagdown \text{ ”} \\
 & EQVTRACE(tt, f(slots' \diagdown\!\!\diagdown slots)) \land slots \preccurlyeq slots' \\
\equiv & \quad \text{“ [R1:def]:p23 backwards ”} \\
 & \mathbf{R1}(EQVTRACE(tt, f(slots' \diagdown\!\!\diagdown slots)))
\end{array}
$$

Next, **R2**, starting with rhs:

$$
\begin{array}{ll}
 & \mathbf{R2}(EQVTRACE(tt, f(slots' \diagdown\!\!\diagdown slots))) \\
\equiv & \quad \text{“ [R2:alt]:p24 ”} \\
 & \exists\, ss \bullet EQVTRACE(tt, f(ss \sharp (slots' \diagdown\!\!\diagdown slots)) \diagdown\!\!\diagdown ss) \land ER(ss, slots) \\
\equiv & \quad \text{“ [CAT:DF:id]:p22, provided } slots' \diagdown\!\!\diagdown slots \text{ is defined ”} \\
 & \exists\, ss \bullet EQVTRACE(tt, f(slots' \diagdown\!\!\diagdown slots)) \land slots \preccurlyeq slots' \land ER(ss, slots) \\
\equiv & \quad \text{“ shrink scope ”} \\
 & EQVTRACE(tt, f(slots' \diagdown\!\!\diagdown slots)) \land slots \preccurlyeq slots' \land \exists\, ss \bullet ER(ss, slots) \\
\equiv & \quad \text{“ [R1:def]:p23 backwards, witness } ss = slots \text{ ”} \\
 & \mathbf{R1}(EQVTRACE(tt, f(slots' \diagdown\!\!\diagdown slots))) \land \textsc{True} \\
\equiv & \quad \text{“ Previous result ”} \\
 & EQVTRACE(tt, f(slots' \diagdown\!\!\diagdown slots))
\end{array}
$$

We can then complete the full proof:

$$
\begin{array}{ll}
 & \mathbf{R2}(\mathbf{R1}(EQVTRACE(tt, f(slots' \diagdown\!\!\diagdown slots)))) \\
\equiv & \quad \text{“ result about } \mathbf{R1} \text{ just proved ”} \\
 & \mathbf{R2}(EQVTRACE(tt, f(slots' \diagdown\!\!\diagdown slots))) \\
\equiv & \quad \text{“ result about } \mathbf{R2} \text{ just proved ”} \\
 & EQVTRACE(tt, f(slots' \diagdown\!\!\diagdown slots)) \\
\square &
\end{array}
$$

A.3.55 Proof

of [NEV:is:R1:R2]:p30

$$\mathbf{R2}(\mathbf{R1}(NOEVTS(slots, slots'))) \quad \equiv \quad NOEVTS(slots, slots')$$

$\mathbf{R2}(\mathbf{R1}(NOEVTS(slots, slots')))$

$\equiv$ “ [NEV:def]:p30 ”

$\mathbf{R2}(\mathbf{R1}(EQVTRACE(\langle\rangle, slots' \setminus\setminus slots)))$

$\equiv$ “ [ETs:slots-diff:R1:R2]:p167, with $tt = \langle\rangle$ and $f = id$. ”

$EQVTRACE(\langle\rangle, slots' \setminus\setminus slots)$

$\equiv$ “ [NEV:def]:p30 backwards ”

$NOEVTS(slots, slots')$

$\square$

A.3.56 Proof

of [EVN:is:R1:R2]

$$\mathbf{R2}(\mathbf{R1}(EVTSNOW(E)(slots, slots'))) \quad \equiv \quad EVTSNOW(E)(slots, slots')$$

$\mathbf{R2}(\mathbf{R1}(EVTSNOW(E)(slots, slots')))$

$\equiv$ “ [EVN:def]:p30 ”

$\mathbf{R2}(\mathbf{R1}(\exists\, tt \bullet elems(tt) = E \land EQVTRACE(tt, slots' \setminus\!\!\setminus slots) \land \#slots = \#slots'\,))$

$\equiv$ “ [R1:def]:p23, [R2:def]:p23, expand $\exists\, tt$ scope. ”

$\exists\, tt, ss \bullet elems(tt) = E$
$\quad \land\, EQVTRACE(tt, slots' \setminus\!\!\setminus slots)[r2] \land slots \preccurlyeq slots'$
$\quad \land\, ER(ss, slots) \land \#ss = \#(ss \sharp (slots' \setminus\!\!\setminus slots))$

$\equiv$ “ using similar technique to [ETs:slots-diff:R1:R2]:p167, **R2** part ”

$\exists\, tt \bullet elems(tt) = E$
$\quad \land\, EQVTRACE(tt, slots' \setminus\!\!\setminus slots)$
$\quad \land\, \exists\, ss \bullet ER(ss, slots) \land \#ss = \#(ss \sharp (slots' \setminus\!\!\setminus slots))$

$\equiv$ “ [CAT:len]:p21 ”

$\exists\, tt \bullet elems(tt) = E$
$\quad \land\, EQVTRACE(tt, slots' \setminus\!\!\setminus slots)$
$\quad \land\, \exists\, ss \bullet ER(ss, slots) \land \#ss = \#ss + \#(slots' \setminus\!\!\setminus slots) - 1$

$\equiv$ “ [DF:len]:p22 ”

$\exists\, tt \bullet elems(tt) = E$
$\quad \land\, EQVTRACE(tt, slots' \setminus\!\!\setminus slots)$
$\quad \land\, \exists\, ss \bullet ER(ss, slots) \land \#ss = \#ss + (1 + \#slots' - \#slots) - 1$

$\equiv$ “ arithmetic ”

$\exists\, tt \bullet elems(tt) = E$
$\quad \land\, EQVTRACE(tt, slots' \setminus\!\!\setminus slots)$
$\quad \land\, \exists\, ss \bullet ER(ss, slots) \land \#slots' = \#slots$

$\equiv$ “ witness, $ss = slots$ ”

$\exists\, tt \bullet elems(tt) = E$
$\quad \land\, EQVTRACE(tt, slots' \setminus\!\!\setminus slots)$
$\quad \land\, \#slots' = \#slots$

$\equiv$ “ [EVN:def]:p30, backwards ” $EVTSNOW(E)(slots, slots')$

$\square$

A.3.57 Proof

of [IME:is:R1:R2]:p30

$$\mathbf{R2}(\mathbf{R1}(IMMEVTS(slots, slots'))) \quad \equiv \quad IMMEVTS(slots, slots')$$

$\mathbf{R2}(\mathbf{R1}(IMMEVTS(slots, slots'))$

$\equiv$ " [IME:def]:p30 "

$\mathbf{R2}(\mathbf{R1}(EQVTRC(tt, head(slots' \setminus\!\setminus slots)) \land tt \neq \langle\rangle))$

$\equiv$ " [R1:distr:and]:p23 "

$\mathbf{R2}(\mathbf{R1}(EQVTRC(tt, head(slots' \setminus\!\setminus slots))) \land tt \neq \langle\rangle)$

$\equiv$ " [R2:distr:and]:p24, $slots', slots$ not free in $tt \neq \langle\rangle$ "

$\mathbf{R2}(\mathbf{R1}(EQVTRC(tt, head(slots' \setminus\!\setminus slots)))) \land tt \neq \langle\rangle$

$\equiv$ " [ETs:sngl]:p18 backwards "

$\mathbf{R2}(\mathbf{R1}(EQVTRACE(tt, \langle head(slots' \setminus\!\setminus slots)\rangle))) \land tt \neq \langle\rangle$

$\equiv$ " [ETs:slots-diff:R1:R2]:p167, with $f = \lambda\, ss \bullet \langle head(ss)\rangle$. "

$EQVTRACE(tt, \langle head(slots' \setminus\!\setminus slots)\rangle) \land tt \neq \langle\rangle$

$\equiv$ " [ETs:sngl]:p18 "

$EQVTRC(tt, head(slots' \setminus\!\setminus slots)) \land tt \neq \langle\rangle$

$\equiv$ " [IME:def]:p30 backwards "

$IMMEVTS(slots, slots')$

$\square$

A.3.58 Proof

[POSS:is:R1:R2]:p**??**

$$\mathbf{R2}(\mathbf{R1}(POSS(c))) \quad \equiv \quad POSS(c)$$

Proof:

$$\begin{array}{ll}
 & \mathbf{R2}(\mathbf{R1}(POSS(c))) \\
\equiv & \quad \text{“ [R1:R2:comm]:p26 ”} \\
 & \mathbf{R1}(\mathbf{R2}(POSS(c))) \\
\equiv & \quad \text{“ [R1:def]:p23,[R2:def]:p23,[POSS:def]:p33 ”} \\
 & GROW \land \exists\, ss \bullet c \notin \bigcup srefs((ss \sharp (slots' \setminus slots)) \setminus ss) \land ER(ss, slots) \\
\equiv & \quad \text{“ [CAT:DF:id]:p22 ”} \\
 & GROW \land \exists\, ss \bullet c \notin \bigcup srefs(slots' \setminus slots) \land ER(ss, slots) \\
\equiv & \quad \text{“ shrink scope ”} \\
 & GROW \land c \notin \bigcup srefs(slots' \setminus slots) \,\exists\, ss \bullet \land ER(ss, slots) \\
\equiv & \quad \text{“ witness, } ss = slots\text{, definedness of } slots' \setminus slots \text{ entails } GROW \text{ ”} \\
 & c \notin \bigcup srefs(slots' \setminus slots) \\
\equiv & \quad \text{“ [POSS:def]:p33, backwards ”} \\
 & POSS(c) \\
\Box &
\end{array}$$

A.3.59 Proof

[WTC:is:R1:R2]:p33

$$\mathbf{R2}(\mathbf{R1}(WTC(c))) \quad \equiv \quad WTC(c)$$

Proof:

$$\mathbf{R2}(\mathbf{R1}(WTC(c))$$

$\equiv$ " [R1:R2:comm]:p26,[R1:def]:p23,[R2:def]:p23,[WTC:def]:p33 "

$$GROW \land \exists ss \bullet wait' \land POSS(c)[r2] \land NOEVTS(ss, ss \sharp (slots' \setminus\!\setminus slots)) \land ER(ss, slots)$$

$\equiv$ " [POSS:def]:p33 "

$$GROW \land \exists ss \bullet wait' \land c \notin \bigcup srefs((ss \sharp (slots' \setminus\!\setminus slots)) \setminus\!\setminus ss)$$
$$\land NOEVTS(ss, ss \sharp (slots' \setminus\!\setminus slots)) \land ER(ss, slots)$$

$\equiv$ " [CAT:DF:id]:p22, shrink scope "

$$GROW \land wait' \land c \notin \bigcup srefs(slots' \setminus\!\setminus slots)$$
$$\exists ss \bullet NOEVTS(ss, ss \sharp (slots' \setminus\!\setminus slots)) \land ER(ss, slots)$$

$\equiv$ " $\setminus\!\setminus$ definedness, [R2:def]:p23 backwards "

$$wait' \land c \notin \bigcup srefs(slots' \setminus\!\setminus slots)$$
$$\mathbf{R2}(NOEVTS(slots' \setminus\!\setminus slots))$$

$\equiv$ " [POSS:def]:p33 backwards, [NEV:is:R1:R2]:p30 "

$$wait' \land POSS(c) \land NOEVTS(slots' \setminus\!\setminus slots)$$

$\equiv$ " [WTC:def]:p33 "

$$WTC(c)$$

$\square$

A.3.60 Proof

[TRMC:is:R1:R2]:p33

$$\mathbf{R2}(\mathbf{R1}(TRMC(c.e))) \quad \equiv \quad TRMC(c.e)$$

Proof:

$$\begin{array}{ll}
& \mathbf{R2}(\mathbf{R1}(TRMC(c.e))) \\
\equiv & \quad \text{“ [R2:def]:p23,[TRMC:def]:p33 ”} \\
& \exists\, ss \bullet \mathbf{R1}(\neg\, wait' \land EVTSNOW\{c.e\}(slots, slots'))[r2] \land ER(ss, slots) \\
\equiv & \quad \text{“ [R1:distr:and]:p23, shrink scope ”} \\
& \neg\, wait' \land \exists\, ss \bullet \mathbf{R1}(EVTSNOW\{c.e\}(slots, slots'))[r2] \land ER(ss, slots) \\
\equiv & \quad \text{“ [R2:def]:p23 backwards ”} \\
& \neg\, wait' \land \mathbf{R2}(\mathbf{R1}(EVTSNOW\{c.e\}(slots, slots'))) \\
\equiv & \quad \text{“ [EVN:is:R1:R2]:p30 ”} \\
& \neg\, wait' \land EVTSNOW\{c.e\}(slots, slots') \\
\equiv & \quad \text{“ [TRMC:def]:p33 backwards ”} \\
& TRMC(c.e) \\
\Box &
\end{array}$$

A.4 Slotted-*Circus* Specific Actions

A.4.1 Proof

$$[\text{specificALaw-1}] : p31 \qquad EVTSNOW(\emptyset)(slots, slots') \equiv slots \cong slots'$$

$EVTSNOW(\emptyset)(slots, slots')$

$\equiv$ “ [EVN:def]:p30 ”

$\exists tt \bullet elems(tt) = \emptyset \land EQVTRACE(tt, slots' \setminus\!\!\setminus slots) \land \#slots = \#slots' \land slots \preccurlyeq slots'$

$\equiv$ “ $tt = \langle\rangle$ ”

$EQVTRACE(\langle\rangle, slots' \setminus\!\!\setminus slots) \land \#slots = \#slots' \land slots \preccurlyeq slots'$

$\equiv$ “ slots are non empty sequences, $slots = pfx \frown \langle(t, r)\rangle$ ”

$\exists t, r, pfx \bullet slots = pfx \frown \langle(t, r)\rangle$

$\land EQVTRACE(\langle\rangle, slots' \setminus\!\!\setminus slots) \land \#slots = \#slots' \land slots \preccurlyeq slots'$

$\equiv$ “ [EX:pfx]:p19 ”

$\exists t, r, pfx, t', r' \bullet slots = pfx \frown \langle(t, r)\rangle \land slots' = pfx \frown \langle(t', r')\rangle \frown sfx$

$\land EQVTRACE(\langle\rangle, slots' \setminus\!\!\setminus slots) \land \#slots = \#slots' \land slots \preccurlyeq slots'$

$\equiv$ “ $\#slots = \#slots' \Rightarrow (sfx = \langle\rangle)$ ”

$\exists t, r, pfx, t', r' \bullet slots = pfx \frown \langle(t, r)\rangle \land slots' = pfx \frown \langle(t', r')\rangle$

$\land EQVTRACE(\langle\rangle, slots' \setminus\!\!\setminus slots) \land \#slots = \#slots' \land slots \preccurlyeq slots'$

$\equiv$ “ [DF:pfx]:p21 ”

$\exists t, r, pfx, t', r' \bullet slots = pfx \frown \langle(t, r)\rangle \land slots' = pfx \frown \langle(t', r')\rangle$

$\land EQVTRACE(\langle\rangle, \langle(t', r')\rangle \setminus\!\!\setminus \langle(t, r)\rangle) \land \#slots = \#slots' \land slots \preccurlyeq slots'$

$\equiv$ “ [ETs:null]:p18 ”

$\exists t, r, pfx, t', r' \bullet slots = pfx \frown \langle(t, r)\rangle \land slots' = pfx \frown \langle(t', r')\rangle$

$\land EQVTRC(\langle\rangle, (t', r') \setminus (t, r)) \land \#slots = \#slots' \land slots \preccurlyeq slots'$

$\equiv$ “ !!!NEEDS REVISING!!! ”

$\exists t, r, pfx, t', r' \bullet slots = pfx \frown \langle(t, r)\rangle \land slots' = pfx \frown \langle(t', r')\rangle$

$\land t' = t \land \#slots = \#slots' \land slots \preccurlyeq slots'$

$\equiv$ “ $t = t'$ ”

$\exists t, r, pfx, r' \bullet slots = pfx \frown \langle(t, r)\rangle \land slots' = pfx \frown \langle(t, r')\rangle$

$\land \#slots = \#slots' \land slots \preccurlyeq slots'$

$\equiv$ “ !!!NEEDS REVISING!!! ”

$\exists t, r, pfx, r' \bullet slots = pfx \frown \langle(t, r)\rangle \land slots' = pfx \frown \langle(t, r')\rangle \land (t, r') \preceq (t, r)$

$\land \#slots = \#slots' \land slots \preccurlyeq slots'$

$\equiv$ “ [EX:def]:p19 ”

$slots' \preccurlyeq slots \land \#slots = \#slots' \land slots \preccurlyeq slots'$

$\equiv$ “ $\cong$ definition ”

$slots \cong slots'$

A.4.2 Proof

[specificALaw-2] : $p31$ $\quad NOEVTS(slots, slots') \land \#slots = \#slots' \equiv EVTSNOW(\emptyset)(slots, slots')$

A.4.3 Proof

[specificALaw-3] : $p31$ $\qquad \exists\, s, s' \bullet EVTSNOW\{c\}(s, s') \wedge sl' \setminus\!\setminus sl = map(SHide(\{c\}))(s' \setminus\!\setminus s) \equiv sl' \cong sl$

Proof

$$
\begin{array}{ll}
 & \exists\, s, s' \bullet EVTSNOW\{c\}(s, s') \wedge sl' \setminus\!\setminus sl = map(shide(\{c\}))(s' \setminus\!\setminus s) \\
\equiv & \quad \text{“ [EVN:def]:p30 ”} \\
 & \exists\, s, s', tt \bullet elems(tt) = \{c\} \wedge sl' \setminus\!\setminus sl = map(shide(\{c\}))(s' \setminus\!\setminus s) \wedge \\
 & s \preccurlyeq s' \wedge \#s = \#s' \wedge EQVTRACE(tt, s' \setminus\!\setminus s) \\
\equiv & \quad \text{“ [DF:len]:p22 ”} \\
 & \exists\, s, s', tt \bullet elems(tt) = \{c\} \wedge sl' \setminus\!\setminus sl = map(shide(\{c\}))(s' \setminus\!\setminus s) \wedge \\
 & s \preccurlyeq s' \wedge \#(s' \setminus\!\setminus s) = 1 \wedge EQVTRACE(tt, s' \setminus\!\setminus s) \\
\equiv & \quad \text{“ } s' \setminus\!\setminus s = \langle (t, r) \rangle \text{ ”} \\
 & \exists\, s, s', tt, t, r \bullet elems(tt) = \{c\} \wedge sl' \setminus\!\setminus sl = map(shide(\{c\}))(\langle (t, r) \rangle) \wedge \\
 & s \preccurlyeq s' \wedge \#\langle (t, r) \rangle = 1 \wedge EQVTRACE(tt, \langle (t, r) \rangle) \wedge s' \setminus\!\setminus s = \langle (t, r) \rangle \\
\equiv & \quad \text{“ Let } s = \langle SNull(r) \rangle \text{ and } s' = \langle (t, r) \rangle \text{ ”} \\
 & \exists\, tt, t, r \bullet elems(tt) = \{c\} \wedge sl' \setminus\!\setminus sl = map(shide(\{c\}))(\langle (t, r) \rangle) \wedge \\
 & EQVTRACE(tt, \langle (t, r) \rangle) \\
\equiv & \quad \text{“ [ETs:def:cons]:p18 ”} \\
 & \exists\, tt, t, r \bullet elems(tt) = \{c\} \wedge EQVTRC(tt, (t, r)) \wedge sl' \setminus\!\setminus sl = map(shide(\{c\}))(\langle (t, r) \rangle) \\
\equiv & \quad \text{“ [Acc:h:eq:elems:ET]:p11 ”} \\
 & \exists\, t, r \bullet acc(t) = \{c\} \wedge sl' \setminus\!\setminus sl = map(shide(\{c\}))(\langle (t, r) \rangle) \\
\equiv & \quad \text{“ } map \text{ def ”} \\
 & \exists\, t, r \bullet acc(t) = \{c\} \wedge sl' \setminus\!\setminus sl = \langle shide(\{c\})(t, r) \rangle \\
\equiv & \quad \text{“ } sl' \setminus\!\setminus sl = \langle (tl, rl) \rangle \text{ ”} \\
 & \exists\, t, r, tl, rl \bullet acc(t) = \{c\} \wedge (tl, rl) = shide(\{c\})(t, r) \wedge sl' \setminus\!\setminus sl = \langle (tl, rl) \rangle \\
\equiv & \quad \text{“ [SHid:def]:p15 ”} \\
 & \exists\, t, tl, rl \bullet acc(t) = \{c\} \wedge tl = shide_H\{c\}(t) \wedge sl' \setminus\!\setminus sl = \langle (tl, rl) \rangle \\
\equiv & \quad \text{“ [hide:it:is:null]:p15 ”} \\
 & \exists\, tl, rl \bullet acc(tl) = \emptyset \wedge sl' \setminus\!\setminus sl = \langle (tl, rl) \rangle \\
\equiv & \quad \text{“ [HN:null]:p17 ”} \\
 & \exists\, tl, rl \bullet tl = hnull \wedge sl' \setminus\!\setminus sl = \langle (tl, rl) \rangle \\
\equiv & \quad \text{“ [SN:def]:p12 ”} \\
 & \exists\, rl \bullet sl' \setminus\!\setminus sl = \langle snull(rl) \rangle \\
\equiv & \quad \text{“ [DF:Null:equal]:p22 ”} \\
 & \exists\, rl \bullet sl' \cong sl \wedge EQVREF(sl') = rl \\
\equiv & \quad \text{“ One point rule ”} \\
 & sl' \cong sl
\end{array}
$$

B Slotted-*Circus* Law Proofs

B.1 Prefix

B.1.1 Proof

[prefixLaw-1] : $p35$ $\quad (c.e \rightarrow Skip) \wedge wait' \equiv \mathbf{CSP1}(ok' \wedge \mathbf{R}(WTC(c))) \wedge wait'$

$$
\begin{array}{ll}
& (c.e \rightarrow Skip) \wedge wait' \\
\equiv & \quad \text{“ [Comm:def]:p33 ”} \\
& \mathbf{CSP1}\left(ok' \wedge \mathbf{R3}\left(WTC(c) \lhd wait' \rhd \left(\begin{array}{l} state' = state \wedge \\ WTC(c);\ TRMC(c) \end{array}\right)\right)\right) \wedge wait' \\
\equiv & \quad \text{“ [CSP1:def]:p27,[R3:def]:p24 ”} \\
& \left(DIV \vee ok' \wedge I\!I_R \lhd wait \rhd \left(WTC(c) \lhd wait' \rhd \left(\begin{array}{l} state' = state \wedge \\ WTC(c);\ TRMC(c) \end{array}\right)\right)\right) \wedge wait' \\
\equiv & \quad \text{“ [Cond:def]:p32 ”} \\
& (DIV \vee ok' \wedge I\!I_R \lhd wait \rhd (WTC(c) \wedge wait')) \wedge wait' \\
\equiv & \quad \text{“ [Cond:def]:p32 ”} \\
& (DIV \vee ok' \wedge I\!I_R \lhd wait \rhd (WTC(c))) \wedge wait' \\
\equiv & \quad \text{“ [CSP1:def]:p27,[R3:def]:p24 ”} \\
& \mathbf{CSP1}\,(ok' \wedge \mathbf{R3}\,(WTC(c))) \wedge wait'
\end{array}
$$

B.1.2 Proof

[prefixLaw-2] : $p35$ $\quad (c.e \rightarrow Skip) \wedge \neg\, wait' \equiv \mathbf{CSP1}\left(ok' \wedge \mathbf{R3}\left(\begin{array}{l} state' = state \wedge \\ WTC(c);\ TRMC(c) \end{array}\right)\right) \wedge \neg wait'$

$(c.e \rightarrow Skip) \wedge \neg\, wait'$

$\equiv$ " [Comm:def]:p33 "

$\mathbf{CSP1}\left(ok' \wedge \mathbf{R3}\left(WTC(c) \lhd wait' \rhd \left(\begin{array}{l} state' = state \wedge \\ WTC(c);\ TRMC(c) \end{array}\right)\right)\right) \wedge \neg wait'$

$\equiv$ " [CSP1:def]:p27,[R3:def]:p24 "

$\left(DIV \vee ok' \wedge I\!I_R \lhd wait \rhd \left(WTC(c) \lhd wait' \rhd \left(\begin{array}{l} state' = state \wedge \\ WTC(c);\ TRMC(c) \end{array}\right)\right)\right) \wedge \neg wait'$

$\equiv$ " [Cond:def]:p32 "

$(DIV \vee ok' \wedge I\!I_R \lhd wait \rhd (state' = state \wedge WTC(c);\ TRMC(c) \wedge \neg wait')) \wedge \neg wait'$

$\equiv$ " [Cond:def]:p32 "

$(DIV \vee ok' \wedge I\!I_R \lhd wait \rhd (state' = state \wedge WTC(c);\ TRMC(c))) \wedge \neg wait'$

$\equiv$ " [CSP1:def]:p27,[R3:def]:p24 "

$\mathbf{CSP1}\,(ok' \wedge \mathbf{R3}\,(state' = state \wedge WTC(c);\ TRMC(c))) \wedge \neg wait'$

B.1.3 Proof

Given healthy P:

$$[\text{prefixLaw-3}] : p35 \qquad \begin{array}{l} (c.e \rightarrow P) \wedge NOEVTS(slots, slots') \\ \equiv \qquad \text{for healthy } P \\ \mathbf{CSP1}(ok' \wedge \mathbf{R3}(WTC(c) \wedge wait')) \wedge NOEVTS(slots, slots') \end{array}$$

Proof:

$$(c.e \rightarrow P) \wedge NOEVTS(slots, slots')$$

$\equiv$ " [Pfx:def]:p33 "

$$((c.e \rightarrow Skip);\ P) \wedge wait' \wedge NOEVTS(slots, slots')$$

$\equiv$ " [Comm:def]:p33 "

$$\left(\mathbf{CSP1}\left(ok' \wedge \mathbf{R3}\left(WTC(c) \lhd wait' \rhd \left(\begin{array}{l} state' = state \wedge \\ WTC(c);\ TRMC(c) \end{array}\right)\right)\right);\ P\right)$$
$$\wedge\ NOEVTS(slots, slots')$$

$\equiv$ " property of NOEVENTS and seq comp "

$$\left(\mathbf{CSP1}\left(ok' \wedge \mathbf{R3}\left(\begin{array}{l} WTC(c) \\ \lhd wait' \rhd \\ \left(\begin{array}{l} state' = state \wedge \\ WTC(c);\ TRMC(c) \end{array}\right) \end{array}\right)\right) \wedge NOEVTS(slots, slots');\ P\right)$$
$$\wedge\ NOEVTS(slots, slots')$$

$\equiv$ " Logic "

$$\left(\mathbf{CSP1}\left(ok' \wedge \mathbf{R3}\left(\left(\begin{array}{l} WTC(c) \\ \lhd wait' \rhd \\ \left(\begin{array}{l} state' = state \wedge \\ WTC(c);\ TRMC(c) \end{array}\right) \end{array}\right) \wedge NOEVTS(slots, slots')\right)\right);\ P\right)$$
$$\wedge\ NOEVTS(slots, slots')$$

$\equiv$ " Logic "

$$\left(\mathbf{CSP1}\left(ok' \wedge \mathbf{R3}\left(\begin{array}{l} WTC(c) \wedge NOEVTS(slots, slots') \\ \lhd wait' \rhd \\ \left(\begin{array}{l} state' = state \wedge \\ WTC(c);\ TRMC(c) \end{array}\right) \wedge NOEVTS(slots, slots') \end{array}\right)\right);\ P\right)$$
$$\wedge\ NOEVTS(slots, slots')$$

$\equiv$ " $TRMC \wedge NOEVENTS = False$ "

$$\left(\mathbf{CSP1}\left(ok' \wedge \mathbf{R3}\left(\begin{array}{l} WTC(c) \wedge NOEVTS(slots, slots') \\ \wedge\ wait' \end{array}\right)\right);\ P\right)$$
$$\wedge\ NOEVTS(slots, slots')$$

$\equiv$ " P is R3 "

$$\left(\mathbf{CSP1}\left(ok' \wedge \mathbf{R3}\left(\begin{array}{l} WTC(c) \wedge NOEVTS(slots, slots') \\ \wedge\ wait' \end{array}\right)\right);\ I\!I_R\right) \wedge NOEVTS(slots, slots')$$

$\equiv$ " Property of $I\!I_R$ "

$$\mathbf{CSP1}\left(ok' \wedge \mathbf{R3}\left(\ WTC(c) \wedge wait'\ \right)\right) \wedge NOEVTS(slots, slots')$$

B.1.4 Lemma

The three proofs above also require the following lemmas:

[lemma:prefixLawa] $P \wedge Q \equiv R \wedge \mathit{slots}, \mathit{slots}'$ not free in Q
$\Rightarrow$
$\mathbf{R}(P) \wedge Q \equiv \mathbf{R}(R) \wedge Q$

Note that $P \wedge Q \equiv R$ implies $P \wedge Q \equiv R \wedge Q$.
Proof, assume antecedent to show:

$$
\begin{array}{ll}
 & \mathbf{R}(P) \wedge Q \\
\equiv & \quad \text{“ [R:def]:p25 ”} \\
 & \mathbf{R1}(\mathbf{R2}(\mathbf{R3}(P))) \wedge Q \\
\equiv & \quad \text{“ [R1:distr:and]:p23 ”} \\
 & \mathbf{R1}(\mathbf{R2}(\mathbf{R3}(P)) \wedge Q) \\
\equiv & \quad \text{“ [R2:distr:and]:p24 ”} \\
 & \mathbf{R1}(\mathbf{R2}(\mathbf{R3}(P) \wedge Q)) \\
\equiv & \quad \text{“ [R3:def]:p24 ”} \\
 & \mathbf{R1}(\mathbf{R2}((\mathit{I\!I}_R \lhd \mathit{wait} \rhd P) \wedge Q)) \\
\equiv & \quad \text{“ } \wedge \text{ distributes through } \lhd\rhd \text{ ”} \\
 & \mathbf{R1}(\mathbf{R2}(\mathit{I\!I}_R \wedge Q \lhd \mathit{wait} \rhd P \wedge Q)) \\
\equiv & \quad \text{“ Assumption, with note ”} \\
 & \mathbf{R1}(\mathbf{R2}(\mathit{I\!I}_R \wedge Q \lhd \mathit{wait} \rhd R \wedge Q)) \\
\equiv & \quad \text{“ reverse 1st five proof steps ”} \\
 & \mathbf{R}(R) \wedge Q \\
\Box &
\end{array}
$$

[lemma:prefixLawb] $P \wedge Q \equiv R \Rightarrow \mathbf{CSP1}(P) \wedge Q \equiv \mathbf{CSP1}(R) \wedge Q$

Proof, assume antecedent to show:

$$
\begin{array}{ll}
 & \mathbf{CSP1}(P) \wedge Q \\
\equiv & \quad \text{“ [CSP1:def]:p27 ”} \\
 & (P \vee \mathit{DIV}) \wedge Q \\
\equiv & \quad \text{“ } \wedge\text{–}\vee \text{ distr. ”} \\
 & P \wedge Q \vee \mathit{DIV} \wedge Q \\
\equiv & \quad \text{“ Assumption, with note ”} \\
 & R \wedge Q \vee \mathit{DIV} \wedge Q \\
\equiv & \quad \text{“ reverse 1st two proof steps ”} \\
 & \mathbf{CSP1}(R) \wedge Q \\
\Box &
\end{array}
$$

B.1.5 Proof

$$WTC(c);\ TRMC(c) \land c \in \bigcap srefs(slots' \setminus\!\!\setminus slots)$$
$$\equiv$$
$$TRMC(c) \land c \in \bigcap srefs(slots' \setminus\!\!\setminus slots)$$

Proof

$$
\begin{aligned}
& WTC(c);\ TRMC(c) \land c \in \bigcap srefs(slots' \setminus\!\!\setminus slots) \\
\equiv \quad & \text{“ [TRMC:def]:p33 ”} \\
& (WTC(c);\ EVTSNOW\{c\}(slots, slots')) \\
& \land\ c \in \bigcap srefs(slots' \setminus\!\!\setminus slots) \\
\equiv \quad & \text{“ [WTC:def]:p33 ”} \\
& (NOEVTS(slots, slots') \land c \notin \bigcup srefs(slots' \setminus\!\!\setminus slots)) \\
& ;\ EVTSNOW\{c\}(slots, slots') \\
& \land\ c \in \bigcap srefs(slots' \setminus\!\!\setminus slots) \\
\equiv \quad & \text{“ [Seq:def]:p32 ”} \\
& \exists\, slots_0 \bullet \\
& NOEVTS(slots, slots_0) \land c \notin \bigcup srefs(slots_0 \setminus\!\!\setminus slots) \\
& \land\ EVTSNOW\{c\}(slots_0, slots') \\
& \land\ c \in \bigcap srefs(slots' \setminus\!\!\setminus slots) \\
\equiv \quad & \text{“ case split ”} \\
& \#slots = \#slots' \land \exists\, slots_0 \bullet \\
& NOEVTS(slots, slots_0) \land c \notin \bigcup srefs(slots_0 \setminus\!\!\setminus slots) \\
& \land\ EVTSNOW\{c\}(slots_0, slots') \\
& \land\ c \in \bigcap srefs(slots' \setminus\!\!\setminus slots) \\
& \lor\ \#slots \neq \#slots' \land \exists\, slots_0 \bullet \\
& NOEVTS(slots, slots_0) \land c \notin \bigcup srefs(slots_0 \setminus\!\!\setminus slots) \\
& \land\ EVTSNOW\{c\}(slots_0, slots') \\
& \land\ c \in \bigcap srefs(slots' \setminus\!\!\setminus slots) \\
\equiv \quad & \text{“ case1 ”}
\end{aligned}
$$

$\equiv$ " case1 "

$TRMC(c) \land c \in \bigcap srefs(slots' \setminus\setminus slots)$

$\lor \#slots \neq \#slots' \land \exists slots_0 \bullet$

$NOEVTS(slots, slots_0) \land c \notin \bigcup srefs(slots_0 \setminus\setminus slots)$

$\land EVTSNOW\{c\}(slots_0, slots')$

$\land c \in \bigcap srefs(slots' \setminus\setminus slots)$

$\equiv$ " case2 "

$TRMC(c) \land c \in \bigcap srefs(slots' \setminus\setminus slots)$

$\lor false$

$\equiv$ " logic "

$TRMC(c) \land c \in \bigcap srefs(slots' \setminus\setminus slots)$

Case1

$\#slots = \#slots' \land \exists\, slots_0 \bullet$
$NOEVTS(slots, slots_0) \land c \notin \bigcup srefs(slots_0 \setminus\!\setminus slots)$
$\land\, EVTSNOW\{c\}(slots_0, slots')$
$\land\, c \in \bigcap srefs(slots' \setminus\!\setminus slots)$

$\equiv$ " [NEV:is:R1:R2]:p30, erefEVN:is:R1:R2, arithmetic "

$\exists\, slots_0 \bullet$
$NOEVTS(slots, slots_0) \land c \notin \bigcup srefs(slots_0 \setminus\!\setminus slots)$
$\land\, EVTSNOW\{c\}(slots_0, slots')$
$\land\, c \in \bigcap srefs(slots' \setminus\!\setminus slots)$
$\land\, \#slots_0 = \#slots' \land \#slots = \#slots_0$

$\equiv$ " [specificALaw-1]:p31, [specificALaw-3]:p31 "

$\exists\, slots_0 \bullet$
$c \notin \bigcup srefs(slots_0 \setminus\!\setminus slots)$
$\land\, EVTSNOW\{c\}(slots_0, slots')$
$\land\, c \in \bigcap srefs(slots' \setminus\!\setminus slots)$
$\land\, slots \cong slots_0 \land \#slots = \#slots_0$

$\equiv$ " property of $\setminus\!\setminus$??? "

$\exists\, slots_0 \bullet$
$c \notin sref(last(slots_0))$
$\land\, EVTSNOW\{c\}(slots_0, slots')$
$\land\, c \in \bigcap srefs(slots' \setminus\!\setminus slots)$
$\land\, slots \cong slots_0 \land \#slots = \#slots_0$

$\equiv$ " [EVN:def]:p30 "

$\exists\, slots_0, tt \bullet$
$c \notin sref(last(slots_0))$
$\land\, elems(tt) = \{c.e\} \land EQVTRACE(tt, slots' \setminus\!\setminus slots_0) \land \#slots' = \#slots_0$
$\land\, slots \cong slots_0 \land \#slots = \#slots_0$
$\land\, c \in \bigcap srefs(slots' \setminus\!\setminus slots)$

$\equiv$ " [proof:lemma:prefixLawb]:p185 "

$\exists\, slots_0, tt \bullet$
$c \notin sref(last(slots_0))$
$\land\, elems(tt) = \{c.e\} \land EQVTRACE(tt, slots' \setminus\!\setminus slots) \land \#slots' = \#slots_0$
$\land\, slots \cong slots_0 \land \#slots = \#slots_0$
$\land\, c \in \bigcap srefs(slots' \setminus\!\setminus slots)$

$\equiv$ " arithmetics "

$\exists\, slots_0, tt \bullet$
$c \notin sref(last(slots_0))$
$\land\, elems(tt) = \{c.e\} \land EQVTRACE(tt, slots' \setminus\!\setminus slots) \land \#slots' = \#slots$
$\land\, slots \cong slots_0 \land \#slots = \#slots_0$
$\land\, c \in \bigcap srefs(slots' \setminus\!\setminus slots)$

$\equiv$ " [EVN:def]:p30 "

$\equiv$ “ [EVN:def]:p30 ”

$\exists\, slots_0 \bullet$

$c \notin sref(last(slots_0))$

$EVTSNOW\{c\}(slots, slots')$

$\wedge\; slots \cong slots_0 \wedge \#slots = \#slots_0$

$\wedge\; c \in \bigcap srefs(slots' \setminus\setminus slots)$

$\equiv$ “ [SSEQV:len]:p**??**????, [SSEQV:expand]:p20, [pfx:ignores:ref:2]:p12, seq def ???? ”

$EVTSNOW\{c\}(slots, slots')$

$\wedge\; c \in \bigcap srefs(slots' \setminus\setminus slots)$

$\equiv$ “ [TRMC:def]:p33 ”

$TRMC(c) \wedge c \in \bigcap srefs(slots' \setminus\setminus slots)$

Case2

$$\#slots \neq \#slots' \land \exists\, slots_0 \bullet$$
$$NOEVTS(slots, slots_0) \land c \notin \bigcup srefs(slots_0 \setminus\!\!\setminus slots)$$
$$\land\ EVTSNOW\{c\}(slots_0, slots')$$
$$\land\ c \in \bigcap srefs(slots' \setminus\!\!\setminus slots)$$

$\equiv$ “ [NEV:len]:p??, [EVN:def]:p30 ”

$$\#slots \neq \#slots' \land \exists\, slots_0 \bullet$$
$$NOEVTS(slots, slots_0) \land \#slots \leq \#slots_0 \land c \notin \bigcup srefs(slots_0 \setminus\!\!\setminus slots)$$
$$\land\ EVTSNOW\{c\}(slots_0, slots') \land \#slots_0 = \#slots'$$
$$\land\ c \in \bigcap srefs(slots' \setminus\!\!\setminus slots)$$

$\equiv$ “ [DF:len]:p22,arithmetics ”

$$\exists\, slots_0 \bullet$$
$$NOEVTS(slots, slots_0) \land \#(slots_0 \setminus\!\!\setminus slots) > 1 \land c \notin \bigcup srefs(slots_0 \setminus\!\!\setminus slots)$$
$$\land\ EVTSNOW\{c\}(slots_0, slots') \land \#slots_0 = \#slots'$$
$$\land\ c \in \bigcap srefs(slots' \setminus\!\!\setminus slots)$$

$\equiv$ “ $slots_0 \setminus\!\!\setminus slots$ has more then one slot ”

$$\exists\, slots_0, t, r, sl, s \bullet$$
$$NOEVTS(slots, slots_0) \land \#(slots_0 \setminus\!\!\setminus slots) > 1 \land c \notin \bigcup srefs(slots_0 \setminus\!\!\setminus slots)$$
$$\land\ EVTSNOW\{c\}(slots_0, slots') \land \#slots_0 = \#slots'$$
$$\land\ c \in \bigcap srefs(slots' \setminus\!\!\setminus slots)$$
$$\land\ slots_0 \setminus\!\!\setminus slots = \langle (t, r) \rangle \frown sl \frown \langle s \rangle$$

$\equiv$ “ set theory ”

$$\exists\, slots_0, t, r, sl, s \bullet$$
$$NOEVTS(slots, slots_0) \land \#(slots_0 \setminus\!\!\setminus slots) > 1 \land c \notin \bigcup srefs(slots_0 \setminus\!\!\setminus slots)$$
$$\land\ EVTSNOW\{c\}(slots_0, slots') \land \#slots_0 = \#slots'$$
$$\land\ c \in \bigcap srefs(slots' \setminus\!\!\setminus slots)$$
$$\land\ slots_0 \setminus\!\!\setminus slots = \langle (t, r) \rangle \frown sl \frown \langle s \rangle \land c \notin Ref(t, r)$$

$\equiv$ “ [EVN:is:R1:R2]:p30 ”

$$\exists\, slots_0, t, r, sl, s \bullet$$
$$NOEVTS(slots, slots_0) \land \#(slots_0 \setminus\!\!\setminus slots) > 1 \land c \notin \bigcup srefs(slots_0 \setminus\!\!\setminus slots)$$
$$\land\ EVTSNOW\{c\}(slots_0, slots') \land \#slots_0 = \#slots' \land slots_0 \preccurlyeq slots'$$
$$\land\ c \in \bigcap srefs(slots' \setminus\!\!\setminus slots)$$
$$\land\ slots_0 \setminus\!\!\setminus slots = \langle (t, r) \rangle \frown sl \frown \langle s \rangle \land c \notin Ref(t, r)$$

$\Rightarrow$

$$
\begin{array}{ll}
\Rightarrow & \\
& \exists\, slots_0, t, r, sl, s \bullet \\
& \land\ slots_0 \preccurlyeq slots' \\
& \land\ slots_0 \setminus\!\!\setminus slots = \langle (t, r) \rangle \frown sl \frown \langle s \rangle \land c \notin Ref(t, r) \\
& \land\ c \in \bigcap srefs(slots' \setminus\!\!\setminus slots) \\
\Rightarrow & \quad \textit{``properties of slot substraction and ordering'} \\
& \exists\, slots_0, t, r, sl, s \bullet \\
& \langle (t, r) \rangle \frown sl \leq slots' \setminus\!\!\setminus slots \land c \notin Ref(t, r) \\
& \land\ c \in \bigcap srefs(slots' \setminus\!\!\setminus slots) \\
\Rightarrow & \quad \textit{``set theory, srefs def''} \\
& \exists\, slots_0, t, r, sl, s \bullet \\
& \langle (t, r) \rangle \frown sl \leq slots' \setminus\!\!\setminus slots \land c \notin Ref(t, r) \\
& \land\ c \in \bigcap srefs(slots' \setminus\!\!\setminus slots) \land c \in Ref(t, r) \\
\equiv & \quad \text{`` Logic ''} \\
& False
\end{array}
$$

B.1.6 Lemma

$$
\begin{array}{l}
slots_a \preccurlyeq slots_c \land slots_b \preccurlyeq slots_c \Rightarrow \\
\quad ((dif(slots_c, slots_a) = dif(slots_c, slots_b)) \equiv (slots_a \cong slots_b))
\end{array}
$$

B.2 Sequential Composition

B.2.1 Proof

of [seqLaw-1]:p36

$$STOP;\ A = STOP, \qquad A \text{ healthy}$$

Proof sketch by Pawel:

$STOP;\ A$

$\equiv$ “ A is **CSP1** and **R3** healthy ”

$STOP;\ \mathbf{CSP1}(\mathbf{R3}(A))$

$\equiv$ “ [Stop:def]:p32 ”

$\mathbf{CSP1}(\mathbf{R3}(ok' \land wait' \land NOEVTS(slots, slots')));\ \mathbf{CSP1}(\mathbf{R3}(A))$

$\equiv$ “ [comp:CSP1:closed]:p27,[comp:R3:closed]:p25 ”

$\mathbf{CSP1}(\mathbf{R3}((ok' \land wait' \land NOEVTS(slots, slots'));\ A))$

$\equiv$ “ [Seq:def]:p32 ”

$\mathbf{CSP1}(\mathbf{R3}(\exists\, obs_0 \bullet ok_0 \land wait_0 \land NOEVTS(slots, slots_0) \land A[seq]))$

$\equiv$ “ A is **R3** ”

$\mathbf{CSP1}(\mathbf{R3}(\exists\, obs_0 \bullet ok_0 \land wait_0 \land NOEVTS(slots, slots_0) \land (I\!I_R[seq] \lhd wait_0 \rhd A[seq])))$

$\equiv$ “ $c \land (A \lhd c \rhd B) \equiv c \land A$, [IIr:def]:p24 ”

$\mathbf{CSP1}(\mathbf{R3}(\exists\, obs_0 \bullet ok_0 \land wait_0 \land NOEVTS(slots, slots_0) \land (DIV[seq] \lor ok' \land RSTET[seq])))$

$\equiv$ “ $ok_0 \land DIV[seq] \equiv \textsc{False}$ ”

$\mathbf{CSP1}(\mathbf{R3}(\exists\, obs_0 \bullet ok_0 \land wait_0 \land NOEVTS(slots, slots_0) \land ok' \land RSTET[seq]))$

$\equiv$ “ [Seq:subs]:p32 ”

$\mathbf{CSP1}(\mathbf{R3}(\exists\, obs_0 \bullet ok_0 \land wait_0 \land NOEVTS(slots, slots_0)$
$\quad \land ok' \land wait_0 = wait' \land slots_0 = slots'))$

$\equiv$ “ one-point ”

$\mathbf{CSP1}(\mathbf{R3}(\exists\, ok_0 \bullet ok_0 \land ok' \land wait' \land NOEVTS(slots, slots')))$

$\equiv$ “ $\exists\, b \bullet b \equiv \textsc{True}$ ”

$\mathbf{CSP1}(\mathbf{R3}(ok' \land wait' \land NOEVTS(slots, slots')))$

$\equiv$ “ [Stop:def]:p32 backwards ”

$STOP$

□

B.2.2 Lemma

[Lemma3.6-1-1]:p187

$$[\mathsf{Lemma3.6\text{-}1\text{-}1}] \qquad DIV\,;\,(wait \land I\!I_R) \equiv DIV$$

$DIV\,;\,(wait \land I\!I_R)$

$\equiv$ " [Seq:def]:p32 "

$\exists\, obs_0 \bullet DIV[obs_0/obs'] \land wait_0 \land I\!I_R[obs_0/obs]$

$\equiv$ " [IIr:def]:p24 "

$\exists\, obs_0 \bullet DIV[obs_0/obs'] \land wait_0 \land (DIV[obs_0/obs] \lor ok' \land RSTET[obs_0/obs])$

$\equiv$ " distributivity "

$\exists\, obs_0 \bullet DIV[obs_0/obs'] \land wait_0 \land DIV[obs_0/obs] \lor$
$\quad DIV[obs_0/obs'] \land wait_0 \land ok' \land RSTET[obs_0/obs]$

$\equiv$ " [DIV:def]:p24, substitution, distr. quantifier "

$(\exists\, obs_0 \bullet \neg\, ok \land slots \preccurlyeq slots_0 \land wait_0 \land \neg\, ok \land slots_0 \preccurlyeq slots') \lor$
$(\exists\, obs_0 \bullet \neg\, ok \land slots \preccurlyeq slots_0 \land wait_0 \land ok' \land RSTET[obs_0/obs])$

$\equiv$ " Line 1: [EX:trans]:p19, shrink quantifier; Line 2: [one-point:RSTET]:p25 "

$\neg\, ok \land slots \preccurlyeq slots' \land (\exists\, obs_0 \bullet slots \preccurlyeq slots_0 \land wait_0 \land slots_0 \preccurlyeq slots') \lor$
$(\exists\, ok_0 \bullet \neg\, ok \land slots \preccurlyeq slots' \land wait' \land ok')$

$\equiv$ " Line 1: witness $wait_0 = \textsc{True}$, $slots_0 = slots$ Line 2: drop quantifier "

$\neg\, ok \land slots \preccurlyeq slots' \land slots \preccurlyeq slots' \lor$
$\neg\, ok \land slots \preccurlyeq slots' \land wait' \land ok'$

$\equiv$ " [DIV:def]:p24 backwards "

$DIV \lor DIV \land wait' \land ok'$

$\equiv$ " absorbtion "

DIV

$\square$

B.2.3 Proof

of [seqLaw-2]:p36

$$((x := e);\ (x := f(x)) \equiv x := f(e)$$

Proof:

$$
\begin{array}{ll}
 & (x := e);\ (x := f(x) \\
\equiv & \text{“ Assignment is } \mathbf{CSP1}, \mathbf{R3} \text{ (see [Asg:def]:p33) ”} \\
 & \mathbf{CSP1}(\mathbf{R3}(x := e));\ \mathbf{CSP1}(\mathbf{R3}(x := f(x))) \\
\equiv & \text{“ [comp:CSP1:closed]:p27,[comp:R3:closed]:p25 ”} \\
 & \mathbf{CSP1}(\mathbf{R3}(x := e;\ x := f(x))) \\
\equiv & \text{“ [Asg:def]:p33,[Seq:def]:p32 ”} \\
 & \mathbf{CSP1}(\mathbf{R3}(\exists\, obs_0 \bullet ok = ok_0 \land ok_0 = ok' \\
 & \qquad wait = wait_0 \land wait_0 = wait' \\
 & \qquad slots = slots_0 \land slots_0 = slots' \\
 & \qquad state_0 = state \oplus \{x \mapsto val(e, state)\} \\
 & \qquad state' = state_0 \oplus \{x \mapsto val(f(x), state_0)\})) \\
\equiv & \text{“ one-point, all except } state_0 \text{ ”} \\
 & \mathbf{CSP1}(\mathbf{R3}(\exists\, state_0 \bullet ok = ok' \land wait = wait' \land slots = slots' \\
 & \qquad state_0 = state \oplus \{x \mapsto val(e, state)\} \\
 & \qquad state' = state_0 \oplus \{x \mapsto val(f(x), state_0)\})) \\
\equiv & \text{“ } state_0(x) = e,\ val(f(x), state_0) = val(f(e), state) \text{ ”} \\
 & \mathbf{CSP1}(\mathbf{R3}(\exists\, state_0 \bullet ok = ok' \land wait = wait' \land slots = slots' \\
 & \qquad state_0 = state \oplus \{x \mapsto val(e, state)\} \\
 & \qquad state' = state_0 \oplus \{x \mapsto val(f(e), state)\})) \\
\equiv & \text{“ one-point ”} \\
 & \mathbf{CSP1}(\mathbf{R3}(ok = ok' \land wait = wait' \land slots = slots' \\
 & \qquad state' = (state \oplus \{x \mapsto val(e, state)\}) \oplus \{x \mapsto val(f(e), state)\})) \\
\equiv & \text{“ map override ”} \\
 & \mathbf{CSP1}(\mathbf{R3}(ok = ok' \land wait = wait' \land slots = slots' \\
 & \qquad state' = state \oplus \{x \mapsto val(f(e), state)\})) \\
\equiv & \text{“ [Asg:def]:p33 ”} \\
 & x := f(e) \\
\square &
\end{array}
$$

B.2.4 Proof

of [seqLaw-3]:p36

$$Wait\ n;\ \ Wait\ m = Wait\ (m+n)$$

Proof sketch by Pawel:

$Wait\ n;\ \ Wait\ m$

$\equiv$ " $Wait$ is healthy, see [Wait:def]:p33 "

$\mathbf{CSP1}(\mathbf{R}(Wait\,n));\ \mathbf{CSP1}(\mathbf{R}(Wait\,m))$

$\equiv$ " [comp:CSP1:closed]:p27,[comp:R:closed]:p**??** "

$\mathbf{CSP1}(\mathbf{R}(Wait\,n;\ \ Wait\,m))$

$\equiv$ " $Wait$ is $\mathbf{R3}$-healthy "

$\mathbf{CSP1}(\mathbf{R}(Wait\,n;\ \mathbf{R3}(Wait\,m)))$

$\equiv$ " [R3:def]:p24 "

$\mathbf{CSP1}(\mathbf{R}(Wait\,n;\ I\!I_R \lhd wait \rhd Wait\,m))$

$\equiv$ " [Seq:def]:p32 "

$\mathbf{CSP1}(\mathbf{R}(\exists\, obs_0 \bullet (Wait\,n)[seq'] \wedge (I\!I_R \lhd wait \rhd Wait\,m)[seq]))$

$\equiv$ " [Wait:def]:p33 "

$\mathbf{CSP1}(\mathbf{R}(\exists\, obs_0 \bullet (ok' \wedge DELAY(n) \wedge NOEVTS(slots, slots'))[seq']$
$\wedge\ (I\!I_R \lhd wait \rhd ok' \wedge DELAY(m) \wedge NOEVTS(slots, slots'))[seq]))$

$\equiv$ " substitute "

$\mathbf{CSP1}(\mathbf{R}(\exists\, obs_0 \bullet ok_0 \wedge DELAY(n)[seq'] \wedge NOEVTS(slots, slots_0)$
$\wedge\ (I\!I_R[seq] \lhd wait_0 \rhd ok' \wedge DELAY(m)[seq] \wedge NOEVTS(slots_0, slots')))$

$\equiv$ " [Del:def]:p33, substitute "

$\mathbf{CSP1}(\mathbf{R}(\exists\, obs_0 \bullet ok_0 \wedge NOEVTS(slots, slots_0)$
$\wedge\ (DELW(n)[seq'] \lhd wait_0 \rhd DELD(n)[seq'])$
$\wedge\ (I\!I_R[seq]$
$\lhd wait_0 \rhd$
$ok' \wedge NOEVTS(slots_0, slots') \wedge (DELW(m)[seq] \lhd wait' \rhd DELD(m)[seq])$

$\equiv$ " $(A \lhd c \rhd B) \wedge (D \lhd c \rhd E) \equiv (A \wedge D) \lhd c \rhd (B \wedge E)$ "

$\mathbf{CSP1}(\mathbf{R}(\exists\, obs_0 \bullet ok_0 \wedge NOEVTS(slots, slots_0)$
$(DELW(n)[seq'] \wedge I\!I_R[seq])$
$\lhd wait_0 \rhd$
$(DELD(n)[seq'] \wedge ok' \wedge NOEVTS(slots_0, slots') \wedge (DELW(m)[seq] \lhd wait' \rhd DELD(m)[seq])$

$\equiv$ " Cond:def "

$\mathbf{CSP1}(\mathbf{R}(\exists\, obs_0 \bullet ok_0 \wedge NOEVTS(slots, slots_0) \wedge ($

(1) $wait_0 \wedge DELW(n)[seq'] \wedge I\!I[seq]$

(2) $\vee \neg\, wait_0 \wedge wait' \wedge DELD(n)[seq'] \wedge ok' \wedge NOEVTS(slots_0, slots') \wedge DELW(m)[seq]$

(3) $\vee \neg\, wait_0 \wedge \neg\, wait' \wedge DELD(n)[seq'] \wedge ok' \wedge NOEVTS(slots_0, slots') \wedge DELD(m)[seq])$

$\equiv$ " $\exists - \vee$ distr., calculations (1)–(3) below "

$\mathbf{CSP1}(\mathbf{R3}(ok' \wedge wait' \wedge NOEVTS(slots, slots') \wedge (\#slots' - \#slots < n) \vee$
$ok' \wedge wait' \wedge NOEVTS(slots, slots') \wedge \#slots' - \#slots < m + n \vee$
$ok' \wedge \neg\, wait' \wedge NOEVTS(slots, slots') \wedge \#slots' - \#slots = m + n \wedge state' = state))$

$\equiv$ " common factors "

$\mathbf{CSP1}(\mathbf{R3}(ok' \wedge NOEVTS(slots, slots') \wedge$
$(wait' \wedge (\#slots' - \#slots < n \vee \#slots' - \#slots < m + n) \vee$
$\neg\, wait' \wedge \#slots' - \#slots = m + n \wedge state' = state)\,))$

$\equiv$ " $a < n \vee a < n + m \equiv a < n + m$, [Cond:def]:p32 "

$\mathbf{CSP1}(\mathbf{R3}(ok' \wedge NOEVTS(slots, slots') \wedge$
$(\#slots' - \#slots < m + n) \lhd wait' \rhd \#slots' - \#slots = m + n \wedge state' = state)))$

$\equiv$ " [Del:def]:p33, backwards "

$\mathbf{CSP1}(\mathbf{R3}(ok' \wedge DELAY(n + m) \wedge NOEVTS(slots, slots')))$

$\equiv$ " [Wait:def]:p33, backwards "

$\equiv$ " Wait (n+m) "

$\square$

We now simplify parts (1),(2),(3), under quantification and assumption $\exists\, obs_0 \bullet ok_0 \land NOEVTS(slots, slots_0)$

$$
\begin{array}{lll}
(1) & & \exists\, obs_0 \bullet ok_0 \land NOEVTS(slots, slots_0) \land \\
& & \quad wait_0 \land DELW(n)[seq'] \land \mathit{II}_R[seq] \\
& \equiv & \quad \text{“ [DELW:def]:p33, [IIr:def]:p24, [DIV:def]:p24, substitute ”} \\
& & \exists\, obs_0 \bullet ok_0 \land NOEVTS(slots, slots_0) \land \\
& & \quad wait_0 \land (\#slots_0 - \#slots < n) \\
& & \land (\neg\, ok_0 \land GROW[seq] \lor ok' \land RSTET[seq]) \\
& \equiv & \quad \text{“ } ok_0 \text{ and } \neg\, ok_0 \text{ ”} \\
& & \exists\, obs_0 \bullet ok_0 \land NOEVTS(slots, slots_0) \land \\
& & \quad wait_0 \land (\#slots_0 - \#slots < n) \land ok' \land RSTET[seq] \\
& \equiv & \quad \text{“ one-point, } RSTET[seq] \text{ equates } x_0 = x' \text{ ”} \\
& & \exists\, ok_0 \bullet ok_0 \land NOEVTS(slots, slots') \land \\
& & \quad wait' \land (\#slots' - \#slots < n) \land ok' \\
& \equiv & \quad \text{“ shrink scope, } \exists\, b \bullet b \text{ ”} \\
& & ok' \land wait' \land NOEVTS(slots, slots') \land (\#slots' - \#slots < n) \\
& & ok' \land wait' \land NOEVTS(slots, slots') \land \#slots' - \#slots < m + n
\end{array}
$$

$$
\begin{array}{lll}
(2) & & \exists\, obs_0 \bullet ok_0 \land NOEVTS(slots, slots_0) \land \\
& & \quad \neg\, wait_0 \land wait' \land DELD(n)[seq'] \land ok' \land NOEVTS(slots_0, slots') \land DELW(m)[seq] \\
& \equiv & \quad \text{“ [DELD:def]:p33, [DELW:def]:p33, substitute ”} \\
& & \exists\, obs_0 \bullet ok_0 \land NOEVTS(slots, slots_0) \land \\
& & \quad \neg\, wait_0 \land wait' \land \#slots_0 - \#slots = n \land state_0 = state \\
& & \quad \land ok' \land NOEVTS(slots_0, slots') \land \#slots' - \#slots_0 < m \\
& \equiv & \quad \text{“ arithmetic, [NEV:trans]:p31 ”} \\
& & \exists\, obs_0 \bullet ok_0 \land NOEVTS(slots, slots_0) \land \\
& & \quad \neg\, wait_0 \land wait' \land \#slots_0 - \#slots = n \land state_0 = state \\
& & \quad \land ok' \land NOEVTS(slots_0, slots') \land \#slots' - \#slots_0 < m \\
& & \quad \land NOEVTS(slots, slots') \land \#slots' - \#slots < m + n \\
& \equiv & \quad \text{“ one-point } (state_0) \text{, shrink scopes ”} \\
& & (\exists\, ok_0, wait_0 \bullet ok_0 \land \neg\, wait_0) \land \\
& & (\exists\, slots_0 \bullet NOEVTS(slots, slots_0) \land \#slots_0 - \#slots = n \\
& & \quad \land NOEVTS(slots_0, slots') \land \#slots' - \#slots_0 < m) \land \\
& & ok' \land wait' \land NOEVTS(slots, slots') \land \#slots' - \#slots < m + n \\
& \equiv & \quad \text{“ witnesses } \textsc{True}, \textsc{False} \text{ for } ok_0, wait_0 \text{ ”} \\
& & (\exists\, slots_0 \bullet NOEVTS(slots, slots_0) \land \#slots_0 - \#slots = n \\
& & \quad \land NOEVTS(slots_0, slots') \land \#slots' - \#slots_0 < m) \land \\
& & ok' \land wait' \land NOEVTS(slots, slots') \land \#slots' - \#slots < m + n \\
& \equiv & \quad \text{“ witness } slots_0 = slots \sharp ((slots' \setminus\!\!\setminus slots)[1 \ldots n]) \text{ ”} \\
& & ok' \land wait' \land NOEVTS(slots, slots') \land \#slots' - \#slots < m + n
\end{array}
$$

(3) $\exists\, obs_0 \bullet ok_0 \land NOEVTS(slots, slots_0) \land$

$\neg\, wait_0 \land \neg\, wait' \land DELD(n)[seq'] \land ok' \land NOEVTS(slots_0, slots') \land DELD(m)[seq])$

$\equiv$ " [DELD:def]:p33, substitute "

$\exists\, obs_0 \bullet ok_0 \land NOEVTS(slots, slots_0) \land$

$\neg\, wait_0 \land \neg\, wait' \land \#slots_0 - \#slots = n \land state_0 = state$

$\land\, ok' \land NOEVTS(slots_0, slots') \land \#slots' - \#slots_0 = m \land state' = state_0$

$\equiv$ " one-point ($state_0$), arithmetic, [NEV:trans]:p31 "

$\exists\, ok_0, wait_0, slots_0 \bullet ok_0 \land NOEVTS(slots, slots_0) \land$

$\neg\, wait_0 \land \neg\, wait' \land \#slots_0 - \#slots = n$

$\land\, ok' \land NOEVTS(slots_0, slots') \land \#slots' - \#slots_0 = m$

$\land\, NOEVTS(slots, slots') \land \#slots' - \#slots = m + n \land state' = state$

$\equiv$ " shrink scopes "

$(\exists\, ok_0, wait_0 \bullet ok_0 \land \neg\, wait_0)$

$(\exists\, slots_0 \bullet NOEVTS(slots, slots_0) \land \#slots_0 - \#slots = n \land$

$NOEVTS(slots_0, slots') \land \#slots' - \#slots_0 = m)$

$ok' \land \neg\, wait' \land NOEVTS(slots, slots') \land \#slots' - \#slots = m + n \land state' = state$

$\equiv$ " witnesses $\textsc{True}, \textsc{False}$ for $ok_0, wait_0$ "

$(\exists\, slots_0 \bullet NOEVTS(slots, slots_0) \land \#slots_0 - \#slots = n \land$

$NOEVTS(slots_0, slots') \land \#slots' - \#slots_0 = m)$

$ok' \land \neg\, wait' \land NOEVTS(slots, slots') \land \#slots' - \#slots = m + n \land state' = state$

$\equiv$ " witness $slots_0 = slots \sharp ((slots' \searrow slots)[1 \ldots n])$ "

$ok' \land \neg\, wait' \land NOEVTS(slots, slots') \land \#slots' - \#slots = m + n \land state' = state$

C Synchronous CSP is not Slotted-*Circus*

This section attempts to define SCSP [Bar93] as a slotted theory, and fails, but demonstrates a fundamental difference between refusals in that theory, and their rôle in almost any other CSP-like theory, including *Circus* and Slotted-*Circus*.

We proceed at the slot level, noting that we require a disjointness invariant (not found in other slot instantiations):

[SCSP:SLOT:structure]	$slot, (acc, ref) \in \mathcal{SCSP}\ E \mathrel{\widehat=} \mathbb{P}\, E \times \mathbb{P}\, E$
[SCSP:SLOT:inv]	$inv\mathcal{SCSP}(acc, ref) \mathrel{\widehat=} acc \cap ref = \emptyset$

C.0.5 Accepted and Refused Events and Equivalent Traces

With a slot we associate the set of events accepted (Acc) as well as the possible trace equivalents ($EqvTrc$).

[SCSP:ACC:sig]	$Acc_{\mathcal{SCSP}} : \mathcal{SCSP}\ E \rightarrow \mathbb{P}\, E$
[SCSP:ACC:def]	$Acc(acc, ref) \mathrel{\widehat=} acc$
[SCSP:ET:sig]	$EqvTrc_{\mathcal{SCSP}} : E^* \leftrightarrow \mathcal{SCSP}\ E$
[SCSP:ET:elems]	$EqvTrc(tr, (acc, ref)) \Rightarrow elems(tr) = acc$

C.0.6 Null Slots

We require the notion of a null slot with no accepted events, but capable of supporting arbitrary refusals.

[SCSP:SN:sig]	$SNull_{\mathcal{SCSP}} : \mathbb{P}\, E \rightarrow \mathcal{SCSP}\ E$
[SCSP:SN:def]	$SNull(ref) \mathrel{\widehat=} (\emptyset, ref)$
[SCSP:SN:ref]	$Ref(acc, ref) = ref$
[SCSP:SN:null]	$Acc(SNull(ref)) = \emptyset$
[SCSP:SN:eq]	$SNull(r) = SNull(r') \equiv r = r'$

All laws hold.

C.0.7 Slot Prefix Relation

The relation $\preceq_{\mathcal{SCSP}}$ captures the notion of one slot being a prefix of another:

[SCSP:pfx:sig]	$\preceq_{\mathcal{SCSP}} : \mathcal{SCSP}\ E \leftrightarrow \mathcal{SCSP}\ E$
[SCSP:pfx:def]	$(acc_1, ref_1) \preceq_{\mathcal{SCSP}} (acc_2, ref_2) \mathrel{\widehat=} acc_1 \subseteq acc_2$
[SCSP:pfx:refl]	$slot \preceq slot = \textsc{True}$
[SCSP:pfx:trans]	$slot_1 \preceq slot_2 \wedge slot_2 \preceq slot_3 \Rightarrow slot_1 \preceq slot_3$
[SCSP:pfx:anti-sym]	$slot_1 \preceq slot_2 \wedge slot_2 \preceq slot_1 \Rightarrow slot_1 = slot_2$

The following properties are required:

[SCSP:SN:pfx]	$SNull(r) \preceq slot$
[SCSP:ET:pfx]	$slot_1 \preceq slot_2 \Rightarrow \exists\, tr_1, tr_2 \bullet EqvTrc(tr_1, s_1) \land EqvTrc(tr_2, s_2) \land tr_1 \leq tr_2$
[SCSP:pfx:ignores:ref:1]	$slot_1 \preceq slot_2 \land slot_2 \preceq slot_1 \not\Rightarrow Ref(slot_1) = Ref(slot_2)$
[SCSP:pfx:ignores:ref:2]	$slot_1 \preceq slot_2 \equiv \forall\, r_1, r_2 \bullet slot_1[r_1] \preceq slot_2[r_2]$

All hold.

C.0.8 Slot Addition (Concatenation)

We also need to have the notion of adding slots in a manner analogous to concatenation:

[SCSP:Sadd:sig]	$Sadd_{\mathcal{SCSP}} : \mathcal{SCSP}\ E \times \mathcal{SCSP}\ E \rightarrow \mathcal{SCSP}\ E$
[SCSP:Sadd:def]	$Sadd_{\mathcal{SCSP}}((a_1, r_1), (a_2, r_2)) \mathrel{\widehat=} (a_1 \cup a_2, r_2 \setminus a_1)$

Note that, unlike other slot instances, that we need to modify the 2nd argument refusal, in order to preserve the $\mathcal{SCSP}$ slot invariant.

We can show the following required properties,

[SCSP:Sadd:events]		$Acc(Sadd(s_1, s_2)) =? Acc(s_1) \cup Acc(s_2)$
		$Acc(Sadd((a_1, r_1), (a_2, r_2)))$
	$=$	$Acc((a_1 \cup a_2, r_2 \setminus a_1)$
	$=$	$a_1 \cup a_2 \qquad OK$
[SCSP:Sadd:assoc]		$Sadd(s_1, Sadd(s_2, s_3)) = Sadd(Sadd(s_1, s_2), s_3)$
Lhs		$Sadd((a_1, r_1), Sadd((a_2, r_2), (a_3, r_3)))$
		$Sadd((a_1, r_1), (a_2 \cup a_3, r_3 \setminus a_2))$
		$a_1 \cup a_2 \cup a_3, r_3 \setminus a_2 \setminus a_1)$
Rhs		$Sadd(Sadd((a_1, r_1), (a_2, r_2)), (a_3, r_3))$
		$Sadd((a_1 \cup a_2, r_2 \setminus a_1), (a_3, r_3))$
		$(a_1 \cup a_2 \cup a_3, r_3 \setminus (a_1 \cup a_2))$
[SCSP:Sadd:prefix]		$s \preceq Sadd(s, s')$
		$(a, r) \preceq Sadd((a, r), (a', r'))$
		$(a, r) \preceq (a \cup a', r' \setminus a)$
		$a \subseteq a \cup a'$
[SCSP:Sadd:ref]		$Ref(Sadd(s_1, s_2)) = Ref(s_2)$
		$Ref(Sadd((a_1, r_1), (a_2, r_2)))$
	$=$	$Ref((a_1 \cup a_2, r_2 \setminus a_1))$
	$=$	$r_2 \setminus a_1$
	$\neq$	$r_2 = Ref(a_2, r_2)$
[SCSP:Sadd:unit]		$Sadd(s_1, s_2) = s_1 \equiv s_2 = SNull(Ref(s_1))$
		$Sadd((a_1, r_1), (a_2, r_2)) = (a_1, r_1)$
	$\equiv$	$(a_1 \cup a_2, r_2 \setminus a_1) = (a_1, r_1)$
	$\equiv$	$a_1 \cup a_2 = a_1 \wedge r_2 \setminus a_1 = r_1$
	$\equiv$	$a_2 \subseteq a_1 \wedge r_2 \setminus a_1 = r_1$
	$\not\equiv$	$(a_2, r_2) = (\emptyset, r_1)$

We find that [Sadd:ref] is not compatible with the $\mathcal{SCSP}$ invariant, but more importantly, that we cannot guarantee [Sadd:unit], even if the invariant is not required!

There is a derived notion of slot equivalence, introduced formally later (3.3.4), and we require *SAdd* to have the following property w.r.t. to such an equivalence:

[SCSP:Sadd:eqv:unit] $\quad SAdd(s_1, s_2) \approx s_1 \;\equiv\; s_2 = SNull(Ref(s_2))$
where $slot_a \approx slot_b = slot_a \preceq slot_b \wedge slot_b \preceq slot_a$

Note that this is weaker than the unit law [Sadd:unit], but also fails, for the same reason.

Finally, we introduce the following binary shorthand for *Sadd* :

[SCSP:Sadd:binop] $\quad s_1 \sharp s_2 \mathrel{\widehat=} Sadd(s_1, s_2)$

C.0.9 Slot Subtraction

The related notion of slot-subtraction is also required:

$$\text{[SCSP:Ssub:sig]} \qquad \begin{array}{l} Ssub_{\mathcal{H}} : \mathcal{H}\ E \times \mathcal{H}\ E \nrightarrow \mathcal{H}\ E \\ Ssub_{\mathcal{SCSP}} : \mathcal{SCSP}\ E \times \mathcal{SCSP}\ E \nrightarrow \mathcal{SCSP}\ E \end{array}$$

where the second argument is subtracted from the first.
As for addition, we define slot subtraction to retain the refusal of its first argument:

$$\text{[SCSP:Ssub:def]} \qquad Ssub_{\mathcal{SCSP}}((h_1, r_1), (h_2, r_2)) \mathrel{\widehat{=}} (Ssub_{\mathcal{H}}(h_1, h_2), r_1)$$

Subtraction is partial, and needs to obey a large collection of laws:

$$\begin{array}{rl}
\text{[SCSP:Ssub:pre]} & \text{pre } Ssub(h_1, h_2) = h_2 \preceq h_1 \\
& \text{pre } Ssub(s_1, s_2) = s_2 \preceq s_1 \\
\text{[SCSP:Ssub:events]} & h_2 \preceq h_1 \land h' = Ssub(h_1, h_2) \Rightarrow \\
& \quad Acc(h_1) \setminus Acc(h_2) \subseteq Acc(h') \subseteq Acc(h_1) \\
\text{[Ssub:events]}_{\mathcal{SCSP}} & s_2 \preceq s_1 \land s' = Ssub(s_1, s_2) \Rightarrow \\
& \quad Acc(s_1) \setminus Acc(s_2) \subseteq Acc(s') \subseteq Acc(s_1) \\
\text{[SCSP:SSub:ref]} & Ref(Ssub(slot', slot)) = Ref(slot') \\
\text{[SCSP:SSub:self]} & Ssub(h, h) = HNull \\
& Ssub(s, s) = SNull(Ref(s)) \\
\text{[SCSP:SSub:nil]} & Ssub(h, HNull) = h \\
& Ssub(s, SNull(r)) = s \\
\text{[SCSP:SSub:same]} & hist \preceq hist'_a \land hist \preceq hist'_b \Rightarrow \\
& \quad Ssub(hist'_a, hist) = Ssub(hist'_b, hist) \equiv hist'_a = hist'_b \\
\text{[SSub:same]}_{\mathcal{SCSP}} & slot \preceq slot'_a \land slot \preceq slot'_b \Rightarrow \\
& \quad Ssub(slot'_a, slot) = Ssub(slot'_b, slot) \equiv slot'_a = slot'_b \\
\text{[SCSP:SSub:subsub]} & hist_c \preceq hist_a \land hist_c \preceq hist_b \land hist_b \preceq hist_a \\
& \quad \Rightarrow Ssub(Ssub(hist_a, hist_c), Ssub(hist_b, hist_c)) = Ssub(hist_a, hist_b) \\
\text{[SSub:subsub]}_{\mathcal{SCSP}} & slot_c \preceq slot_a \land slot_c \preceq slot_b \land slot_b \preceq slot_a \\
& \quad \Rightarrow Ssub(Ssub(slot_a, slot_c), Ssub(slot_b, slot_c)) = Ssub(slot_a, slot_b)
\end{array}$$

The law [Ssub:events] may seem a little weak, but in general subtracting s_2 from s_1 does not guarantee that the result will not mention events in s_2. As for *Sadd*, we need a property linking *Ssub* and slot equivalence:

$$\text{[SCSP:SSub:eqv]} \qquad \begin{array}{l} s_1 \approx s_2 \equiv Ssub(s_1, s_2) = SNull(Ref(s_1)) \\ \textbf{where } slot_a \approx slot_b = slot_a \preceq slot_b \land slot_b \preceq slot_a \end{array}$$

This law is a consequence of the anti-symmetric of $\preceq_{\mathcal{H}}$, and the laws [SSub:self] and [Ssub:def].
Finally, we introduce the following binary shorthand for *Ssub* :

$$\text{[SCSP:Sadd:binop]} \qquad s_1 \setminus s_2 \mathrel{\widehat{=}} Ssub(s_1, s_2)$$

C.0.10 Relating Addition and Subtraction

We also require addition and subtraction to satisfy the following laws, the first of which can be considered a defining feature of subtraction, and the second being required to ensure that **R2** (see [R2:def]:p23) is idempotent:

[SCSP:Sadd:Ssub] $hist \preceq hist' \Rightarrow Sadd(hist, Ssub(hist', hist)) = hist'$
$slot \preceq slot' \Rightarrow Sadd(slot, Ssub(slot', slot)) = slot'$

[SCSP:Ssub:Sadd] $Ssub(Sadd(h_1, h_2), h_1) = h_2$
$Ssub(Sadd(s_1, s_2), s_1) = s_2$

We will allow certain variants of slotted-*Circus* that fail to satisfy [Ssub:Sadd], provided a different form of the **R2** healthiness condition is used. An example of this is that case where we model the event occurrences as a set, and *Sadd* and *Ssub* correspond to set union and set difference respectively. In this case it is generally the case that:

$$(S_1 \cup S_2) \setminus S_1 \neq S_2 \qquad \text{e.g.: } (\{a\} \cup \{a\}) \setminus \{a\} = \emptyset \neq \{a\}.$$

C.0.11 Hiding Slot Events

We need to specify how to hide events in a slot:

[SCSP:SHid:sig] $SHide_{\mathcal{H}} : \mathbb{P}\,E \rightarrow \mathcal{H}\ E \rightarrow \mathcal{H}\ E$
$SHide_{\mathcal{SCSP}} : \mathbb{P}\,E \rightarrow \mathcal{SCSP}\ E \rightarrow \mathcal{SCSP}\ E$

Hiding shrinks the event-set, and enlarges the refusals:

[SCSP:SHid:def] $SHide_{\mathcal{SCSP}}(hid)(hist, ref) \mathrel{\widehat=} (SHide_{\mathcal{H}}(hid)hist, refs \cup hid)$

[SCSP:SHid:evts] $Acc(SHide(hid)(h)) = Acc(h) \setminus hid$
$Acc(SHide(hid)(s)) = Acc(s) \setminus hid$

[SCSP:SHid:refs] $Ref(SHide(hid)(s)) = Ref(s) \cup hid$

C.0.12 Slot Synchronisation

Finally, we need a function that captures the way in which two slots can synchronise on a given channel-set:

[SCSP:SNC:sig] $SSync_{\mathcal{H}} : \mathbb{P}\,E \rightarrow \mathcal{H}\ E \times \mathcal{H}\ E \rightarrow \mathbb{P}(\mathcal{H}\ E)$
$SSync_{\mathcal{SCSP}} : \mathbb{P}\,E \rightarrow \mathcal{SCSP}\ E \times \mathcal{SCSP}\ E \rightarrow \mathbb{P}(\mathcal{SCSP}\ E)$

We define the slot-version in terms of the history one as follows:

[SCSP:SNC:def] $SSync_{\mathcal{SCSP}}(cs)((hist_1, ref_1), (hist_2, ref_2)$
$\mathrel{\widehat=} SSync_{\mathcal{H}}(cs)(hist_1, hist_2) \times \{RSync(cs)(ref_1, ref_2)\}$

[SCSP:RSYN:sig] $RSync : \mathbb{P}\,E \rightarrow \mathbb{P}\,E \times \mathbb{P}\,E \rightarrow \mathbb{P}\,E$

[SCSP:RSYN:def] $RSync(cs)(r_1, r_2) \mathrel{\widehat=} ((r_1 \cup r_2) \cap cs) \cup ((r_1 \cap r_2) \setminus cs)$

[SCSP:RSYN:sym] $RSync(cs)(r_1, r_2) = RSync(cs)(r_2, r_1)$

[SCSP:RSYN:assoc] $RSync(cs)(r_1, RSync(cs)(r_2, r_3) = RSync(cs)(RSync(cs)(r_1, r_2), r_3)$

Synchronisation needs to satisfy the following:

[SCSP:SNC:sym]	$SSync(cs)(h_1, h_2) = SSync(cs)(h_2, h_1)$
	$SSync(cs)(s_1, s_2) = SSync(cs)(s_2, s_1)$
[SCSP:SNC:null]	$SSync(cs)(SNull(r_1), SNull(r_2)) = \{SNull(RSync(r_1, r_2))\}$
[SCSP:SNC:one]	$\forall h' \in SSync(cs)(h_1, HNull) \bullet Acc(h') \subseteq Acc(h_1) \setminus cs$
	$\forall r_2 \bullet \forall s' \in SSync(cs)(s_1, SNull(r_2)) \bullet Acc(s') \subseteq Acc(s_1) \setminus cs$
[SCSP:SNC:only]	$h' \in Acc(SSync(cs)(h_1, h_2)) \Rightarrow Acc(h') \subseteq Acc(h_1) \cup Acc(h_2)$
	$s' \in Acc(SSync(cs)(s_1, s_2)) \Rightarrow Acc(s') \subseteq Acc(s_1) \cup Acc(s_2)$
[SCSP:SNC:sync]	$h' \in Acc(SSync(cs)(h_1, h_2)) \Rightarrow cs \cap Acc(h') \subseteq cs \cap (Acc(h_1) \cap Acc(h_2))$
	$s' \in Acc(SSync(cs)(s_1, s_2)) \Rightarrow cs \cap Acc(s') \subseteq cs \cap (Acc(s_1) \cap Acc(s_2))$

Note that these laws are weaker than might be expected—in particular, they do not specify the difference between what happens to events common to both slots, *vis-a-vis* their membership of the synchronisation set. This aspect of behaviour depends on the specifics of a given slotted theory.

We would like an associativity principle, but in order to do that we need to handle sychronisation of one history against a set of same:

[SCSP:SNCS:sig]	$SyncSet : \mathbb{P}\, E \to \mathcal{H}\, E \to \mathbb{P}(\mathcal{H}\, E) \to \mathbb{P}(\mathcal{H}\, E)$
[SCSP:SNCS:def]	$SyncSet(cs)(h)(H) \mathrel{\widehat=} \bigcup\{SSync(cs)(h, h') \mid h' \in H\}$
[SCSP:SNC:assoc]	$SyncSet(cs)(h_1)(SSync(cs)(h_2, h_3)) = SyncSet(cs)(h_3)(SSync(cs)(h_1, h_2))$
	$SyncSet(cs)(s_1)(SSync(cs)(s_2, s_3)) = SyncSet(cs)(s_3)(SSync(cs)(s_1, s_2))$

D Slotted-*Circus* Proof Principles

D.1 Proof Principles

D.1.1 Generalised One-point

Of particular importance, because of the extensive use of $slots' \cong slots$ rather than $slots' = slots$, is the following generalisation of the one-point rule:

$$\begin{array}{ll} \text{[Gen:One-Point]} & (\exists\, p \bullet Q(p) \land p \simeq q) \equiv \exists\, b \bullet Q((\pi_1 q, b)) \\ \textbf{where} & \\ & p : P \approx A \times B \\ & \pi_1 : P \rightarrow A \\ & \pi_2 : P \rightarrow B \\ & (p \simeq q) \equiv (\pi_1 p = \pi_1 q) \end{array}$$

In other word, P is (isomorphic to) a product type, and $\simeq$ is defined as equality over a projection from that type.
Proof:

$$\begin{array}{ll} & \exists\, p \bullet Q(p) \land p \simeq q \\ \equiv & \quad \text{“ one-point rule backwards, } a : A, b : B \text{ fresh ”} \\ & \exists\, a, b, p \bullet Q(p) \land p \simeq q \land a = \pi_1 p \land b = \pi_2 p \\ \equiv & \quad \text{“ rewrite equalities (possible for products) ”} \\ & \exists\, a, b, p \bullet Q(p) \land p \simeq q \land p = (a, b) \\ \equiv & \quad \text{“ one-point rule, } p \text{ ”} \\ & \exists\, a, b \bullet Q((a, b)) \land (a, b) \simeq q \\ \equiv & \quad \text{“ defn } \simeq \text{ ”} \\ & \exists\, a, b \bullet Q((a, b)) \land a = \pi_1 q \\ \equiv & \quad \text{“ One-point rule, } a \text{ ”} \\ & \exists\, b \bullet Q((\pi_1 q, b)) \end{array}$$

In effect, we are performing a change of variables where they are related by a bijection.
We need a few bijections for slots:

$$\begin{array}{rcl} slot & : & \mathcal{S}\, E \approx \mathcal{H}\, E \times \mathbb{P}\, E \\ hist & : & \mathcal{H}\, E \\ ref & : & \mathbb{P}\, E \\ slot & = & (hist, ref) \qquad \text{— bijection} \\ slots & : & (\mathcal{S}\, E)^+ \quad \approx \quad (\mathcal{H}\, E \times \mathbb{P}\, E)^+ \\ prior & : & (\mathcal{S}\, E)^* \\ slots & = & prior \frown \langle (hist, ref) \rangle \qquad \text{— bijection} \end{array}$$

Another variant arises when the relationship isn't quite an isomorphism:

$$\begin{array}{rcl} slots & : & (\mathcal{S}\,E)^+ \quad \approx \quad (\mathcal{S}'\,E \times \mathbb{P}\,E)^+ \\ evtrc & : & (\mathcal{S}'\,E)^+ \\ refs & : & (\mathbb{P}\,E)^+ \\ slots & = & zip(evtrc, refs) \qquad \text{— not quite, requires } \#evtrc = \#refs \end{array}$$

Assume the following definitions:

$$\begin{array}{rcl} a & : & A^* \\ b & : & B^* \\ s & : & S^* = (A \times B)^* \\ unzip & : & S^* \to A^* \times B^* \\ unzip(s) & \widehat{=} & (a, b) \,\mathbf{where}\, a = (\pi_1)^* s \land b = (\pi_2)^* s \\ zip & : & A^* \times B^* \nrightarrow S^* \\ \text{pre } zip(a, b) & \widehat{=} & \#a = \#b \\ zip(a, b) & \widehat{=} & s \,\mathbf{where}\, a = (\pi_1)^* s \land b = (\pi_2)^* s \\ \textsc{True} & \equiv & \text{pre } zip(unzip(s)) \\ zip(unzip(s) & = & s \\ \text{pre } zip(a, b) & \Rightarrow & unzip(zip(a, b)) = (a, b) \end{array}$$

We explore using this relationship in a one-point style situation, where we have the following definition of $\simeq$:

$$\begin{array}{rcl} s \simeq s' & \widehat{=} & (\pi_1)^* s = (\pi_1)^* s' \\ s \simeq s' & \equiv & \pi_1(unzip(s)) = \pi_1(unzip(s')) \end{array}$$

$$\begin{array}{ll} & \exists\, s \bullet P(s) \land s \simeq s' \\ \equiv & \quad \text{“ reverse one-point, } a, b \text{ ”} \\ & \exists\, a, b, s \bullet P(s) \land s \simeq s' \land (a, b) = unzip(s) \\ \equiv & \quad \text{“ flip equality ”} \\ & \exists\, a, b, s \bullet P(s) \land s \simeq s' \land s = zip(a, b) \land \#a = \#b \\ \equiv & \quad \text{“ one-point } s \text{ ”} \\ & \exists\, a, b \bullet P(zip(a, b)) \land zip(a, b) \simeq s' \land \#a = \#b \\ \equiv & \quad \text{“ alternate defn. of } \simeq \text{ ”} \\ & \exists\, a, b \bullet P(zip(a, b)) \land \pi_1(unzip(zip(a, b)) = \pi_1(unzip(s')) \land \#a = \#b \\ \equiv & \quad \text{“ pre-zip holds ”} \\ & \exists\, a, b \bullet P(zip(a, b)) \land \pi_1(a, b) = \pi_1(unzip(s')) \land \#a = \#b \\ \equiv & \quad \text{“ projection ”} \\ & \exists\, a, b \bullet P(zip(a, b)) \land a = \pi_1(unzip(s')) \land \#a = \#b \\ \equiv & \quad \text{“ one-point } a \text{ ”} \\ & \exists\, b \bullet P(zip(\pi_1(unzip(s')), b)) \land \#\pi_1(unzip(s')) = \#b \end{array}$$

D.1.2 Change of Variable

We can also do wholesale variable changes in this manner:

$$
\begin{aligned}
& P(slots, slots') \\
\equiv \quad & \text{“ reverse one-point, introducing } prior, hist, ref, prior', hist', ref' \text{ ”} \\
& \exists\, prior, hist, ref, prior', hist', ref' \bullet \\
& \quad P(slots, slots') \\
& \wedge\ prior = front(slots) \wedge hist = last(slots).1 \wedge ref = last(slots).2 \\
& \wedge\ prior' = front(slots') \wedge hist' = last(slots').1 \wedge ref' = last(slots').2 \\
\equiv \quad & \text{“ Invert equalities ”} \\
& \exists\, prior, hist, ref, prior', hist', ref' \bullet \\
& \quad P(slots, slots') \\
& \wedge\ slots = prior \frown \langle (hist, ref) \rangle \\
& \wedge\ slots' = prior' \frown \langle (hist', ref') \rangle
\end{aligned}
$$

[Intro:prior-hist-ref]

Now consider the following variation:

$$
\begin{aligned}
& \exists\, slots_0 \bullet P(slots, slots_0, slots') \\
\equiv \quad & \text{“ reverse one-point, introducing } prior_0, hist_0, ref_0 \text{ ”} \\
& \exists\, prior_0, hist_0, ref_0, slots_0 \bullet \\
& \quad P(slots, slots_0, slots') \\
& \wedge\ prior_0 = front(slots_0) \wedge hist_0 = last(slots_0).1 \wedge ref_0 = last(slots_0).2 \\
\equiv \quad & \text{“ Invert equalities ”} \\
& \exists\, prior_0, hist_0, ref_0, slots_0 \bullet \\
& \quad P(slots, slots_0, slots') \\
& \wedge\ slots_0 = prior_0 \frown \langle (hist_0, ref_0) \rangle \\
\equiv \quad & \text{“ One-point } slots_0 \text{ ”} \\
& \exists\, prior_0, hist_0, ref_0 \bullet \\
& \quad P(slots, prior_0 \frown \langle (hist_0, ref_0) \rangle, slots')
\end{aligned}
$$

[Change:slots:prior-hist-ref]

D.1.3 Variable Change: $\preccurlyeq$

[EX:prior:hist:ref] $P \land slots \preccurlyeq slots' \quad \equiv$

$$\exists\, prior, hist, ref, prior', hist', ref' \bullet P \land$$
$$prior \le prior' \land (hist, ref) \preceq (prior' \frown \langle (hist', ref') \rangle)(\#prior + 1)$$
$$\land\ slots = prior \frown \langle (hist, ref) \rangle$$
$$\land\ slots' = prior' \frown \langle (hist', ref') \rangle$$

$P \land slots \preccurlyeq slots'$

$\equiv$ “ [Intro:prior-hist-ref]:p201 ”

$\exists\, prior, hist, ref, prior', hist', ref' \bullet$

$P \land slots \preccurlyeq slots'$

$\land\ slots = prior \frown \langle (hist, ref) \rangle$

$\land\ slots' = prior' \frown \langle (hist', ref') \rangle$

$\equiv$ “ [EX:def]:p19 ”

$\exists\, prior, hist, ref, prior', hist', ref' \bullet$

$P \land front(slots) < slots'$

$\land\ last(slots) \preceq slots'(\#slots)$

$\land\ slots = prior \frown \langle (hist, ref) \rangle$

$\land\ slots' = prior' \frown \langle (hist', ref') \rangle$

$\equiv$ “ Liebniz $slots, slots'$ ”

$\exists\, prior, hist, ref, prior', hist', ref' \bullet$

$P \land front(prior \frown \langle (hist, ref) \rangle) < prior' \frown \langle (hist', ref') \rangle$

$\land\ last(prior \frown \langle (hist, ref) \rangle) \preceq (prior' \frown \langle (hist', ref') \rangle)(\#(prior \frown \langle (hist, ref) \rangle))$

$\land\ slots = prior \frown \langle (hist, ref) \rangle$

$\land\ slots' = prior' \frown \langle (hist', ref') \rangle$

$\equiv$ “ defn $front$, $last$, $\#$, $<$ ”

$\exists\, prior, hist, ref, prior', hist', ref' \bullet$

$P \land prior \le prior'$

$\land\ (hist, ref) \preceq (prior' \frown \langle (hist', ref') \rangle)(\#prior + 1)$

$\land\ slots = prior \frown \langle (hist, ref) \rangle$

$\land\ slots' = prior' \frown \langle (hist', ref') \rangle$

D.1.4 Variable Change: $\cong$

[SSEQV:prior:hist:ref] $Q \land slots \cong slots' \quad \equiv$
$\exists\, prior, hist, ref, ref' \bullet Q \land$
$\quad slots = prior \frown \langle (hist, ref) \rangle$
$\quad \land\, slots' = prior \frown \langle (hist, ref') \rangle$

$Q \land slots \cong slots'$
$\equiv$ “ [SSEQV:def]:p20 ”
$Q \land slots \preccurlyeq slots' \land slots' \preccurlyeq slots$
$\equiv$ “ [Intro:prior-hist-ref]:p201 ”
$\exists\, prior, hist, ref, prior', hist', ref' \bullet Q \land$
$\quad slots \preccurlyeq slots' \land slots' \preccurlyeq slots$
$\quad \land\, slots = prior \frown \langle (hist, ref) \rangle \land slots' = prior' \frown \langle (hist', ref') \rangle$
$\equiv$ “ Liebniz, $slots$, $slots'$ ”
$\exists\, prior, hist, ref, prior', hist', ref' \bullet Q \land$
$\quad front(prior \frown \langle (hist, ref) \rangle) < prior' \frown \langle (hist', ref') \rangle$
$\quad \land\, last(prior \frown \langle (hist, ref) \rangle) \preceq (prior' \frown \langle (hist', ref') \rangle)(\#(prior \frown \langle (hist, ref) \rangle))$
$\quad \land\, front(prior' \frown \langle (hist', ref') \rangle) < prior \frown \langle (hist, ref) \rangle$
$\quad \land\, last(prior' \frown \langle (hist', ref') \rangle) \preceq (prior \frown \langle (hist, ref) \rangle)(\#(prior' \frown \langle (hist', ref') \rangle))$
$\quad \land\, slots = prior \frown \langle (hist, ref) \rangle \land slots' = prior' \frown \langle (hist', ref') \rangle$
$\equiv$ “ props. and defs. of $front$, $last$, $<$ and $\#$ ”
$\exists\, prior, hist, ref, prior', hist', ref' \bullet Q \land$
$\quad prior \leq prior' \land (hist, ref) \preceq (prior' \frown \langle (hist', ref') \rangle)(\#prior + 1)$
$\quad \land\, prior' \leq prior \land (hist', ref') \preceq (prior \frown \langle (hist, ref) \rangle)(\#prior' + 1)$
$\quad \land\, slots = prior \frown \langle (hist, ref) \rangle \land slots' = prior' \frown \langle (hist', ref') \rangle$
$\equiv$ “ sequence ordering is anti-symmetric ”
$\exists\, prior, hist, ref, prior', hist', ref' \bullet Q \land$
$\quad prior = prior'$
$\quad \land\, (hist, ref) \preceq (prior' \frown \langle (hist', ref') \rangle)(\#prior + 1)$
$\quad \land\, (hist', ref') \preceq (prior \frown \langle (hist, ref) \rangle)(\#prior' + 1)$
$\quad \land\, slots = prior \frown \langle (hist, ref) \rangle \land slots' = prior' \frown \langle (hist', ref') \rangle$
$\equiv$ “ one-point $prior'$ ”
$\exists\, prior, hist, ref, hist', ref' \bullet Q \land$
$\quad \land\, (hist, ref) \preceq (prior \frown \langle (hist', ref') \rangle)(\#prior + 1)$
$\quad \land\, (hist', ref') \preceq (prior \frown \langle (hist, ref) \rangle)(\#prior + 1)$
$\quad \land\, slots = prior \frown \langle (hist, ref) \rangle \land slots' = prior \frown \langle (hist', ref') \rangle$
$\equiv$ “ $(\sigma \frown \langle e \rangle)(\#\sigma + 1) = e$ ”

$$
\begin{aligned}
& \exists\, prior, hist, ref, hist', ref' \bullet Q \land \\
& \qquad (hist, ref) \preceq (hist', ref') \land (hist', ref') \preceq (hist, ref) \\
& \quad \land slots = prior \frown \langle (hist, ref) \rangle \land slots' = prior \frown \langle (hist', ref') \rangle \\
\equiv & \quad \text{“ [pfx:def]:p12 ”} \\
& \exists\, prior, hist, ref, hist', ref' \bullet Q \land \\
& \qquad hist \preceq hist' \land hist' \preceq hist \\
& \quad \land slots = prior \frown \langle (hist, ref) \rangle \land slots' = prior \frown \langle (hist', ref') \rangle \\
\equiv & \quad \text{“ [pfx:anti-sym]:p17 ”} \\
& \exists\, prior, hist, ref, hist', ref' \bullet Q \land \\
& \qquad hist = hist' \\
& \quad \land slots = prior \frown \langle (hist, ref) \rangle \land slots' = prior \frown \langle (hist', ref') \rangle \\
\equiv & \quad \text{“ One-point, } hist' \text{ ”} \\
& \exists\, prior, hist, ref, ref' \bullet Q \land \\
& \qquad slots = prior \frown \langle (hist, ref) \rangle \land slots' = prior \frown \langle (hist, ref') \rangle
\end{aligned}
$$

□

We point out that this law is more general, as the variables *slots* and *slots'* above are arbitrary. Most important of all, in the above proof, it can been seen that all the steps carry through, even if both of the slot-sequences being asserted as slot-equivalent are in fact denoted by arbitrary expressions:

[SSEQV:prior:hist:ref:E]

$$
\begin{aligned}
& Q \land E_1 \cong E_2 \quad \equiv \\
& \exists\, prior_1, hist_1, ref_1, ref_2 \bullet Q \land \\
& \qquad E_1 = prior_1 \frown \langle (hist_1, ref_1) \rangle \\
& \quad \land E_2 = prior_1 \frown \langle (hist_1, ref_2) \rangle
\end{aligned}
$$

D.1.5 Proof of [Hsub:equal]

$$[\mathsf{Hsub{:}equal}] \qquad hn \preceq h \Rightarrow (h = ssub_{\mathcal{H}}(h, hn) \equiv hn = hnull)$$

We assume $hn \preceq h$ throughout. The right-to-left implication is trivial.
We use infix notation, and concentrate on the left-to-right case, and so assume $h = h \setminus hn$

$h = h \setminus hn$

$\Rightarrow$ " left-add hn to both sides "

$hn \sharp h = hn \sharp (h \setminus hn)$

$\equiv$ " [sadd:ssub]:p17, given $hn \preceq h$ "

$hn \sharp h = h$

$\equiv$ " [sadd:unit]:p13 "

$hn = hnull$

$\square$

D.1.6 Proof of [SSub:equal]

$$sn \preceq s \Rightarrow s = ssub(s, sn) \equiv \exists\, rn \bullet sn = snull(rn)$$

Start with the lhs:

$$\begin{array}{ll}
 & s = ssub(s, sn) \\
\equiv & \quad \text{“ let } s = (h, r) \text{ and } sn = (hn, rn) \text{ ”} \\
 & (h, r) = ssub((h, r), (hn, rn)) \\
\equiv & \quad \text{“ [ssub:def]:p14 ”} \\
 & (h, r) = (ssub_{\mathcal{H}}(h, hn), r) \\
\equiv & \quad \text{“ pair equality ”} \\
 & h = ssub_{\mathcal{H}}(h, hn) \land r = r \\
\equiv & \quad \text{“ [Hsub:equal]:p205 ”} \\
 & hn = hnull
\end{array}$$

Now the rhs:

$$\begin{array}{ll}
 & \exists\, rn \bullet sn = snull(rn) \\
\equiv & \quad \text{“ [SN:def]:p12 ”} \\
 & \exists\, rn \bullet sn = (hnull, rn) \\
\equiv & \quad \text{“ let } sn = (hn, rn) \text{ ”} \\
 & \exists\, rn \bullet (hn, rn) = (hnull, rn) \\
\equiv & \quad \text{“ pair equality ”} \\
 & \exists\, rn \bullet hn = hnull \land rn = rn \\
\equiv & \quad \text{“ reflexivity of = ”} \\
 & \exists\, rn \bullet hn = hnull \\
\equiv & \quad \text{“ drop quantifier ”} \\
 & hn = hnull
\end{array}$$

□

D.1.7 Shifting

Much of this material has become obsolete since ***R2*** *was re-formulated using slots-sequence addition and subtraction ([R2:def]:p23,[Shift:obsolete]:p**??**. However many proofs done before that reformulation still rely on this material.*

In the definition of **R2**, the *Shift* function is introduced ([R2:shift]:p**??**). In some key proofs, the following expression crops up:

$$ss \frown Shift(s, slots,\ \textsc{post})$$

where POST denotes some slot-sequence expression, which has the property that $slots \preccurlyeq \textsc{post}$. We shall define a special operator $\mathcal{S}$ (Abstract Shift) that abstracts out that last argument:

$$\begin{array}{rl}
[\text{AS:sig}] & \mathcal{S} : (\mathcal{S}\ E)^+ \to (\mathcal{S}\ E)^+ \\
[\text{AS:pre}] & \text{pre}\ \mathcal{S}(post) \mathrel{\widehat=} slots \preccurlyeq post \\
[\text{AS:def}] & \mathcal{S}(post) \mathrel{\widehat=} ss \frown Shift(s, slots, post) \\
[\text{AS:expand}] & \mathcal{S}(post) = ss \frown \langle s + (slot' - slot)\rangle \frown sfx \\
\textbf{where} & slot = last(slots) \\
 & (slot' \mathbin{⨟} sfx) = post - front(slots)
\end{array}$$

Here we introduce $+$ and $-$ as shorthands for *sadd* and *ssub*.

We note first of all that the result of this gives us the following guarantee:

$$[\text{AS:post}] \qquad slots \preccurlyeq post \Rightarrow ss \frown \langle s\rangle \preccurlyeq \mathcal{S}(post)$$

More importantly, this guarantee gives the pre-condition of an inverse function:

$$\begin{array}{rl}
[\text{ASinv:sig}] & \mathcal{S}^{-1} : (\mathcal{S}\ E)^+ \to (\mathcal{S}\ E)^+ \\
[\text{ASinv:pre}] & \text{pre}\ \mathcal{S}^{-1}(shftd) \mathrel{\widehat=} ss \frown \langle s\rangle \preccurlyeq shftd \\
[\text{ASinv:post}] & ss \frown \langle s\rangle \preccurlyeq shftd \Rightarrow slots \preccurlyeq \mathcal{S}^{-1}(shftd) \\
[\text{AS:ASinv}] & slots \preccurlyeq post \Rightarrow post = \mathcal{S}^{-1}(\mathcal{S}(post)) \\
[\text{ASinv:AS}] & ss \frown \langle s\rangle \preccurlyeq shftd \Rightarrow shftd = \mathcal{S}(\mathcal{S}^{-1}(shftd))
\end{array}$$

We posit the following definition of the inverse:

$$\begin{array}{rl}
[\text{ASinv:def}] & \mathcal{S}^{-1}(shftd) \mathrel{\widehat=} front(slots) \frown Shift(last(slots), ss \frown \langle s\rangle, shftd) \\
[\text{ASinv:expand1}] & \mathcal{S}^{-1}(shftd) = front(slots) \frown \langle last(slots) + (sslot' - sslot)\rangle \frown ssfx \\
\textbf{where} & sslot = last(ss \frown \langle s\rangle) \\
 & (sslot' \mathbin{⨟} ssfx) = shftd - front(ss \frown \langle s\rangle) \\
[\text{ASinv:expand2}] & \mathcal{S}^{-1}(shftd) = front(slots) \frown \langle last(slots) + (sslot' - s)\rangle \frown ssfx \\
\textbf{where} & (sslot' \mathbin{⨟} ssfx) = shftd - ss
\end{array}$$

We also posit that both $\mathcal{S}$ and $\mathcal{S}^{-1}$ preserve the $\preccurlyeq$ relationship:

$$\begin{array}{rl}
[\text{AS:prsrv:EX}] & post_1 \preccurlyeq post_2 \Leftrightarrow \mathcal{S}(post_1) \preccurlyeq \mathcal{S}(post_2) \\
[\text{ASinv:prsrv:EX}] & shftd_1 \preccurlyeq shftd_2 \Leftrightarrow \mathcal{S}^{-1}(shftd_1) \preccurlyeq \mathcal{S}^{-1}(shftd_2)
\end{array}$$

An immediate corollary of the above is that they also preserve $\cong$:

$$\begin{array}{rl}
[\text{AS:prsrv:SSEQV}] & post_1 \cong post_2 \Leftrightarrow \mathcal{S}(post_1) \cong \mathcal{S}(post_2) \\
[\text{ASinv:prsrv:SSEQV}] & shftd_1 \cong shftd_2 \Leftrightarrow \mathcal{S}^{-1}(shftd_1) \cong \mathcal{S}^{-1}(shftd_2)
\end{array}$$

Proof of [AS:ASinv]:p207, so assume

[AS:ASinv:hyp] $\quad slots \preccurlyeq post$

to show:

$\mathcal{S}^{-1}(\mathcal{S}(post))$

$\equiv$ “ [AS:expand]:p207 ”

$\mathcal{S}^{-1}(ss \frown \langle s + (slot' - slot)\rangle \frown sfx)$

$slot = last(slots)$

$(slot' \fatsemi sfx) = post - front(slots)$

$\equiv$ “ [ASinv:expand2]:p207 ”

$front(slots) \frown \langle last(slots) + (sslot' - s)\rangle \frown ssfx$

$(sslot' \fatsemi ssfx) = (ss \frown \langle s + (slot' - slot)\rangle \frown sfx) - ss$

$slot = last(slots)$

$(slot' \fatsemi sfx) = post - front(slots)$

$\equiv$ “ law of sequence subtraction ”

$front(slots) \frown \langle last(slots) + (sslot' - s)\rangle \frown ssfx$

$(sslot' \fatsemi ssfx) = \langle s + (slot' - slot)\rangle \frown sfx$

$slot = last(slots)$

$(slot' \fatsemi sfx) = post - front(slots)$

$\equiv$ “ $\fatsemi$ pattern match, Liebniz on $sslot'$ and $ssfx$ ”

$front(slots) \frown \langle last(slots) + ((s + (slot' - slot)) - s)\rangle \frown sfx$

$slot = last(slots)$

$(slot' \fatsemi sfx) = post - front(slots)$

$\equiv$ “ $(s + t) - s = t$ ”

$front(slots) \frown \langle last(slots) + (slot' - slot)\rangle \frown sfx$

$slot = last(slots)$

$(slot' \fatsemi sfx) = post - front(slots)$

$\equiv$ “ Liebniz on $slot$, $s + (t - s) = t$, ok because $slot \preceq slot'$ by [AS:ASinv:hyp]:p208 ”

$front(slots) \frown \langle slot'\rangle \frown sfx$

$(slot' \fatsemi sfx) = post - front(slots)$

$\equiv$ “ Liebniz on $slot', sfx$, noting $\langle e\rangle \frown s = e \fatsemi s$ ”

$front(slots) \frown (post - front(slots))$

$\equiv$ “ $s + (t - s) = t$, ok because $front(slots) \preccurlyeq post$, by [AS:ASinv:hyp]:p208 ”

$post$

□

Proof of [AS:prsrv:EX]:p207, so assume

[AS:prsrv:EX:hyp1] $\quad slots \preccurlyeq post_1$

[AS:prsrv:EX:hyp2] $\quad slots \preccurlyeq post_2$

[AS:prsrv:EX:hyp3] $\quad post_1 \preccurlyeq post_2$

We can then rewrite $post_1$ and $post_2$ as:

[AS:prsrv:EX:post1]	$post_1 = pfx \frown \langle sl \rangle$
[AS:prsrv:EX:post2]	$post_1 = pfx \frown \langle sl' \rangle \frown rest$
[AS:prsrv:EX:hyp4]	$sl \preceq sl'$

We then expand out:

$$
\begin{aligned}
& \mathcal{S}(post_1) \\
= & \quad \text{“ [AS:prsrv:EX:post1]:p209 ”} \\
& \mathcal{S}(pfx \frown \langle sl \rangle) \\
= & \quad \text{“ [AS:def]:p207 ”} \\
& ss \frown \langle s + (slot' - slot) \rangle \frown sfx \\
& \quad slot = last(slots) \\
& \quad (slot' \fatsemi sfx) = (pfx \frown \langle sl \rangle) - front(slots) \\
= & \quad \text{“ Liebniz on } slot, slot', sfx \text{ ”} \\
& ss \frown \langle s + (head((pfx \frown \langle sl \rangle) - front(slots)) - last(slots)) \rangle \\
& \frown tail((pfx \frown \langle sl \rangle) - front(slots))
\end{aligned}
$$

Expanding $post_2$:

$$
\begin{aligned}
& \mathcal{S}(post_2) \\
= & \quad \text{“ [AS:prsrv:EX:post2]:p209 ”} \\
& \mathcal{S}(pfx \frown \langle sl' \rangle \frown rest) \\
= & \quad \text{“ similar to above ”} \\
& ss \frown \langle s + (head((pfx \frown \langle sl' \rangle \frown rest) - front(slots)) - last(slots)) \rangle \\
& \frown tail((pfx \frown \langle sl' \rangle \frown rest) - front(slots))
\end{aligned}
$$

We note that $head(\sigma - front(\tau)) = \sigma(\#\tau)$, so we get:

$$
\begin{aligned}
& \mathcal{S}(post_1) \\
= & \quad \text{“ above law ”} \\
& ss \frown \langle s + ((pfx \frown \langle sl \rangle)(\#slots)) - last(slots)) \rangle \\
& \frown tail((pfx \frown \langle sl \rangle) - front(slots))
\end{aligned}
$$

and

$$
\begin{aligned}
& \mathcal{S}(post_2) \\
= & \quad \text{“ above law ”} \\
& ss \frown \langle s + ((pfx \frown \langle sl' \rangle \frown rest)(\#slots)) - last(slots)) \rangle \\
& \frown tail((pfx \frown \langle sl' \rangle \frown rest) - front(slots))
\end{aligned}
$$

The sequence indexing in the $post_1$ expansion is well-defined, so $\#slots \leq \#pfx + 1$, so we can simplify $post_2$ further:

$$
\begin{array}{ll}
 & \mathcal{S}(post_2) \\
= & \quad \text{“ above law ”} \\
 & ss \frown \langle s + ((pfx \frown \langle sl' \rangle)(\#slots)) - last(slots)) \rangle \\
 & \frown tail((pfx \frown \langle sl' \rangle \frown rest) - front(slots))
\end{array}
$$

As $slots \preccurlyeq post_1 = pfx \frown \langle sl \rangle$, we know that $front(slots) < pfx$, so $(pfx \frown \sigma) - front(slots)$ is equal to $(pfx - front(slots)) \frown \sigma$. So we can substitute $less = pfx - front(slots)$ into both expansions:

$$
\begin{array}{ll}
 & \mathcal{S}(post_1) \\
= & \quad \text{“ above principle ”} \\
 & ss \frown \langle s + ((pfx \frown \langle sl \rangle)(\#slots)) - last(slots)) \rangle \\
 & \frown tail(less \frown \langle sl \rangle) \\
 & \mathcal{S}(post_2) \\
= & \quad \text{“ above principle ”} \\
 & ss \frown \langle s + ((pfx \frown \langle sl' \rangle)(\#slots)) - last(slots)) \rangle \\
 & \frown tail(less \frown \langle sl' \rangle \frown rest))
\end{array}
$$

Both are identical except for the last component $tail(\ldots)$, and it is clear that that $\mathcal{S}(post_1)$ and $\mathcal{S}(post_2)$ first differ at sl and sl', where we have $sl \preceq sl'$ by [AS:prsrv:EX:hyp4]:p209. So we can assert that

$$\mathcal{S}(post_1) \preccurlyeq \mathcal{S}(post_2)$$

□

We can also generalise $\mathcal{S}$ to work with variables other than ss and s, useful for certain proofs:

[AS:par:sig]	$\mathcal{S} : ((\mathcal{S}\,E)^+ \times \mathcal{S}\,E) \rightarrow (\mathcal{S}\,E)^+ \rightarrow (\mathcal{S}\,E)^+$
[AS:par:pre]	$\text{pre } \mathcal{S}_{xx,x}(post) \mathrel{\widehat=} slots \preccurlyeq post$
[AS:par:def]	$\mathcal{S}_{xx,x}(post) \mathrel{\widehat=} xx \frown Shift(x, slots, post)$
[AS:par:expand]	$\mathcal{S}_{xx,x}(post) = xx \frown \langle x + (slot' - slot) \rangle \frown sfx$
where	$slot = last(slots)$
	$(slot' \mathbin{⨾} sfx) = post - front(slots)$

Also we may want to change the $slots$ variable, giving yet another parameterisation:

[AS:pars:sig]	$\mathcal{S} : ((\mathcal{S}\,E)^+ \times \mathcal{S}\,E) \rightarrow ((\mathcal{S}\,E)^+ \times (\mathcal{S}\,E)^+) \rightarrow (\mathcal{S}\,E)^+$
[AS:pars:pre]	$\text{pre } \mathcal{S}_{xx,x}(post, sls) \mathrel{\widehat=} sls \preccurlyeq post$
[AS:pars:def]	$\mathcal{S}_{xx,x}(post, sls) \mathrel{\widehat=} xx \frown Shift(x, sls, post)$
[AS:pars:expand]	$\mathcal{S}_{xx,x}(post, sls) = xx \frown \langle x + (slot' - slot) \rangle \frown sfx$
where	$slot = last(sls)$
	$(slot' \mathbin{⨾} sfx) = post - front(sls)$

E Theory of Sequences

We present some theory of sequences that proves useful.

E.1 Sequence Definitions

E.1.1 Sequence Head

[Seq:Head:Sig]	$head : \Sigma^+ \to \Sigma$
[Seq:Head:def]	$head(x \fatsemi \sigma) \mathrel{\widehat{=}} x$

E.1.2 Sequence Tail

[Seq:Tail:Sig]	$tail : \Sigma^+ \to \Sigma^*$
[Seq:Tail:def]	$tail(x \fatsemi \sigma) \mathrel{\widehat{=}} \sigma$

E.1.3 Sequence Concatenation

[Seq:Cat:Sig]	$\frown : \Sigma^* \to \Sigma^*$
[Seq:Cat:def:nil]	$\langle\rangle \frown \tau \mathrel{\widehat{=}} \tau$
[Seq:Cat:def:cons]	$(x \fatsemi \sigma) \frown \tau \mathrel{\widehat{=}} x \fatsemi (\sigma \frown \tau)$

E.1.4 Sequence Length

[Seq:Len:Sig]	$\# : \Sigma^* \to \mathbb{N}$
[Seq:Len:def:nil]	$\#\langle\rangle \mathrel{\widehat{=}} 0$
[Seq:Len:def:cons]	$\#(x \fatsemi \sigma) \mathrel{\widehat{=}} 1 + \#\sigma$

E.1.5 Sequence Index

[Seq:Index:Sig]	$\_(\_) : \Sigma^+ \to \mathbb{N} \setminus 0 \nrightarrow \Sigma$
[Seq:Index:one]	$(x \fatsemi \sigma)(1) \mathrel{\widehat{=}} x$
[Seq:Index:more]	$(x \fatsemi \sigma)(n+1) \mathrel{\widehat{=}} \sigma(n)$

E.1.6 Prefix Relation

[Seq:Pfx:Sig]	$\_ \leq \_ : \Sigma^* \times \Sigma^* \to \mathbb{B}$
[Seq:Pfx:def:nil]	$\langle\rangle \leq \tau \mathrel{\widehat{=}} \textsc{True}$
[Seq:Pfx:def:cons]	$(x \mathbin{⨾} \sigma) \leq (y \mathbin{⨾} \tau) \mathrel{\widehat{=}} x = y \wedge \sigma \leq \tau$
[Seq:Pfx:def:lin]	$(x \mathbin{⨾} \sigma) \leq \langle\rangle \mathrel{\widehat{=}} \textsc{False}$

$$\frac{}{\langle\rangle \leq \tau} \quad \text{[Seq:Pfx-rel:def:nil]}$$

$$\frac{x = y \qquad \sigma \leq \tau}{x \mathbin{⨾} \sigma \leq y \mathbin{⨾} \tau} \quad \text{[Seq:Pfx-rel:def:cons]}$$

E.1.7 Sequence Subtraction

[Seq:Sub:Sig]	$\_ - \_ : \Sigma^* \times \Sigma^* \nrightarrow \Sigma^*$
[Seq:Sub:pre]	$\mathrm{pre}\ \sigma - \tau \mathrel{\widehat{=}} \tau \leq \sigma$
[Seq:Sub:def:nil]	$s - \langle\rangle \mathrel{\widehat{=}} s$
[Seq:Sub:def:cons]	$(x \mathbin{⨾} \sigma) - (x \mathbin{⨾} \tau) \mathrel{\widehat{=}} \sigma - \tau$

E.1.8 Strict Prefix Relation

[Seq:SPfx:Sig]	$\_ < \_ : \Sigma^* \times \Sigma^* \to \mathbb{B}$
[Seq:SPfx:def:nil]	$\langle\rangle < (y \mathbin{⨾} \tau) \mathrel{\widehat{=}} \textsc{True}$
[Seq:SPfx:def:cons]	$(x \mathbin{⨾} \sigma) < (y \mathbin{⨾} \tau) \mathrel{\widehat{=}} x = y \wedge \sigma < \tau$
[Seq:SPfx:def:lin]	$\sigma < \langle\rangle \mathrel{\widehat{=}} \textsc{False}$

$$\frac{}{\langle\rangle \leq (y \mathbin{⨾} \tau)} \quad \text{[Seq:SPfx-rel:def:nil]}$$

$$\frac{x = y \qquad \sigma \leq \tau}{x \mathbin{⨾} \sigma \leq y \mathbin{⨾} \tau} \quad \text{[Seq:SPfx-rel:def:cons]}$$

E.1.9 Sequence Front

[Seq:Front:Sig]	$\mathit{front} : \Sigma^+ \to \Sigma^*$
[Seq:Front:def:sngl]	$\mathit{front}\langle x\rangle \mathrel{\widehat{=}} \langle\rangle$
[Seq:Front:def:cons]	$\mathit{front}(x \mathbin{⨾} \sigma) \mathrel{\widehat{=}} x \mathbin{⨾} \mathit{front}(\sigma)$
[Seq:Front:def:alt]	$\mathit{front}(s \frown \langle x\rangle) \mathrel{\widehat{=}} s$

E.1.10 Sequence Last

[Seq:Last:Sig]	$last : \Sigma^{+} \rightarrow \Sigma^{*}$
[Seq:Last:def:sngl]	$last\langle x \rangle \mathrel{\widehat{=}} x$
[Seq:Last:def:cons]	$last(x \mathbin{⨾} \sigma) \mathrel{\widehat{=}} last(\sigma)$
[Seq:Last:def:alt]	$last(s \frown \langle x \rangle) \mathrel{\widehat{=}} x$
[Seq:Last:def:alt']	$last(s) \mathrel{\widehat{=}} head(s - front(s))$

E.2 Sequence Properties

E.2.1 Proof

[Seq:Front-Last:eq]

[Seq:Front-Last:eq] $s, t \neq \langle\rangle \Rightarrow$
$$(\mathit{front}(s) = \mathit{front}(t) \wedge \mathit{last}(s) = \mathit{last}(t)) \equiv (s = t)$$

E.2.2 Proof

[Seq:LE:prefix]

[Seq:LE:prefix] $s \leq t \Rightarrow \forall\, i \in 1 \ldots \# s \bullet s(i) = t(i)$

E.2.3 Proof

[Seq:Front:len]

[Seq:Front:len] $s \neq \langle\rangle \Rightarrow \#(\mathit{front}(s)) = (\# s) - 1$
[Seq:Front:len'] $s \neq \langle\rangle \Rightarrow \# s = \#(\mathit{front}(s)) + 1$

E.2.4 Proof

[Seq:Front:index]

[Seq:Front:index] $\forall\, i \in 1 \ldots \#\mathit{front}(s) \bullet (\mathit{front}(s))(i) = s(i)$
[Seq:Front:index'] $i \in 1 \ldots \#\mathit{front}(s) \Rightarrow (\mathit{front}(s))(i) = s(i)$

E.2.5 Proof

[Seq:Front:len-index]

[Seq:Front:len-index] $(\mathit{front}(s) \frown t)(\# s) = \mathit{head}(t)$

E.2.6 Proof

[Seq:Front:len]

[Seq:Front:len] $\mathit{front}(s) = \mathit{front}(t) \Rightarrow \# s = \# t$

E.2.7 Proof

[Seq:FrontLT:len]

[Seq:FrontLT:len] $\mathit{front}(s) < t \Rightarrow \# s \leq \# t$

E.2.8 Proof

[Seq:FrontLT:trans]

[Seq:FrontLT:trans] $\mathit{front}(s) < t \wedge \mathit{front}(t) < u \Rightarrow \mathit{front}(s) < u$

E.2.9 Proof

[Seq:FrontLT:eqv]

[Seq:FrontLT:eqv] $\mathit{front}(s) < t \wedge s < \mathit{front}(t) \equiv \mathit{front}(s) = \mathit{front}(t)$

E.2.10 Proof

[Seq:FrontLT:anti]

[Seq:FrontLT:anti] $\mathit{front}(s) < \mathit{front}(t) \wedge \mathit{front}(t) < \mathit{front}(s) \Rightarrow \mathit{front}(s) = \mathit{front}(t)$

E.2.11 Proof

[Seq:FrontEQ:end]

[Seq:FrontEQ:end] $(\mathit{front}(s \frown \langle x \rangle) = \mathit{front}(s \frown \langle y \rangle \frown t)) \;\equiv\; (t = \langle\rangle)$
[Seq:FrontEQ:end'] $(\mathit{front}(s) = \mathit{front}(t)) \;\equiv\; (\mathit{tail}(t - \mathit{front}(s)) = \langle\rangle)$

E.2.12 Proof

[Seq:HdSub:index]

[Seq:HdSub:index] $t < s \Rightarrow \mathit{head}(s - t) = s(1 + \#t)$
[Seq:HdSub:index'] $\mathit{front}(t) < s \Rightarrow \mathit{head}(s - \mathit{front}(t)) = s(\#t)$

E.2.13 Proof

[Seq:TailSub]

[Seq:TailSub] $\mathit{tail}(s) = \mathit{tail}(s - t) \equiv t = \langle\rangle$

E.2.14 Proof

[Seq:Front:Cat:Le]

[Seq:Front:Cat:Le] $s = \mathit{front}(t) \frown v \wedge \mathit{front}(s) < t \Rightarrow \#v = 1$

References

[Bar93] Janet E. Barnes. A Mathematical Theory of Synchronous Communication. Technical Monograph PRG-112, Oxford University Computing Laboratory Programming Research Group, Hilary Term 1993.

[BGW09] Andrew Butterfield, Pawel Gancarski, and Jim Woodcock. State visibility and communication in unifying theories of programming. *Theoretical Aspects of Software Engineering, Joint IEEE/IFIP Symposium on*, 0:47–54, 2009.

[CSW03] Ana Cavalcanti, Augusto Sampaio, and Jim Woodcock. A Refinement Strategy for *Circus*. *Formal Aspects of COmputing*, 5(2–3):146–181, November 2003.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Intl. Series in Computer Science. Prentice Hall, 1985.

[Ros97] A. W. Roscoe. *The Theory and Practice of Concurrency*. international series in computer science. Prentice Hall, 1997.

[Sch00] Steve Schneider. *Concurrent and Real-time Systems — The CSP Approach*. Wiley, 2000.

[SH03] Adnan Sherif and Jifeng He. Towards a time model for *circus*. *Lecture Notes in Computer Science*, 2495:613–624, 2003.

[She06] Adnan Sherif. *A Framework for Specification and Validation of Real Time Systems using Circus Action*. Ph.d. thesis, Universidade Federale de Pernambuco, Recife, Brazil, Jan 2006.

[SJ02] Adnan Sherif and He Jifeng. Towards a Time Model for *Circus*. Technical Report 257, UNU/IIST, P.O. Box 3058, Macau, 2002.