Abstract

This report defines an operational semantics for the event layer of the dataflow algebra, to complement the denotational semantics that has already been defined for it. The report defines a transition system and derivation sequences for this semantics, and then normal forms for these derivation sequences, which allow semantic functions to be defined directly for constructions in the algebra. The correctness of this operational semantics is then demonstrated, by proving the soundness and completeness of the axioms of the algebra with respect to it.

Key Words and Phrases

Formal specifications, dataflow algebra, syntax of expressions, parallel systems, distributed systems, semantic domains, operational semantics, denotational semantics, soundness of axioms.

1. Introduction

In the early stages of developing the dataflow algebra (DFA from now on) as a formal model for data flow diagrams [1], definitions were produced for both its abstract syntax and for its semantics [2]. The syntax was used extensively by Nike in the work for his PhD thesis [3], but the semantics received comparatively little attention until a more recent report [4] produced a simplified version of the abstract syntax. This report also made more rigorous the earlier definition of the formal semantics for the DFA (or, more precisely, for the layer of it that is now termed the event layer), and this included correcting an error in that earlier definition, which was discovered as a consequence of proving the soundness of the axioms of the algebra with respect to it.

The semantics that have been defined in this way are of the form that is usually known as denotational, because they associate with every construction that can occur within the event layer of the DFA a structure that denotes its meaning [5]. Where formal structures specify computations, however, it is often convenient to have an alternative form of semantics, where the meaning of each structure is defined instead in terms of the steps that must be carried out in order to operate (that is, execute) the corresponding computations, and such semantics are known as operational [6]. Although the event layer of the DFA only specifies one aspect of a computation, namely the sequence of actions that occur as data is communicated from one process to another during the course of that computation, this is sufficient to mean that it would be useful to have an operational semantics defined for this layer of the DFA.

The basic purpose of this report is therefore to define such an operational semantics, but just creating such a definition is only part of what has to be done in order for the results to be useful. Most importantly, as with the denotational semantics, it is also necessary to demonstrate the correctness of the definition by proving the soundness of the axioms of the algebra with respect to the semantics, where the axioms of the algebra are those stated as (i) to (ix) in section 6 of [4]. Indeed, from now on terms defined in [4] will be used throughout this report as appropriate, without further reference to [4] as their source.

Thus, the structure of the rest of this report is as follows. The basis for any operational semantics is the concept of modelling computations in terms of a transition system [7], and so section 2 describes the construction of this system, and section 3 the way in which derivation sequences are produced from it. The structures used to represent these derivation sequences have various properties, which can be described in terms of normal forms for them, and the basic concepts of these properties are defined in section 4. Then, sections 5 and 6 define respectively for each of these normal forms the operations that will normalise structures into them, and these sections also establish the properties of the structures built by these operations.
This then leads to an alternative formulation of the semantics in terms of a direct mapping from constructions in the DFA to these normalised structures, so that section 7 introduces various auxiliary operations that are needed for this purpose, and section 8 goes on to define properties of these auxiliary operations which reflect features derived from the semantics, such as associative and distributive properties for some of the operations. On this basis, the alternative formulation of the semantics is defined in section 9, along with its relationship with the original formulation of the derivation sequences. This therefore completes the definition of the semantics, and so section 10 goes on to demonstrate its consistency with the axioms of the DFA, while section 11 demonstrates the completeness of the DFA with respect to the semantics, by defining an inverse semantic function and establishing its properties. Finally, section 12 summarises the work and evaluates the significance of the results achieved, and also draws conclusions in respect of further work that is suggested by these results. This is then followed by an appendix that summarises the results of each of the 72 theorems included in this report.

2. The Transition System for the Operational Semantics

As indicated above, for any formal language that can specify computations an operational semantics is defined essentially in terms of a transition system, which describes the possible configurations of a computation that can occur as a program in that language is executed, and also describes the allowable transitions from one configuration to the next that can constitute the steps in this execution. Such configurations have two main components, one representing the language constructions that are to be executed (in other words, the program, or the rest of it), and the other representing the state of some abstract machine that models the computational process of executing this program. For conventional programming languages this process state is typically structured into two further components, one representing the memory of this abstract machine (in other words, the set of variable bindings created as the computation proceeds), and the other representing its control state. For the DFA event layer, though, the notion of variables and bindings does not arise, and so it is only the control state component that is required.

In creating any operational semantics, a key issue is to fix the size of the individual step or transition. For the DFA these will obviously be associated with the individual actions, and if this semantics were being constructed for the computational layer then it would be necessary to do this by breaking down the performance of any single action into a number of sub-steps, which at the very least would need to include the following:

- the generation by the action’s source process of the value to be output;
- the transmission of this value along the action’s channel; and
- the reception of this value by the action’s destination process.

Since this semantics is for the event layer, though, it is not necessary to go into such detail, and so it is sufficient to interpret a step in the computation as the performance of an action. This therefore suggests that it is appropriate to use a labelled transition system, in which each transition that corresponds to executing an element of PA is labelled with the action that is executed. In addition, there will need to be transitions that do not involve the execution of an element of PA, such as those that represent the execution of the constants ε or φ. These can be viewed either as unlabelled transitions, or as ones for which a label is not defined, in the sense of being empty. Where necessary this empty label will be denoted ⊥, although this should be understood as a specific value, rather than as the absence of a defined value, which is a meaning that is sometimes associated with this symbol.

Hence, the extension of the type PA by union with this empty value will be represented by a type that we will call MayAct. Similarly, we will also extend to MayAct the notation defined in [4], where the symbol = is used to represent the situation where two elements of PA have the same construction, and the symbol =/= to represent the situation where they have different constructions, so that we can write a =/= ⊥ for all elements a of MayAct such that a ∈ PA.

Given this labelling of transitions, the control state for these computations will therefore simply have to capture three possibilities for how the rest of the computation may proceed after a transition. One of these possibilities is that this transition is intended to be the last in the computation, which therefore terminates normally after it. The second possibility, which is needed to cater for the semantics of the constant φ, is that this next step will actually be the last, even though it may not have been intended to be, meaning that at this point the computation terminates abnormally. The third possibility is then that the computation does actually carry on, by making further transitions. To represent these three possibilities, we therefore introduce another type, which we will call TermState, and which will consist of the enumeration of the three values that we will call normend, abnormend and continues.

Given these, then a configuration can in principle be represented essentially by a type that we will call Config1, defined as

\[
\text{Config1} = \text{SeqConst} \times \text{TermState},
\]

so that an element of Config1 will be a pair of the form

\[
\text{Config1} = \langle \text{prog} : \text{SeqConst}, \text{control} : \text{TermState} \rangle.
\]

In practice, though, after a transition has occurred there is also the possibility that the prog component could be empty, if there is no further program to execute, and so we also need to define a second type to represent this extended possibility
An Operational Semantics for the Dataflow Algebra  A. J. Cowling

for a configuration. This type will be called Config2 and defined as $\text{Config2} = (\text{SeqConst} \cup \bot) \times \text{TermState}$, so that an element of Config2 will therefore be a pair of the form

\[ <\text{prog} : \text{SeqConst} \cup \bot, \text{control} : \text{TermState}>. \]

As with PA and MayAct, so we extend the use of $\equiv$ rather than $=$ from SeqConst to the prog elements of Config1 and Config2.

Transitions can then be represented by a type that we will call Trans, defined by

\[ \text{Trans} = \text{Config1} \times \text{MayAct} \times \text{Config2}, \]

so that elements of this type (ie elements of the transition relation) will be triples of the form

\[ <\text{old} : \text{Config1}, \text{label} : \text{MayAct}, \text{new} : \text{Config2}>. \]

In practice, though, we will follow the usual conventions for writing down these elements, so that they will be written either in the form

\[ \text{old} - \text{label} \rightarrow \text{new} \]  
\[ \text{old} \rightarrow \text{new} \]

when $\text{label} \in \text{PA}$, or

when $\text{label} = \bot$.

The elements of the transition relation then need to be defined for the various possible values that can be taken by the prog component of the old configuration. For the case of a single action this gives the following three kinds of elements, where $a$ denotes any element of PA.

(i) $<a, \text{continues}> - a \rightarrow <\bot, \text{normend}>$
(ii) $<\epsilon, \text{continues}> \rightarrow <\bot, \text{normend}>$
(iii) $<\phi, \text{continues}> \rightarrow <\bot, \text{abnormend}>$

For the case of the sequential composition $s_1 ; s_2$ separate rules are needed to cover the different possibilities for the structure of $s_1$. Firstly, if $s_1$ is an action there are three cases that are similar to those that have already been defined, as follows.

(iv) $<a ; s_2, \text{continues}> - a \rightarrow <s_2, \text{continues}>$
(v) $<\epsilon ; s_2, \text{continues}> \rightarrow <s_2, \text{continues}>$
(vi) $<\phi ; s_2, \text{continues}> \rightarrow <\bot, \text{abnormend}>$

Secondly, if $s_1$ is itself a sequential composition, then we have

(vii) $<(s_1a ; s_1b) ; s_2, \text{continues}> \rightarrow <s_1a ; (s_1b ; s_2), \text{continues}>$

where the fact that this involves some rearrangement of the prog component justifies treating it as a transition, rather than simply writing an equivalence of the form

\[ <(s_1a ; s_1b) ; s_2, \text{continues}> = <s_1a ; (s_1b ; s_2), \text{continues}>. \]

Thirdly, if $s_1$ is an alternation, then by the same argument we have

(viii) $<(s_1a | s_1b) ; s_2, \text{continues}> \rightarrow <(s_1a | s_1b) ; (s_1b ; s_2), \text{continues}>$

The final main case is that of the alternation $s_1 | s_2$, where the fact that this represents a non-deterministic choice means that in principle two possible transitions are required, of the form

\[ <s_1 | s_2, \text{continues}> \rightarrow <s_1, \text{continues}> \]
\[ <s_1 | s_2, \text{continues}> \rightarrow <s_2, \text{continues}> \]

In practice, though, the situation is not as simple as this, for if $s_1$ has the form $\phi$ or $\phi ; s_1'$ then only the transition to $s_2$ should be allowable, unless $s_2$ also has either of the forms $\phi$ or $\phi ; s_2'$, in which case both transitions are allowable, as it does not matter which of them should be taken, since either will result in abnormal termination. A similar argument applies to $s_2$, and of course both $s_1$ and $s_2$ must be treated uniformly in such situations: it would not be acceptable to model the treatment of these possibilities using any mechanism that was asymmetric.

Conceptually there are two approaches to modelling the effects of these requirements. One would be to allow the unwanted transitions to be taken, but then to introduce a backtracking mechanism to deal with those cases where it was then established that this transition should not have been taken. The other is to introduce a mechanism for guarding the transitions. Neither of these is ideal, but the concept of a backtracking mechanism seems to be less satisfactory, as introducing it would involve complicating significantly the structure of the control state for a computation in order to record when backtracking had occurred, particularly since such a mechanism would have to handle not just a simple case
such as \( s_1 = \phi \), but also cases such as \( s_1 = (\phi ; s_1a) \mid (\phi ; s_1b) \), which would require multiple levels of backtracking. Thus, the approach of guarding transitions seems to be preferable, although the guards will essentially have to involve looking ahead in the derivation sequences for \( s_1 \) and \( s_2 \), to determine whether either of them would involve executing \( \phi \) as the next step. There is, therefore, a danger that this approach could lead to circular definitions, but this can be avoided, as follows.

To formalise this approach a function \( \text{SeqHeads} \) is introduced, which will have signature \( \text{SeqConst} \to \mathcal{P} \text{Act} \), and which for any sequence will return the set of actions that could possibly be executed as the first step in executing that sequence. Thus, this function is defined directly in terms of the construction of the sequence that it takes as parameter, as follows.

\[
\begin{align*}
\text{SeqHeads}(a) &\equiv \{ a \} \text{ where } a \in \text{PA} \\
\text{SeqHeads}(\varepsilon) &\equiv \{ \varepsilon \} \\
\text{SeqHeads}(\phi) &\equiv \{ \phi \} \\
\text{SeqHeads}(s_1 \mid s_2) &\equiv \text{if } \text{SeqHeads}(s_1) = \{ \phi \} \text{ then } \text{SeqHeads}(s_2) \\
&\quad\quad\text{elseif } \text{SeqHeads}(s_2) = \{ \phi \} \text{ then } \text{SeqHeads}(s_1) \\
&\quad\quad\text{else } \text{SeqHeads}(s_1) \cup \text{SeqHeads}(s_2) \\
\text{fi} \\
\text{SeqHeads}(s_1 ; s_2) &\equiv \text{if } \text{SeqHeads}(s_1) = \{ \varepsilon \} \text{ then } \text{SeqHeads}(s_2) \\
&\quad\quad\text{elseif } \varepsilon \in \text{SeqHeads}(s_1) \text{ then } (\text{SeqHeads}(s_1) – \{ \varepsilon \}) \cup (\text{SeqHeads}(s_2) – \{ \phi \}) \\
&\quad\quad\text{else } \text{SeqHeads}(s_1) \\
\text{fi}
\end{align*}
\]

Here the significance of the various conditions in the definitions of the function for the cases of alternation and sequencing is that they need to preserve as invariants three properties of the set produced by the function that arise from properties of DFA sequences and the way in which these behave, as expressed partly (although not completely) in the axioms of the DFA. The first of these properties, which is not expressed directly in any of the axioms, is that any DFA sequence must be able to do something, even if only terminate normally or abnormally, and so the result set must never be empty. The second property is that, since \( \phi \) is the identity for alternation, it can not be an element of such a result set if this set also has any other elements, so that the only case where this set contains \( \phi \) must be the case of the singleton set \( \{ \phi \} \).

The third property is also related to identity elements, and it is that, since \( \varepsilon \) is the identity for sequencing, then it may only be an element of such a result set if the sequence can genuinely terminate normally without performing any further actions. Hence, for an sequence of the form \( \varepsilon ; a \), where \( a \) is some action, and which under the axioms must be equal to \( a \), the result of the function must be \( \{ a \} \) rather than \( \{ \varepsilon \} \), and this also applies to the two special cases of \( a = \varepsilon \) and \( a = \phi \).

Given this definition of this function with these properties, then the required guarded transitions can be written as

\[
\begin{align*}
\text{(ix)} &\text{ if } (\text{SeqHeads}(s_1) \neq \{ \phi \}) \lor (\text{SeqHeads}(s_2) = \{ \phi \}) \text{ then } <s_1 \mid s_2, \text{continues}> \rightarrow <s_1, \text{continues}> \text{ fi} \\
\text{(x)} &\text{ if } (\text{SeqHeads}(s_2) \neq \{ \phi \}) \lor (\text{SeqHeads}(s_1) = \{ \phi \}) \text{ then } <s_1 \mid s_2, \text{continues}> \rightarrow <s_2, \text{continues}> \text{ fi}
\end{align*}
\]

It will be apparent that the transitions defined in (i) to (x) have a common feature, in that the \text{control} component of the \text{old} configuration has the value \text{continues} for all of them. The implication of this is that there are no possible transitions out of configurations where this \text{control} component has either of the values \text{normend} or \text{abnormend}, and so these configurations must be terminal ones. This is also why the transition in (vi) had has its \text{new} configuration defined to have an empty \text{prog} component, rather than for this component to have the value \( s_2 \) that might have been suggested by a comparison with (iv) and (v). Thus, to complete the definition of the transition relation, these properties of configurations being terminal should be specified formally, as

\[
\begin{align*}
\text{(xi)} &\text{ \forall c : Config1 \mid c.\text{control} = \text{normend} \bullet c \text{ is a terminal configuration, and} } \\
\text{(xii)} &\text{ \forall c : Config1 \mid c.\text{control} = \text{abnormend} \bullet c \text{ is a terminal configuration.}
\end{align*}
\]

Here, the notation used for the existential qualifiers is that defined for \( Z \) [8], and this will be used as standard throughout this report, along with the equivalent form for set comprehensions. Note, though, that it is not necessary to also specify explicitly that any configuration \( c \) with \( c.\text{prog} = \perp \) must be terminal, since this follows implicitly from the way in which the type \text{Config1} has been defined to exclude this case. It is also not necessary to specify formally the inverse property, namely that any new configuration \( c \) with \( c.\text{control} = \text{continues} \) must be a non-terminal state, in the sense that it must have \( c.\text{prog} = \perp \), so that subsequent transitions are defined, but in practice this property is also an important one.
3. Derivation Sequences

In an operational semantics for any formal language, the derivation sequence for any program in that language is the sequence of transitions that results from the execution of that program by the transition system. Thus, there is a sense in which the transition system represents the semantics of the formal language itself, and then the derivation sequence for any particular program in that language represents the semantics of that program. Following this view suggests that one goal should be to define a set of semantic functions for constructions in the DFA, that will associate them with the corresponding derivation sequences, so that these can then be used for showing the soundness of the axioms of the DFA with respect to the semantics, and also for the comparisons with the denotational semantics.

In doing this, there are two important features of the DFA that need to be captured in the derivation sequences. The first of these is that, because of the inherent non-determinism of the alternation construct, the derivation sequences can not just be conventional linear sequences. Rather, they must have a branching structure, and this needs to be captured explicitly, so that the derivation sequences will actually need to be structured as trees. To avoid confusion, though, we shall not refer to them as derivation trees, since in the context of operational semantics this term is usually reserved for the structure that models the way in which a sequence of transitions is derived from the abstract syntax of the program.

The other feature of the derivation sequences for the DFA is that, as generated by the transition relation, they contain large numbers of unlabelled transitions. While these are necessary for the internal description of the process of executing a DFA specification, they have no significance for any external view of this execution process, where all that matters is the sequence of actions that are actually performed, as represented by the labelled transitions. Thus, for instance, from any external perspective the behaviour of the specification ε ; a is the same when it is executed as the behaviour of the specification a, in that both of them just perform the single action a, and this is why these two specifications are equal under the axioms of the algebra.

To reflect this, therefore, the unlabelled transitions need to be removed (or, more accurately, elided) wherever possible, and so some distinction needs to be made between the actual derivation sequences, which include all the unlabelled transitions, and the semantic structures that are derived from them by eliding as many unlabelled transitions as possible. In particular, for an alternation such as (a1 ; s1) | (a2 ; s2), this distinction means that while the original derivation sequence is a single tree, the semantic structure needs to be a forest that consists of two trees, one for (a1 ; s1) and the other for (a2 ; s2).

The obvious solution to this is to create one structure that can fulfill both purposes, and then to reflect the distinction between them by identifying special cases of these structures. Hence, the nodes for these tree or forest structures need to capture three aspects of a transition: firstly, whether it is labelled or not, and if so what its label is; secondly, the control state for the new configuration; and thirdly, the set of descendant sequences that represent the rest of the computation.

For the transition label, it is convenient for some purposes to separate the two aspects of whether a transition is labelled, and if so what its label is, and hence the natural structure is that of a dependent pair, in which the first element is a boolean to define whether or not the transition has a label (i.e., whether or not a “proper” action is performed in this step), and the second element is the proper action itself if the first element is true, and otherwise is undefined (meaning in this case that it will have the value ⊥).

Similarly, the control state and the set of descendant sequences also form a dependent pair, since this set must be empty if the new control state is either normend or abnormend, and must be non-empty if it is continues. Consequently, a single node of the tree or forest structures that are required must be a four-tuple that consists of these two dependent pairs, and to model this we introduce a type that will be called OpSUnit. Since objects of this type incorporate their descendants, they can also be understood as the root nodes of trees, and hence the required forest structures will simply be sets of these objects. To model these we introduce a type that will be called OpSem, so that these two types are defined as:

\[
\text{OpSem} = \mathcal{P} \text{OpSUnit} \quad \text{and}\quad \text{OpSUnit} = \text{Boolean} \times \text{MayAct} \times \text{TermState} \times \text{OpSem}
\]

Then, an element of the type OpSUnit is a four-tuple, with the following components.

- DoesAct: Boolean, where DoesAct is true if a possible action is performed, and false otherwise;
- TheAct: MayAct, where TheAct = ⊥ if and only if a possible action is not performed;
- NextState: TermState, which specifies whether the computation terminates or continues; and
- Rest: OpSem which specifies how the computation continues, if it does.

The dependencies between these components give rise to several invariants for values of OpSUnit, which can be stated formally as follows.
For these objects we then want to define a set of basic constructor functions, where this term is being used in the narrow sense that is common in object-oriented programming, to mean a function that builds a basic value of a type, rather than in the broad sense that is common in functional programming, where a constructor for a type may be any function that returns a result of that type. For elements of \textit{OpSUnit} there are six such constructors that are required, and most of them are given names that reflect their significance as semantic structures, rather than necessarily their role in the original derivation sequences. Thus, the first two are constants, as follows.

\[
\text{EpsSem} \equiv (\text{false}, \bot, \text{normend}, \emptyset) \\
\text{PhiSem} \equiv (\text{false}, \bot, \text{abnormend}, \emptyset)
\]

Then, the other four are functions. One, called \text{FinalActSem}, has the signature \(\text{PA} \rightarrow \text{OpSUnit}\) and is defined as

\[
\text{FinalActSem} (a) \equiv (\text{true}, a, \text{normend}, \emptyset)
\]

The second is similar, and is called \text{FinalAbActSem}, and also has the signature \(\text{PA} \rightarrow \text{OpSUnit}\). It is defined as

\[
\text{FinalAbActSem} (a) \equiv (\text{true}, a, \text{abnormend}, \emptyset)
\]

The third one is then called \text{ContActSem}, and has the signature \(\text{PA} \times \text{OpSem} \rightarrow \text{OpSUnit}\), and it is defined as

\[
\text{ContActSem} (a, os) \equiv (\text{true}, a, \text{continues}, os)
\]

The final one is only required to represent transitions in the original derivation sequences, and so it is called \text{EmptyTrans}; it has the signature \(\text{OpSem} \rightarrow \text{OpSUnit}\), and it is defined as

\[
\text{EmptyTrans} (os) \equiv (\text{false}, \bot, \text{continues}, os)
\]

Thus, in terms of the forest structures, \text{EpsSem}, \text{PhiSem}, \text{FinalActSem} and \text{FinalAbActSem} all build terminal nodes of the trees involved, while both \text{ContActSem} and \text{EmptyTrans} implicitly have the precondition that \(os \neq \emptyset\), and in this case they build non-terminal nodes of the trees. Hence, it should in principle be obvious that these all create elements of \text{OpSUnit} that conform to invariants (i) to (v), except that \text{ContActSem} and \text{EmptyTrans} will only create elements that conform to invariant (v) in the case where this implicit precondition of \(os \neq \emptyset\) holds. It is therefore useful to formalise these properties as theorems, as follows.

**Theorem 1.**

\text{EpsSem} and \text{PhiSem} are both consistent with the invariants (i), (ii), (iii), (iv) and (v) above.

**Proof.**

Each invariant follows directly by calculation of the right hand side from the construction of the objects concerned. For example, for invariant (i) we have in each case that the field \text{DoesAct} = \text{false} and the field \text{TheAct} = \bot, and hence the implication in the invariant evaluates to \text{true}. The corresponding calculations for the other invariants are similar.

\[\blacksquare\]

**Theorem 2.**

\[\forall a : \text{PA} \Rightarrow \text{FinalActSem} (a) \text{ and } \text{FinalAbActSem} (a) \text{ are both consistent with the invariants (i), (ii), (iii), (iv) and (v) above}.\]

**Proof.**

Again, each invariant follows directly by calculation of the right hand side from the construction of the objects concerned. For example, for invariant (ii) we have in each case that the field \text{DoesAct} = \text{true} and the field \text{TheAct} \in \text{PA}, and hence the implication in the invariant evaluates to \text{true}. The corresponding calculations for the other invariants are similar.

\[\blacksquare\]

For the equivalent theorem for \text{ContActSem} does, however, need additional conditions, for any condition defined over the nodes can only apply to a tree structure if it applies to all of the nodes in the tree, and not just to the root. Hence, for each invariant, the requirement that it holds for the set of trees representing the rest of the computation must be specified as a condition in the theorem, which is therefore stated as follows.

**Theorem 3.**

\[\forall a : \text{PA}, \forall os : \text{OpSem} \Rightarrow \text{os is consistent with invariant (i) } \Rightarrow \text{ContActSem} (a, os) \text{ is consistent with invariant (i)} \]
EmptyTrans (os) is consistent with invariant (i) \land
os is consistent with invariant (ii) \Rightarrow ContActSem (a, os) is consistent with invariant (ii) \land
EmptyTrans (os) is consistent with invariant (ii) \land
os is consistent with invariant (iii) \Rightarrow ContActSem (a, os) is consistent with invariant (iii) \land
EmptyTrans (os) is consistent with invariant (iii) \land
os \neq \emptyset \land os is consistent with invariant (iv) \Rightarrow ContActSem (a, os) is consistent with invariant (iv) \land
EmptyTrans (os) is consistent with invariant (iv) \land
os \neq \emptyset \land os is consistent with invariant (v) \Rightarrow ContActSem (a, os) is consistent with invariant (v) \land
EmptyTrans (os) is consistent with invariant (v).

Proof. Again, each invariant follows directly by calculation of the right hand side from the construction of the objects concerned. In particular, for invariant (v) we have that the field NextState = continues and (from the condition that os \neq \emptyset) that the field Rest \neq \emptyset, and hence the implication in the invariant evaluates to \text{true} for the root node.

\[\]

Using these constructors and their properties, we can define the basic function that produces a derivation sequence for any construction in the DFA, by constructing for each possible first transition an equivalent root node, and then recursively applying itself to the resultant configuration in order to produce the rest of the sequence. This function is called DerSeq, and it has the signature \(\text{SeqConst} \rightarrow \text{OpSem}\). It is then defined as follows, where the lines in the definition are annotated to identify the relevant clauses in the definition of the transition relation.

\[
\text{DerSeq} (s) \equiv \{ \forall t : \text{Trans} | t.old.prog = s \bullet
  \begin{cases}
    \text{if } t.new.control = \text{continues} \land t.label \in \text{PA} & \text{(iv)} \\
    \text{then } \text{ContActSem} (t.label, \text{DerSeq} (t.new.prog)) & \\
    \text{elsif } t.new.control = \text{continues} \land t.label = \bot & (v), (vii), (viii), (ix), (x) \\
    \text{then } \text{EmptyTrans} (\text{DerSeq} (t.new.prog)) & \\
    \text{elsif } t.new.control = \text{normend} \land t.label \in \text{PA} & (i) \\
    \text{then } \text{FinalActSem} (t.label) & \\
    \text{elsif } t.new.control = \text{normend} \land t.label = \bot & \\
    \text{then } \text{EpsSem} & (ii) \\
    \text{else } \text{PhiSem} & (iii), (vi)
  \end{cases}
\}
\]

4. Normal Forms for the Semantic Structures

As has already been noted, the semantic structures that will eventually be defined for the various constructions in the DFA will be special cases of the derivation sequences, in which certain classes of empty (ie unlabelled) transitions will have been elided. These classes of special case have a similar status to the notion of normal forms in relational databases, in the sense that we can define conditions that will determine whether or not particular forest structures possess the properties of belonging to these special cases (in other words, whether or not they are in one of these normal forms), and we can define normalisation functions that will transform arbitrary objects into equivalents that are in these normal forms.

In principle there are just two of these normal forms, and each one is associated with two properties of constructions in the DFA that need to be reflected in the semantics. In practice, though, it is also convenient to treat the invariants (i) to (v) as defining a normal form, since there are no encapsulation rules in the language of mathematics, and hence one can not assume that the creation of objects of type OpSUnit will be restricted to applications of the constructor functions defined above. Hence, technically it would be possible to denote the creation of an arbitrary tuple of this type that did not conform to these invariants, and so tuples that do conform to them can usefully be regarded as constituting what we will call a zeroth normal form. This can be expressed in terms of a pair of characteristic functions that are called OSNorm0 and OUNorm0, with signatures OpSem \rightarrow \text{Boolean} and OpSUnit \rightarrow \text{Boolean} respectively, and which are defined as follows.

\[
\text{OSNorm0} (os) \equiv \text{if } os = \emptyset \text{ then true else } \forall ou : \text{OpSUnit} | ou \in os \bullet \text{OUNorm0} (ou) \bullet fi \\
\text{OUNorm0} (ou) \equiv \text{if } ou.\text{DoesAct} \text{ then } ou.\text{TheAct} \in \text{PA} \text{ else } ou.\text{TheAct} = \bot \bullet fi \land
  \begin{cases}
    \text{if } ou.\text{NextState} = \text{normend} \lor ou.\text{NextState} = \text{abnormend} & \\
    \text{then } ou.\text{Rest} = \emptyset & (i) \\
    \text{else } ou.\text{Rest} \neq \emptyset \land \text{OSNorm0} (ou.\text{Rest}) & \\
    \text{fi}
  \end{cases}
\]
Thus, the recursive nature of these definitions implicitly captures the property referred to above, that for a tree structure to conform to the invariant properties given above they must hold for all nodes, and not just for the root. Given these definitions, then alternative versions of theorems 1 and 2 can be expressed as follows.

**Theorem 4.**

\[
\begin{align*}
&\text{OUNorm0 (EpsSem) = true} \\
&\land \text{OUNorm0 (PhiSem) = true} \\
&\land \forall a : PA \bullet \text{OUNorm0 (FinalActSem (a)) = true} \\
&\land \forall a : PA \bullet \text{OUNorm0 (FinalAbActSem (a)) = true}.
\end{align*}
\]

**Proof.**

The proofs follow directly from the definition of OUNorm0 and from theorems 1 and 2.

For the equivalent alternative version of theorem 3, however, the recursive definition of the functions means that the proof needs to be inductive, and this induction is essentially over the heights of the trees concerned. These therefore need to be defined formally, and this can be done in terms of two functions called \( \text{HeightOS} \) and \( \text{HeightOU} \), which have signatures \( \text{OpSem} \rightarrow \mathbb{N} \) and \( \text{OpSUnit} \rightarrow \mathbb{N} \) respectively, and are defined as follows.

\[
\begin{align*}
\text{HeightOS (os)} &\equiv \text{if } os = \emptyset \text{ then } 0 \text{ else } \max \{ \forall ou : \text{OpSUnit} | ou \in os \bullet \text{HeightOU (ou)} \} \\
\text{HeightOU (ou)} &\equiv 1 + \text{HeightOS (ou.Rest)}
\end{align*}
\]

Hence it will be apparent that, since any object of type \( \text{OpSUnit} \) constructed by any of the functions \( \text{EpsSem}, \text{PhiSem}, \text{FinalActSem} \) or \( \text{FinalAbActSem} \) must be a terminal node, then it must have height 1, while any valid object constructed by \( \text{ContActSem} \) or \( \text{EmptyTrans} \) must be a non-terminal node, and so must have height greater than 1. The inverse of this property is that, for objects in zeroth normal form, non-terminal nodes (with height greater than 1) must be constructed by \( \text{ContActSem} \) or \( \text{EmptyTrans} \), while terminal nodes (with height 1) must be constructed by one of the other four functions. The latter of these properties is important, as it provides the base cases needed for structural inductions over the trees, and it is expressed formally as the following theorem.

**Theorem 5.**

\[
\forall ou : \text{OpSUnit} \bullet ( \text{HeightOU (ou) = 1}) \land \text{OUNorm0 (ou)} \Rightarrow (ou = \text{EpsSem}) \lor (ou = \text{PhiSem}) \lor \\
( \exists a : PA \bullet (ou = \text{FinalActSem (a)}) \lor (ou = \text{FinalAbActSem (a)}) ).
\]

**Proof.**

\[
\begin{align*}
\text{HeightOU (ou)} &\equiv 1 \Rightarrow ou.Rest = \emptyset \\
(ou.Rest = \emptyset) \land \text{OUNorm0 (ou)} \Rightarrow ou.NextState = \text{normend} \lor ou.NextState = \text{abnormend}.
\end{align*}
\]

Hence, there are only four possible cases for the construction of \( ou \) satisfying the conditions of the theorem, as follows.

Case (i):

\[
ou.\text{DoesAct} = \text{false} \land ou.\text{NextState} = \text{normend} \Rightarrow ou.\text{TheAct} = \bot \Rightarrow ou = \text{EpsSem}
\]

Case (ii):

\[
ou.\text{DoesAct} = \text{true} \land ou.\text{NextState} = \text{normend} \Rightarrow \exists a : PA \bullet ou.\text{TheAct} = a \Rightarrow ou = \text{FinalActSem (a)}
\]

Case (iii):

\[
ou.\text{DoesAct} = \text{false} \land ou.\text{NextState} = \text{abnormend} \Rightarrow ou.\text{TheAct} = \bot \Rightarrow ou = \text{PhiSem}
\]

Case (iv):

\[
ou.\text{DoesAct} = \text{true} \land ou.\text{NextState} = \text{abnormend} \Rightarrow \exists a : PA \bullet ou.\text{TheAct} = a \Rightarrow ou = \text{FinalAbActSem (a)}
\]

The theorem then follows directly from the combination of these four cases.

Given these properties, then the alternative version of theorem 3 can be expressed as follows.

**Theorem 6.**

\[
\forall a : PA, os : \text{OpSem} | os \neq \emptyset \bullet \text{OSNorm0 (os) } \Rightarrow \text{OUNorm0 (ContActSem (a, os))} \\
\land \forall os : \text{OpSem} | os \neq \emptyset \bullet \text{OSNorm0 (os) } \Rightarrow \text{OUNorm0 (EmptyTrans (os))}.
\]
Proof.
The proof is by induction over the heights of the forests that comprise \( \text{os} \). Hence, the base case is where \( \text{os} \) is constructed entirely from elements \( \text{ou} : \text{OpSUnit} \) whose height is 1, where the condition \( \text{OSNorm0} (\text{os}) \) implies that the property \( \text{OUNorm0} (\text{ou}) \) holds, and since all such elements can be constructed as in theorem 5, this base case holds directly from theorem 4.

The induction step is then to show that, for any natural number \( n \geq 1 \), the theorem must hold for any element constructed as either \( \text{ContActSem} (\text{a}, \text{os}) \) or \( \text{EmptyTrans} (\text{os}) \) and with height \( n + 1 \) if it holds for all \( \text{os} : \text{OpSem} \) satisfying the conditions:

\[
\begin{align*}
\text{os} &\neq \emptyset, \\
\text{all elements of os have height} &\leq n, \text{ and} \\
\text{at least one element of os has height} &n. 
\end{align*}
\]

The proof of this induction step follows directly from:

- the definition of \( \text{HeightOS} (\text{os}) \), which under these conditions must evaluate to \( n \),
- the definition of \( \text{OSNorm0} (\text{os}) \), which under these conditions must evaluate to \text{true}, and
- from theorem 3 and the definition of \( \text{OUNorm0} \).

Hence, by induction the theorem holds for all values of \( n \geq 1 \), and so holds.

Hence, we can define that any object \( \text{ou} : \text{OpSUnit} \) is in zeroth normal form if the condition \( \text{OUNorm0} (\text{ou}) \) holds, and similarly that any object \( \text{os} : \text{OpSem} \) is in zeroth normal form if the condition \( \text{OSNorm0} (\text{os}) \) holds. Then, invariants (i) to (v) effectively specify:

\[
\forall \text{ou} : \text{OpSUnit} \bullet \text{OUNorm0} (\text{ou}) \quad \text{and} \\
\forall \text{os} : \text{OpSem} \bullet \text{OSNorm0} (\text{os}) 
\]

which we may regard as meaning that all valid elements are in this form. Similarly, theorems 4 and 6 guarantee that any trees or forests constructed by valid applications of the constructor functions must be in zeroth normal form, which suggests that for practical purposes we could perhaps ignore the issue of structures that are not in zeroth normal form, as they could only be produced by writing directly tuples that were not in zeroth normal form. For completeness, though, we do regard it as necessary to prove that some of the functions to be introduced below do produce structures that are in this form, and in particular it is important to do this for the function \( \text{DerSeq} \). This property is expressed as the following theorem.

Theorem 7.
\[
\forall s : \text{SeqConst} \bullet \text{OSNorm0} (\text{DerSeq} (s) ).
\]

Proof.
Essentially the proof is by induction over the heights of the forests that are produced by \( \text{DerSeq} \), but because these depend on the parameter \( s \) the proof actually has to be structured as an induction over the structure of \( s \), using the function \( \text{SCC}(s) \) as metric. In doing this, the different cases for \( s \) that need to be analysed correspond directly to the different cases in the definition of the transition relation, which are referred to simply as “clauses” for the purposes of this proof.

Base case.
The base case is defined by \( \text{SCC}(s) = 1 \), meaning that \( s \) is a single action, and so it has three sub-cases, corresponding to the different classes of action that need to be distinguished, as follows.

Sub-case (i): \( s = a \in \text{PA} \)

\[
\begin{align*}
\Rightarrow & \quad \text{DerSeq} (s) = \{ \text{FinalActSem} (a) \} \\
\Rightarrow & \quad \text{OSNorm0} (\text{DerSeq} (s)) 
\end{align*}
\]

from clause (i)

from theorem 4.

Sub-case (ii): \( s = \varepsilon \)

\[
\begin{align*}
\Rightarrow & \quad \text{DerSeq} (s) = \{ \text{EpsSem} \} \\
\Rightarrow & \quad \text{OSNorm0} (\text{DerSeq} (s)) 
\end{align*}
\]

from clause (ii)

from theorem 4.

Sub-case (iii): \( s = \phi \)

\[
\begin{align*}
\Rightarrow & \quad \text{DerSeq} (s) = \{ \text{PhiSem} \} \\
\Rightarrow & \quad \text{OSNorm0} (\text{DerSeq} (s)) 
\end{align*}
\]

from clause (iii)

from theorem 4.
Hence, the theorem holds for all three sub-cases of the base case, and hence for the base case.

**Inductive case.**

The inductive case is defined by SCC(s) = n where n > 1, and the induction hypothesis is that the theorem holds for all s1, s2 such that SCC(s1) < n, SCC(s2) < n. The induction step is then to show from this that the theorem must therefore hold for all sequences s with SCC(s) = n, and this involves two main sub-cases, one for each of the possible structures of s, as follows.

Inductive sub-case: Alternation: \( s = s_1 | s_2 \).

Let os1, os2 : OpSem be defined as os1 = DerSeq (s1), os2 = DerSeq(s2), so that the induction hypothesis gives OSNorm0 (os1) and OSNorm0 (os2).

There are then four sub-cases to be considered, depending on whether or not SeqHeads (s1) = \{ \phi \} and whether or not SeqHeads (s2) = \{ \phi \}.

Sub-case (i): SeqHeads (s1) = \{ \phi \} ∧ SeqHeads (s2) = \{ \phi \}

\[ \Rightarrow \text{DerSeq (s) = \{ EmptyTrans (os1), EmptyTrans (os2)\}} \] from clauses (ix), (x)

\[ \Rightarrow \text{OSNorm0 (DerSeq (s))} \] from theorem 6.

Sub-case (ii): SeqHeads (s1) = \{ \phi \} ∧ SeqHeads (s2) \neq \{ \phi \}

\[ \Rightarrow \text{DerSeq (s) = \{ EmptyTrans (os2)\}} \] from clauses (ix), (x)

\[ \Rightarrow \text{OSNorm0 (DerSeq (s))} \] from theorem 6.

Sub-case (iii): SeqHeads (s1) \neq \{ \phi \} ∧ SeqHeads (s2) = \{ \phi \}

\[ \Rightarrow \text{DerSeq (s) = \{ EmptyTrans (os1)\}} \] from clauses (ix), (x)

\[ \Rightarrow \text{OSNorm0 (DerSeq (s))} \] from theorem 6.

Sub-case (iv): SeqHeads (s1) \neq \{ \phi \} ∧ SeqHeads (s2) \neq \{ \phi \}

\[ \Rightarrow \text{DerSeq (s) = \{ EmptyTrans (os1), EmptyTrans (os2)\}} \] from clauses (ix) and (x)

\[ \Rightarrow \text{OSNorm0 (DerSeq (s))} \] from theorem 6.

Hence, the theorem holds for all four of these sub-cases, and hence for this inductive sub-case.

Inductive sub-case: Sequencing: \( s = s_1 ; s_2 \).

Again, let os1, os2 : OpSem be defined as os1 = DerSeq (s1), os2 = DerSeq(s2), so that the induction hypothesis gives OSNorm0 (os1) and OSNorm0 (os2).

Here the sub-cases to be considered depend on the structure of s1, and involve a second inner induction over SCC(s1), for which the base case consists of the same three cases as in the base case above, as follows.

Sub-case (i): \( s_1 = a \in PA \)

\[ \Rightarrow \text{DerSeq (s1 ; s2) = \{ ContActSem (a, os2)\}} \] from clause (iv)

\[ \Rightarrow \text{OSNorm0 (DerSeq (s))} \] from theorem 6.

Sub-case (ii): \( s_1 = \varepsilon \)

\[ \Rightarrow \text{DerSeq (s1 ; s2) = \{ EmptyTrans (os2)\}} \] from clause (v)

\[ \Rightarrow \text{OSNorm0 (DerSeq (s))} \] from theorem 6.

Sub-case (iii): \( s_1 = \phi \)

\[ \Rightarrow \text{DerSeq (s1 ; s2) = \{ PhiSem\}} \] from clause (vi)

\[ \Rightarrow \text{OSNorm0 (DerSeq (s))} \] from theorem 4.

Then there are two inductive sub-cases, one for s1 constructed by alternation and the other for s1 constructed by sequencing, where the induction hypothesis for these is that, for some natural number ni such that 1 < ni < n, the theorem holds for all s1 such that SCC(s1) < ni. The induction step is therefore to show that in each sub-case it also holds for all s1 such that SCC(s1) = ni, as follows.

Sub-case (iv): \( s_1 = s_1a | s_1b \), so that from clause (viii)

\[ \text{DerSeq (s1 ; s2) = \{ EmptyTrans ((DerSeq ((s_1a ; s_2) | (s_1b ; s_2)) )\}} \]

and as in the main alternation case we have four different sub-cases for DerSeq ((s1a ; s2) | (s1b ; s2)), depending on whether or not SeqHeads (s1a) = \{ \phi \} and whether or not SeqHeads (s1b) = \{ \phi \}. As in that main alternation case, though, this actually just gives rise to three possible constructions, which are
\[ \text{DerSeq } ((s_1a ; s_2) | (s_1b ; s_2)) = \{ \text{EmptyTrans (DerSeq } (s_1a ; s_2) ), \text{EmptyTrans (DerSeq } (s_1b ; s_2) ) \} \]
\[ \text{or} \]
\[ \text{DerSeq } ((s_1a ; s_2) | (s_1b ; s_2)) = \{ \text{EmptyTrans (DerSeq } (s_1b ; s_2) ) \} \]

In each of these three cases, though, since \( \text{SCC}(s_1a) < \text{SCC}(s_1) \) and \( \text{SCC}(s_1b) < \text{SCC}(s_1) \), it follows that \( \text{SCC}(s_1a ; s_2) < n \) and \( \text{SCC}(s_1b ; s_2) < n \), and so the induction hypothesis applies to both \( \text{DerSeq } (s_1a ; s_2) \) and \( \text{DerSeq } (s_1b ; s_2) \), and so we have

\[ \text{OSNorm0} (\text{DerSeq } (s_1a ; s_2)) \land \text{OSNorm0} (\text{DerSeq } (s_1b ; s_2)) \]
\[ \Rightarrow \text{OSNorm0} (\text{EmptyTrans (DerSeq } (s_1a ; s_2) )) \land \text{OSNorm0} (\text{EmptyTrans (DerSeq } (s_1b ; s_2) )) \]
\[ \Rightarrow \text{OSNorm0} (\text{DerSeq } ((s_1a ; s_2) | (s_1b ; s_2))) \]

from theorem 6

\[ \Rightarrow \text{OSNorm0} (\text{DerSeq } (s_1 ; s_2)) \]

Sub-case (v): \( s_1 = s_1a ; s_1b \), so that from clause (vii)

\[ \text{DerSeq } (s_1 ; s_2) = \{ \text{EmptyTrans (DerSeq } (s_1a ; (s_1b ; s_2))) \} \]

Here, since \( \text{SCC}(s_1a) + \text{SCC}(s_1b) = \text{SCC}(s_1) \) and \( \text{SCC}(s_1) + \text{SCC}(s_2) = n \), it follows that \( \text{SCC}(s_1a) < n \) and \( \text{SCC}(s_1b ; s_2) < n \), and so the inductive argument for this inner induction is that, if by the outer induction we have

\[ \text{OSNorm0} (\text{DerSeq } (s_1a ; (s_1b ; s_2))) \]

then we also have

\[ \text{OSNorm0} (\{ \text{EmptyTrans (DerSeq } (s_1a ; s_2)) \mid (s_1b ; s_2) \}) \]

from theorem 6

\[ \Rightarrow \text{OSNorm0} (\text{DerSeq } (s_1 ; s_2)) \]

Then sub-cases (i) to (iii) provide the base cases for this inner induction, and from sub-cases (iv) and (v) it follows by this inner induction that the theorem holds for the inductive sub-case of sequencing for all \( ni \). This together with the inductive sub-case for alternation shows that the outer induction holds for all \( s \) with \( \text{SCC}(s) = n > 1 \), and hence for all \( n \). Hence the theorem holds.

Having proved this property, we can turn attention to the two normal forms that are to be constructed on top of the definition of zeroth normal form. Each of these is associated with two properties of the DFA constructions, and so they can be treated as independent of each other, meaning that in principle they need to be defined separately. In practice, though, we wish to define both of them in a way that depends on zeroth normal form, and also to define second normal form in a way that depends on zeroth normal form. This suggests that in practice the situation will be analogous to that of the normal forms for relational databases (as described, for instance, in [9]), where part of the definition of a database being in second normal form is that it must also be in first normal form, and similarly part of the definition of a database being in third normal form is that it must also be in second normal form, and so by implication also in first normal form.

The way in which this is dealt with here is by introducing the term **strict normal form** to refer to each of the two independent concepts of normal form, so that for each of the strict normal forms the defining characteristic condition just has two parts, corresponding to the two properties of the DFA constructions that it models. Thus, where the properties of the normal forms are being discussed within this report, the focus will be on the strict normal forms. Then, for more general use, characteristic functions will be defined for each of first and second normal forms that also include the requirements for structures to be in the normal forms on which they depend, so that for first normal form these functions will also require objects to be in zeroth normal form, and similarly for second normal form they will also require objects to be in first normal form, and hence in zeroth normal form as well.

The first of these strict normal forms arises partly from the case of DFA expressions such as \( \varepsilon ; a \), where the semantics need to capture the fact that actually only one action is performed, namely \( a \). Thus, while the initial derivation sequence for such an expression would involve two objects of type OpSUnit, one corresponding to each action, which in this case would have values of the form:

\[ \text{first} = (\text{false}, \bot, \text{continues}, \{\text{second}\}) \quad \text{and} \quad \text{second} = (\text{true}, a, \text{normend}, \emptyset) \]

what is actually required in the semantic structure to capture the fact that only one action is performed is to collapse these two down into a single object of the form

\[ \text{single} = (\text{true}, a, \text{normend}, \emptyset) \]

To express the fact that this collapsing is required, the normal form that is needed must specify that, if execution continues, then an action must be performed. In terms of the individual tree structures involved, what this means is that non-terminal nodes must perform actions, and this can be expressed formally as a normalisation condition called \( \text{OUNorm1a} \), which has signature \( \text{OpSUnit} \rightarrow \text{Boolean} \), and is defined as
Hence, it is desirable to insist that any set of elements of
that each provide just part of the answer.

performed?”, and answering such questions becomes more diffi
cult if there may be a number of different constructions
semantics at all is to provide answers to questions of the form “what can happen next after a particular action has been
both elements the first step would be to perform the action
expressed in the form

The need for the second normal form can be seen by considering a DFA expression such as $\text{a ; } ε$, and indeed this would suggest that in most cases a stronger invariant than (vi) could be expected to apply, because the initial derivation sequence for this expression would involve two objects of type $\text{OpSUnit}$, one corresponding to each action, with values of the form:

\[
\begin{align*}
third &= (\text{true, a, continues, (fourth)}) \\
\text{fourth} &= (\text{false, } ⊥, \text{normend, } ∅)
\end{align*}
\]

Again, though, what is actually required, in order to capture the fact that only one action is performed, is that these two
should be collapsed down into a single object of the form

\[
single2 = (\text{true, a, normend, } ∅)
\]

This would therefore suggest that an action must also be performed in any object where execution terminates normally, so
that the only valid case where an action is not performed would be an object where the computation terminates
abnormally. In terms of the individual tree structures, this would suggest that nodes which terminate normally without
performing an action are unnecessary, even as terminal nodes of a tree, and indeed it might also suggest by analogy that
nodes which terminate abnormally without performing an action are also unnecessary. There is, though, one obvious case
where the first of these breaks down, namely the DFA expression $ε$ on its own, and while this is effectively an empty
specification, it is still a valid one. Consequently, its semantics (which would basically consist of the object called $\text{fourth}$
above) must be valid. Similarly, the DFA expression $ϕ$ on its own is also valid, and its semantics would need to be the
single node that terminates abnormally without performing an action, so this tree structure has to be valid too.

Focussing first on the case of normal termination, in terms of the tree structures, a node that performs no action would be
perfectly legitimate as a member of a set of descendents where the others did perform actions, as for instance with the two
possibilities that follow the action $a$ in the DFA expression $a ; (b | ε)$. Where it is not necessary to have such a node,
though, is if it is the only descendant, and so the normalisation condition that is needed here can not just have a similar
form to $\text{OUNorm1a}$. Instead, it must be stated in the form that, if the rest of the computation consists of a single element
set, then the object that is the element of this set must perform an action. It can therefore be expressed formally as a
normalisation condition called $\text{OUNorm1b}$, which again has signature $\text{OpSUnit} → \text{Boolean}$, and is defined as

\[
\text{OUNorm1b} (\text{ou}) = \text{if } ∃\text{ou2 : OpSUnit} \bullet \text{ou.Rest} = \{ \text{ou2} \} \text{ then ou2.DoesAct } \text{ true } \text{ fi}
\]

Then, we can define the strict first normal form rigorously, in terms of the recursive closure of the individual
normalisation conditions over the complete tree and forest structures, rather than just their applications to the root nodes.
This requires a pair of functions, one called $\text{OSNorm1}$ that applies to values from $\text{OpSem}$, and the other called
$\text{OUNorm1}$ that applies to values from $\text{OpSUnit}$, so that the functions have signatures $\text{OpSem} → \text{Boolean}$ and $\text{OpSUnit} → \text{Boolean}$ respectively. Thus, any object is said to be in strict first normal form if the appropriate one of these
conditions holds for it, where the conditions are defined as

\[
\text{OUNorm1a} (\text{ou}) ≡ \text{ou.DoesAct} = \text{false } ⇒ \text{ou.NextState } ≠ \text{ continues}
\]

\[
\begin{align*}
\text{OSNorm1} (\text{os}) &≡ \text{if os } = \emptyset \text{ then true else } \forall \text{ou : OpSUnit } | \text{ou } ∈ \text{os } \bullet \text{OSNorm1 (ou) fi}
\end{align*}
\]

Similarly, for the more general notion of first normal form, we can define a pair of functions, called $\text{OSNorm01}$ and
$\text{OUNorm01}$, and with signatures $\text{OpSem} → \text{Boolean}$ and $\text{OpSUnit} → \text{Boolean}$ respectively, so that any object is said to be in first normal form if the appropriate one of these conditions holds for it, where the conditions are defined as

\[
\begin{align*}
\text{OUNorm1} (\text{ou}) &≡ \text{OUNorm0 (ou) } \land \text{OUNorm1 (ou)} \\
\text{OSNorm01} (\text{os}) &≡ \text{OSNorm0 (os) } \land \text{OSNorm1 (os)}
\end{align*}
\]

The need for the second normal form can be seen by considering a DFA expression such as $a ; (b | c)$, which can also be
expressed in the form $(a ; b) | (a ; c)$. The obvious structure for the semantics of this second form would involve a set of
two elements of type $\text{OpSUnit}$, one representing the semantics of $a ; b$ and the other the semantics of $a ; c$, so that for
both elements the first step would be to perform the action $a$. However, part of the reason for wanting an operational
semantics at all is to provide answers to questions of the form “what can happen next after a particular action has been
performed?”, and answering such questions becomes more difficult if there may be a number of different constructions
that each provide just part of the answer.

Hence, it is desirable to insist that any set of elements of $\text{OpSUnit}$ must be constrained so that each performs a different
action, meaning that the semantics of such an expression may not take the form sketched above, but implicitly must
instead be structured so as to correspond to the first form of the expression. In terms of the forest structure, this is
equivalent to requiring that each tree in a forest must have a different action associated with its root node. This property applies essentially to a set of trees, and so it can be expressed formally as a normalisation condition called $\text{OSNorm2a}$, which has signature $\text{OpSem} \rightarrow \text{Boolean}$, and is defined as

\[
\text{OSNorm2a} (\text{os}) \equiv \forall \text{ou1}, \text{ou2} : \text{OpSUnit} | \text{ou1} \in \text{os} \land \text{ou2} \in \text{os} \bullet
\text{ou1}.\text{TheAct} = \text{ou2}.\text{TheAct} \Rightarrow \text{ou1} = \text{ou2}
\]

This definition then illustrates the limitation of the conditions in $\text{OUNorm1b}$ to single element sets that was described above, for in the case of the DFA expression $a ; (b | c)$ that was discussed there the application of $\text{OSNorm2a}$ would mean that the semantics would have to correspond to this form of the expression, rather than to the alternative form $(a ; b) | a$. Hence, the rest of the computation for the object representing the step of performing $a$ would have to consist of two objects, one of which corresponds to $c$, and this cannot be eliminated, as already indicated. It is therefore different from the case where it would appear as the only possibility for the rest of the computation, and so would have to be eliminated to conform to $\text{OUNorm1b}$.

Turning to the case of abnormal termination, the other condition that is required for $\text{OpSem}$ is to capture the property of the DFA that $\phi$ is the identity for alternation, as in the guards to clauses (ix) and (x) of the definition of the transition relation. In terms of these semantic structures, what this means is that, if any set of alternatives includes one that performs no action but just terminates abnormally, then this alternative will never be selected, and so it should be eliminated from the set. Of course, the guards on these particular transitions will ensure that no derivation sequence is ever produced that contains such alternatives, but in the more general context of the semantic structures that are produced from the derivation sequences by the process of normalisation it can not necessarily be guaranteed that this property would automatically be maintained, and so it needs to be included explicitly in the normalisation conditions.

Hence, the last of the four properties required for the normal forms is one to specify that no set of more than one object may include such an object, and this is expressed formally as a normalisation condition called $\text{OSNorm2b}$, which again has signature $\text{OpSem} \rightarrow \text{Boolean}$, and is defined as follows, where (as in $\mathbb{Z}$) the operator $\#$ is used to denote the cardinality of a set.

\[
\text{OSNorm2b} (\text{os}) \equiv \# \text{os} > 1 \Rightarrow
\neg (\exists \text{ou} : \text{OpSUnit} | \text{ou} \in \text{os} \bullet (\text{ou}.\text{DoesAct} = \text{false} \land \text{ou}.\text{NextState} = \text{abnormend}) )
\]

It will be noted that a special case of this condition, which could arise if there were another element of the set that performs no action but terminates normally, would also be prohibited by the conditions of $\text{OSNorm2a}$, since both of these objects would have the same value for the component $\text{TheAct}$ (viz $\perp$), but would be different objects (because they would have different values for the component $\text{NextState}$). Where this situation does not arise, though, the form in $\text{OSNorm2b}$ is more general, and so both are needed.

The rigorous definition of strict second normal form is then analogous to that of strict first normal form, in that the two functions required are called $\text{OSNorm2}$ and $\text{OUNorm2}$, and again their signatures are respectively $\text{OpSem} \rightarrow \text{Boolean}$ and $\text{OpSUnit} \rightarrow \text{Boolean}$. Thus, any object is said to be in strict second normal form if the appropriate one of these conditions holds for it, where the conditions are defined as

\[
\text{OSNorm2} (\text{os}) \equiv \text{OSNorm2a} (\text{os}) \land \text{OSNorm2b} (\text{os}) \land
\text{if } \text{os} = \emptyset \text{ then true else } \forall \text{ou} : \text{OpSUnit} | \text{ou} \in \text{os} \bullet \text{OUNorm2} (\text{ou}) \text{ fi}
\]

\[
\text{OUNorm2} (\text{ou}) \equiv \text{if } \text{ou}.\text{Rest} = \emptyset \text{ then true else } \text{OSNorm2} (\text{ou}.\text{Rest}) \text{ fi}
\]

Similarly, for the more general notion of second normal form we define a pair of functions called $\text{OSNorm012}$ and $\text{OUNorm012}$, with signatures $\text{OpSem} \rightarrow \text{Boolean}$ and $\text{OpSUnit} \rightarrow \text{Boolean}$ respectively, so that any object is said to be in second normal form if the appropriate one of these conditions holds for it, where the conditions are defined as

\[
\text{OSNorm012} (\text{ou}) \equiv \text{OUNorm01} (\text{ou}) \land \text{OSNorm2} (\text{ou})
\]

\[
\text{OUNorm012} (\text{os}) \equiv \text{OSNorm01} (\text{os}) \land \text{OSNorm2} (\text{os})
\]

From these definitions, we can then establish equivalents of theorems 4 and 6 for these two strict normal forms. For the first normal form these equivalents are the following.

**Theorem 8.**

\[
\text{OUNorm1} (\text{EpsSem}) = \text{true}
\land
\text{OUNorm1} (\text{PhiSem}) = \text{true}
\]
∧ ∀ a : PA ⋅ OUNorm1 (FinalActSem (a)) = true
∧ ∀ a : PA ⋅ OUNorm1 (FinalAbActSem (a)) = true.

Proof.
The proofs follow directly by calculation from the definition of OUNorm1, as follows.
OUNorm1 (EpsSem) = (true ⇒ true) ∧ true ∧ true = true
OUNorm1 (PhiSem) = (true ⇒ true) ∧ true ∧ true = true
∀ a : PA ⋅ OUNorm1 (FinalActSem (a)) = (false ⇒ true) ∧ true ∧ true = true
∀ a : PA ⋅ OUNorm1 (FinalAbActSem (a)) = (false ⇒ true) ∧ true ∧ true = true

Theorem 9.
∀ a : PA, os : OpSem | (#os > 1) ∨ ((#os = 1) ∧ (∃ ou : OpSUnit | os = {ou} ⋅ ou.DoesAct) ⋅
OSNorm0 (os) ∧ OSNorm1 (os) ⇒ OUNorm1 (ContActSem (a, os)).

Proof.
As for theorem 6, the proof is directly by induction over the heights of the forests that comprise os. Hence, the base case is where os is constructed entirely from elements ou : OpSUnit whose height is 1, where the condition OSNorm0 (os) implies that the property OUNorm0 (ou) holds, and since all such elements can be constructed as in theorem 5, this base case holds directly from theorem 8.

The induction hypothesis is then that, for any natural number n ≥ 1, the theorem holds for all os : OpSem satisfying the conditions:
(#os > 1) ∨ ((#os = 1) ∧ (∃ ou : OpSUnit | os = {ou} ⋅ ou.DoesAct),
all elements of os have height ≤ n, and
at least one element of os has height n.
and the induction step is to show that, under this hypothesis, it must also hold for any element (ContActSem (a, os)), which, from these conditions, will therefore have height n + 1.

The proof of this induction step follows directly from:
the definition of HeightOS (os), which under these conditions must evaluate to n,
the induction hypothesis that OSNorm1 (os) = true, and the calculation
OUNorm1 (ContActSem (a, os)) = (false ⇒ false) ∧ (if #os = 1 then true else true fi) ∧ true = true.

Hence, by induction the theorem holds for all values of n ≥ 1, and so holds.

Similarly, for strict second normal form the required theorems are as follows.

Theorem 10.
OUNorm2 (EpsSem) = true
∧ OUNorm2 (PhiSem) = true
∧ ∀ a : PA ⋅ OUNorm2 (FinalActSem (a)) = true
∧ ∀ a : PA ⋅ OUNorm2 (FinalAbActSem (a)) = true.

Proof.
The proof follows directly from the definition of OUNorm2.

Theorem 11.
∀ a : PA, os : OpSem | os ≠ ∅ ⋅ OSNorm2 (os) ⇒ OUNorm2 (ContActSem (a, os)).

Proof.
Again, the proof follows directly from the definition of OUNorm2, and no induction is required.

5. Constructing Objects in First Normal Form

Having defined the normal forms, the next step is to define the functions that will transform objects into these forms. Then the properties of these functions will need to be proved, namely that they both maintain the invariants (i) to (v) by
constructing objects in zeroth normal form, and that they construct objects which are indeed in the appropriate higher normal forms. Essentially this requires a pair of functions to be defined for each strict normal form, where in each pair one function will apply to objects of type \textit{OpSem} and the other to objects of type \textit{OpSUnit}.

The first step in this for first normal form is actually to establish a simple property of this form additional to those given by theorems 8 and 9, which then forms the basis for defining a useful auxiliary function and then for proving the various properties of the functions that transform objects into first normal form. This property is stated as the following theorem.

**Theorem 12.**
\[ \forall \text{ou : OpSUnit} | \text{OUNorm01 (ou)} \n\bullet \text{ou.DoesAct = false} \Rightarrow (\text{ou} = \text{EpsSem}) \lor (\text{ou} = \text{PhiSem}) \].

**Proof.**
\[ \text{OUNorm1 (ou)} \land \text{ou.DoesAct = false} \Rightarrow \text{ou.NextState} \neq \text{continues} \Rightarrow \text{ou.NextState} = \text{normend} \lor \text{ou.NextState} = \text{abnormend} \]
\[ \text{OUNorm0 (ou)} \land \text{ou.NextState} \neq \text{continues} \Rightarrow \text{ou.Rest} = \emptyset \]
\[ \Rightarrow (\text{ou} = \text{EpsSem}) \lor (\text{ou} = \text{PhiSem}) \]

Given this property, then the next step is to introduce an auxiliary function that will behave like \textit{ContActSem}, but that will handle properly the various special cases that give rise to some of the conditions in the statement of theorem 9, so as to construct objects that are in first normal form under a wider range of conditions. This function is therefore called \textit{Cont1ActSem}, and it has the same signature as \textit{ContActSem}, namely \textit{PA} \times \textit{OpSem} \rightarrow \textit{OpSUnit}. Its definition assumes that the parameter \textit{os} will be in zeroth and first normal forms, because this is how it will subsequently be used, and it is

\[
\text{Cont1ActSem} (a, os) \equiv \text{if os} = \emptyset \quad \text{then \ FinalActSem} (a) \\
\quad \text{elsif } \# os = 1 \land (\exists \text{ou : OpSUnit} | os = \{ \text{ou} \} \n\bullet \text{ou} = \text{EpsSem}) \\
\quad \text{then \ FinalActSem} (a) \\
\quad \text{elsif } \# os = 1 \land (\exists \text{ou : OpSUnit} | os = \{ \text{ou} \} \n\bullet \text{ou} = \text{PhiSem}) \\
\quad \text{then \ FinalAbActSem} (a) \\
\quad \text{else \ ContActSem} (a, os) \\
\text{fi}
\]

Then, the equivalent of theorem 9 for this function is stated as follows, where for subsequent convenience we also incorporate the equivalent of theorem 6 as well.

**Theorem 13.**
\[ \forall a : \text{PA}, os : \text{OpSem} | \text{OSNorm01 (os)} \n\bullet \text{OUNorm0 (Cont1ActSem (a, os)}) \land \text{OUNorm1 (Cont1ActSem (a, os))}. \]

**Proof.**
The proof of this follows from an analysis of the various cases for the size and construction of \textit{os} that arise from the conditions of theorem 9, as these are reflected in the definition of \textit{Cont1ActSem}.

Case (i): \# os = 0 \Rightarrow os = \emptyset \Rightarrow \text{Cont1ActSem} (a, os) = \text{FinalActSem} (a)
\[ \Rightarrow \text{OUNorm0 (Cont1ActSem (a, os)}) \land \text{OUNorm1 (Cont1ActSem (a, os))} \] from theorems 4 and 8.

Case (ii): \# os = 1 \Rightarrow \exists \text{ou : OpSUnit} \n\bullet os = \{ \text{ou} \}
There are then two sub-cases of this, depending on the value of \textit{ou}.\textit{DoesAct}, as follows.
\[ \text{ou}.\text{DoesAct} = \text{true} \Rightarrow (\text{ou} \neq \text{EpsSem}) \land (\text{ou} \neq \text{PhiSem}) \Rightarrow \text{Cont1ActSem} (a, os) = \text{ContActSem} (a, os) \]
\[ \Rightarrow \text{OUNorm0 (Cont1ActSem (a, os)}) \land \text{OUNorm1 (Cont1ActSem (a, os))} \] from theorems 6 and 9.
\[ \text{ou}.\text{DoesAct} = \text{false} \Rightarrow (\text{ou} \neq \text{EpsSem}) \lor (\text{ou} \neq \text{PhiSem}) \]
\[ \Rightarrow (\text{Cont1ActSem} (a, os) = \text{FinalActSem} (a)) \lor (\text{Cont1ActSem} (a, os) = \text{FinalAbActSem} (a)) \]
\[ \Rightarrow \text{OUNorm0 (Cont1ActSem (a, os)}) \land \text{OUNorm1 (Cont1ActSem (a, os))} \] from theorems 4 and 8.

Case (iii): \# os > 1 \Rightarrow \text{Cont1ActSem} (a, os) = \text{ContActSem} (a, os)
\[ \Rightarrow \text{OUNorm0 (Cont1ActSem (a, os)}) \land \text{OUNorm1 (Cont1ActSem (a, os))} \] from theorems 6 and 9.

Hence the theorem holds for all cases, and so holds.
The reason why the definition of $\text{Cont1ActSem}$ needs to treat as special cases forests that are singleton sets consisting of either the objects $\text{EpsSem}$ or $\text{PhiSem}$ is that, although they are in zeroth, first and second normal forms, they represent objects that can not perform any action from $\text{PA}$, whereas any other object that is in these normal forms can perform some such action. We therefore describe these two objects as passive, and any other non-null forest as active. The latter is formalised by defining a function called $\text{IsActive}$, which has signature $\text{OpSem} \rightarrow \text{Boolean}$, and is defined as

$$\text{IsActive (os)} \equiv \text{os} \neq \emptyset \land \text{os} \neq \{ \text{EpsSem} \} \land \text{os} \neq \{ \text{PhiSem} \}$$

There are then two important properties of active objects, which are expressed as the following theorems.

**Theorem 14.**

$$\forall \ a : \text{PA}, \ \text{os} : \text{OpSem} \bullet \ \text{IsActive (os)} \Rightarrow \text{Cont1ActSem} (a, \text{os}) = \text{ContActSem} (a, \text{os}).$$

**Proof.**

The proof of this follows directly from the four cases that arise from the definition of $\text{Cont1ActSem}$, where in each case the calculation of the truth values for the two sides of the implication is trivial.

The second property then extends theorem 5 so as to define the canonical set of possible constructions for objects that are in zeroth and first strict normal forms, as follows.

**Theorem 15.**

$$\forall \ \text{ou} : \text{OpSUnit} \mid \text{OUNorm01 (ou)} \bullet \ \text{ou} = \text{EpsSem} \lor \text{ou} = \text{PhiSem} \lor ( \exists \ a : \text{PA} \bullet \ \text{ou} = \text{FinalActSem} (a) \lor \text{ou} = \text{FinalAbActSem} (a) ) \lor ( \exists \ \text{os} : \text{OpSem} \mid \text{IsActive (os)} \bullet \ \text{ou} = \text{ContActSem} (a, \text{os}) ) \).$$

**Proof.**

The proof has two cases, depending on $\text{HeightOU (ou)}$, as follows.

Case (i): $\text{HeightOU (ou)} = 1 \Rightarrow \ \text{ou} = \text{EpsSem} \lor \text{ou} = \text{PhiSem} \lor ( \exists \ a : \text{PA} \bullet \ \text{ou} = \text{FinalActSem} (a) \lor \text{ou} = \text{FinalAbActSem} (a) )$ from theorem 5

Case (ii): $\text{HeightOU (ou)} = 1 \Rightarrow \ \text{ou}.\text{Rest} \neq \emptyset \Rightarrow \ \text{ou}.\text{NextState} = \text{continues}$ since $\text{OUNorm0 (ou)}$

$$\Rightarrow \ \text{ou}.\text{DoesAct} \quad \text{since } \text{OUNorm1 (ou)}$$

$$\Rightarrow \ \text{ou}.\text{TheAct} \in \text{PA} \quad \text{since } \text{OUNorm0 (ou)}$$

$$\Rightarrow \ \exists \ a : \text{PA}, \ \text{os} : \text{OpSem} \mid \text{os} \neq \emptyset \bullet \ \text{ou} = \text{ContActSem} (a, \text{os})$$

There are then two sub-cases, depending on $\# \ \text{os}$, as follows.

Sub-case (ii)(a): $\# \ \text{os} = 1 \Rightarrow \ \exists \ \text{ou1} : \text{OpSUnit} \bullet \ \text{os} = \{ \text{ou1} \}$

$$\text{OUNorm1 (ou)} \Rightarrow \ \text{ou1}.\text{DoesAct} \Rightarrow \ \text{ou1} \neq \text{EpsSem} \land \text{ou1} \neq \text{PhiSem} \Rightarrow \text{IsActive (os)}$$

Sub-case (ii)(b): $\# \ \text{os} > 1 \Rightarrow \ \text{os} \neq \{ \text{EpsSem} \} \land \text{os} \neq \{ \text{PhiSem} \} \Rightarrow \text{IsActive (os)}$

Hence, for both of these sub-cases we have $\text{IsActive (os)}$, and so the theorem follows from the disjunction of the two main cases.

Given this definition of $\text{Cont1ActSem}$ with these properties, then the actual transformation functions for the first normal form can be defined as follows. The function that is applied to objects of type $\text{OpSem}$ has the primary purpose of ensuring that the condition $\text{OUNorm1a}$ will hold for all of its component tree structures, while the function that is applied to objects of type $\text{OpSUnit}$ has the primary purpose of ensuring that condition $\text{OUNorm1b}$ will hold. In addition, of course, each recursively calls the other, to ensure that an entire forest structure is processed.

Because of the way in which the first of these transformations needs to be performed, it sometimes involves replacing a single $\text{OpSUnit}$ object by a set of them. It is therefore convenient to introduce an auxiliary function that merges sets of these objects, so as to construct the union of the individual sets. Rather than give this function a name that reflects the fact that it constructs a union, it is actually called $\text{FlattenOS}$, reflecting the fact that its parameter is an arbitrary sized set of sets. Hence, it has signature $\text{P} \mid \text{OpSem} \rightarrow \text{OpSem}$, and is defined as

$$\text{FlattenOS (oss)} \equiv \{ \forall \ \text{os} : \text{OpSem}, \ \text{ou} : \text{OpSUnit} \mid \text{os} \in \text{oss} \land \text{ou} \in \text{os} \bullet \text{ou} \}$$
Then, the two normalisation functions to produce objects in first normal form are called \( \text{NormOS1} \) and \( \text{NormOU1} \), and their signatures are respectively \( \text{OpSem} \rightarrow \text{OpSem} \) and \( \text{OpSUnit} \rightarrow \text{OpSUnit} \). As already indicated, the first of them primarily performs the transformation of replacing any object, that does no action but continues, by the set of objects representing the rest of the computation with which it continues, and it is defined as follows.

\[
\text{NormOS1} (os) \equiv \text{FlattenOS} ( \{ \forall \text{ou} : \text{OpSUnit} \mid \text{ou} \in os \bullet \text{NormOS1Unit} (\text{ou}) \} )
\]

Where the function \( \text{NormOS1Unit} \) has signature \( \text{OpSUnit} \rightarrow \text{OpSem} \), and is defined as follows.

\[
\text{NormOS1Unit} (\text{ou}) \equiv \begin{cases} 
\text{if} \ \text{ou}.\text{DoesAct} \ \text{then} \ \{ \text{NormOU1} (\text{ou}) \} \\
\text{elseif} \ \text{ou}.\text{NextState} = \text{continues} \ \text{then} \ \text{NormOS1} (\text{ou}.\text{Rest}) \\
\text{else} \ \{ \ \text{ou} \}
\end{cases}
\]

Similarly, the second of these functions primarily performs the transformation of removing any one-element set that represents the rest of a computation, if all this does is terminate without performing an action. Hence, \( \text{NormOU1} \) is defined as follows.

\[
\text{NormOU1} (\text{ou}) \equiv \begin{cases} 
\text{if} \ \text{ou}.\text{Rest} = \emptyset \ \text{then} \ \text{ou} \\
\text{else} \ \text{Cont1ActSem} (\text{ou}.\text{TheAct}, \text{NormOS1} (\text{ou}.\text{Rest}) )
\end{cases}
\]

Here there are two features of these definitions that need to be noted. The first is that all three of them implicitly rely on the objects to which they are applied being in zeroth normal form, and so any properties of the results produced by these functions have to be conditional on the parameters to the functions being in this form. The second feature is the way in which the definitions of \( \text{NormOS1} \) and \( \text{NormOU1} \) interact, in that the definition of \( \text{NormOU1} \) relies for part of its structure on the way in which it is called from within \( \text{NormOS1Unit} \), since this ensures that \( \text{ou}.\text{DoesAct} = \text{true} \).

There are then two properties that need to be established for the pair of functions \( \text{NormOS1} \) and \( \text{NormOU1} \). The first property is that the objects produced by them are in zeroth normal form, and so any properties of the results produced by these functions have to be conditional on the parameters to the functions being in this form. The second property is then that these objects are indeed in strict first normal form. Each of these properties is expressed as a theorem, which has two alternative statements, where the theorem for the maintenance of zeroth normal form is as follows.

**Theorem 16.**

\[
\forall \text{os} : \text{OpSem}, \text{ou}' : \text{OpSUnit} \mid \text{ou}' \in \text{NormOS1} (\text{os}) \bullet \text{OSNorm0} (\text{os}) \Rightarrow \text{OUNorm0} (\text{ou}')
\]

or, alternatively

\[
\forall \text{os} : \text{OpSem} \bullet \text{OSNorm0} (\text{os}) \Rightarrow \text{OSNorm0} (\text{NormOS1} (\text{os}))
\]

where the alternative form follows directly from the first form and the definition of \( \text{OSNorm0} \) in terms of \( \text{OUNorm0} \).

**Proof.**
The proof focuses on the first form of the theorem, and it involves an induction over the height of the forest \( \text{os} \), within which it uses case analysis of the two functions \( \text{NormOS1} \) and \( \text{NormOU1} \).

For any arbitrary value \( \text{os} : \text{OpSem} \), and for any arbitrary value \( \text{ou}' : \text{OpSUnit} \) such that \( \text{ou}' \in \text{NormOS1} (\text{os}) \), the operation of \( \text{NormOS1} (\text{os}) \) is such that \( \text{ou}' \) will have been derived by \( \text{NormOS1Unit} \) from a single element of \( \text{os} \) (and for any such element of \( \text{os} \) there may be more than one such \( \text{ou}' \), but this does not affect the argument). Let this single element of \( \text{os} \) that produces \( \text{ou}' \) be denoted \( \text{ou} : \text{OpSUnit} \), so that \( \text{ou} \in \text{os} \).

Furthermore, the operation of \( \text{FlattenOS} \) is such that this value \( \text{ou}' \) will have been produced from a single element of one of the sets that are produced by \( \text{NormOS1Unit} \) and passed as parameters to \( \text{FlattenOS} \). Let this parameter element be denoted \( \text{oue} : \text{OpSUnit} \), and then the operation of \( \text{FlattenOS} \) ensures that \( \text{ou}' = \text{oue} \).

**Base case.**
The base case is that

\[
\text{HeightOS} (\text{os}) = 1 \Rightarrow \text{HeightOU} (\text{ou}) = 1
\]

From the definition of \( \text{OSNorm0} \), it then follows that

\[
\text{OSNorm0} (\text{os}) \Rightarrow \text{OUNorm0} (\text{ou})
\]
Hence, from theorem 5 there are four sub-cases for the construction of ou, and a common feature of all of these sub-cases is that, from the definition of NormOU1,
\[ \text{ou}.\text{Rest} = \emptyset \Rightarrow \text{NormOU1}(\text{ou}) = \text{ou} \]

Another common feature of these four sub-cases is that, because ou.NextState \(\neq\) continues, each element ou gives rise to exactly one element oue, and hence to exactly one element ou'.

Then, the four sub-cases for the construction of ou are as follows.

Base sub-case (i): \(\text{ou} = \text{EpsSem} \Rightarrow \text{oue} = \text{ou} \)
\[ \Rightarrow \text{OUNorm0}(\text{oue}) = \text{true} \quad \text{from theorem 4} \]
\[ \Rightarrow \text{OUNorm0}(\text{ou'}) = \text{true} \]

Base sub-case (ii): \(\exists a : \text{PA} \bullet \text{ou} = \text{FinalActSem}(a) \)
\[ \Rightarrow \text{oue} = \text{NormOU1}(\text{ou}) \quad \text{from the definition of NormOU1} \]
\[ \Rightarrow \text{oue} = \text{FinalActSem}(a) \quad \text{from the definition of NormOU1} \]
\[ \Rightarrow \text{OUNorm0}(\text{oue}) = \text{true} \quad \text{from theorem 4} \]
\[ \Rightarrow \text{OUNorm0}(\text{ou'}) = \text{true} \]

Base sub-case (iii): \(\text{ou} = \text{PhiSem} \Rightarrow \text{oue} = \text{ou} \)
\[ \Rightarrow \text{OUNorm0}(\text{oue}) = \text{true} \quad \text{from theorem 4} \]
\[ \Rightarrow \text{OUNorm0}(\text{ou'}) = \text{true} \]

Base sub-case (iv): \(\exists a : \text{PA} \bullet \text{ou} = \text{FinalAbActSem}(a) \)
\[ \Rightarrow \text{oue} = \text{NormOU1}(\text{ou}) \quad \text{from the definition of NormOU1} \]
\[ \Rightarrow \text{oue} = \text{FinalAbActSem}(a) \quad \text{from the definition of NormOU1} \]
\[ \Rightarrow \text{OUNorm0}(\text{oue}) = \text{true} \quad \text{from theorem 4} \]
\[ \Rightarrow \text{OUNorm0}(\text{ou'}) = \text{true} \]

Hence, the base case of the induction follows directly from the combination of these four sub-cases.

**Inductive case.**
For any natural number \(n > 1\), the induction hypothesis is that the theorem holds for all values \(\text{os}' : \text{OpSem}\) such that \(\text{HeightOS}(\text{os}') < n\), and this hypothesis applies to both forms of the theorem, although it is actually the second form that is mainly used in the proof below. The induction step is then to show that the theorem must also hold for any arbitrary value \(\text{os} : \text{OpSem}\) such that \(\text{HeightOS}(\text{os}) = n\), where the structure of the argument here is based on the first form of the theorem.

Without loss of generality, suppose that \(\text{os}\) is such that \(\text{HeightOS}(\text{os}) = n\), so that by the definition of \(\text{HeightOS}\), \(\text{HeightOU}(\text{ou}) \leq n\). Also without loss of generality, assume that \(\text{HeightOU}(\text{ou}) > 1\), since otherwise the problem will simply reduce to the base case.

Then, let \(\text{os1} : \text{OpSem} = \text{ou}.\text{Rest}\), so that \(\text{os1} \neq \emptyset\) and hence \(1 \leq \text{HeightOS}(\text{os1}) < n\), and let \(\text{ou1} : \text{OpSUnit}\) be any arbitrary value such that \(\text{ou1} \in \text{os1}\).

From the definition of \(\text{OSNorm0}\), it therefore follows that
\[ \text{OSNorm0}(\text{os}) \Rightarrow \text{OUNorm0}(\text{ou}) \Rightarrow \text{OSNorm0}(\text{os1}) \Rightarrow \text{OUNorm0}(\text{ou1}) \]
and, from the induction hypothesis, since \(\text{HeightOS}(\text{os1}) < n\), that
\[ \text{OSNorm0}(\text{NormOS1}(\text{os1})) = \text{true} \]

Unlike the base case, in this inductive case the possibility does arise that a single element ou may give rise to more than one element oue, and hence to more than one element ou'. For any such single element ou there are three sub-cases that arise from the structure of \(\text{NormOS1Unit}\), as follows.

Inductive sub-case (i): \(\text{ou}.\text{DoesAct} = \text{true} \Rightarrow \text{oue} = \text{NormOU1}(\text{ou}) \)
\[ \Rightarrow \text{oue} = \text{Cont1ActSem}(\text{ou}.\text{TheAct}, \text{NormOS1}(\text{os1})) \quad \text{since os1} \neq \emptyset \]

From the induction hypothesis,
\[ \text{OSNorm0}(\text{NormOS1}(\text{os1})) = \text{true} \]
\[ \Rightarrow \text{OUNorm0}(\text{oue}) = \text{true} \quad \text{from theorem 13} \]
\[ \Rightarrow \text{OUNorm0}(\text{ou'}) = \text{true} \]

and this applies to any such elements oue and ou', and so the theorem holds for this sub-case.
Inductive sub-case (ii): \( \text{ou}.\text{DoesAct} = \text{false} \land \text{ou}.\text{NextState} = \text{continues} \)
\[ \Rightarrow \text{ou}_e \in \text{NormOS1} (\text{os1}) \]
\[ \Rightarrow \text{ou}' \in \text{NormOS1} (\text{os1}) \]

But, as noted above, from the induction hypothesis \( \text{OSNorm0} (\text{NormOS1} (\text{os1})) = \text{true} \). Hence, for this sub-case \( \text{OUNorm0} (\text{ou}') = \text{true} \), for any such elements \( \text{ou}_e \) and \( \text{ou}' \), and so the theorem holds for this sub-case.

Inductive sub-case (iii): \( \text{ou}.\text{DoesAct} = \text{false} \land \text{ou}.\text{NextState} \neq \text{continues} \)

In this case we would have
\[ \text{OUNorm0} (\text{ou}) \Rightarrow \text{os1} = \emptyset \]

But since we have assumed that \( \text{HeightOU} (\text{ou}) > 1 \) in order to avoid reducing to the base case, this sub-case (which was derived purely from the structure of \( \text{NormOS1Unit} \)) can not arise as part of the inductive case, and therefore does not need to be analysed further here.

Hence the inductive case follows from the combination of these two applicable sub-cases, and so by induction the theorem holds for all values of \( n \geq 1 \), and so holds.

\[ \blacksquare \]

The corresponding theorem for the second property that is required for the pair of functions \( \text{NormOS1} \) and \( \text{NormOU1} \), namely that the objects that they produce are in strict first normal form, has an almost identical structure, as does its proof. This theorem is as follows.

**Theorem 17.**
\[ \forall \text{os} : \text{OpSem}, \text{ou}' : \text{OpSUnit} \mid \text{ou}' \in \text{NormOS1} (\text{os}) \Rightarrow \text{OUNorm1} (\text{ou}') \]
or, alternatively
\[ \forall \text{os} : \text{OpSem} \Rightarrow \text{OSNorm0} (\text{os}) \Rightarrow \text{OSNorm1} (\text{NormOS1} (\text{os})) \]

where again the alternative form follows directly from the first form and the definition of \( \text{OSNorm1} \) in terms of \( \text{OUNorm1} \).

**Proof.**

Again, the proof focuses on the first form of the theorem, and involves an induction over the height of the forest \( \text{os} \), within which it uses case analysis of the two functions \( \text{NormOS1} \) and \( \text{NormOU1} \). As in the proof of theorem 16, let the single element of \( \text{os} \) that produces \( \text{ou}' \) be denoted \( \text{ou} : \text{OpSUnit} \), so that \( \text{ou} \in \text{os} \), and let the parameter element to \( \text{FlattenOS} \) that produces \( \text{ou}' \) be denoted \( \text{ou}_e : \text{OpSUnit} \), where again the operation of \( \text{FlattenOS} \) ensures that \( \text{ou}' = \text{ou}_e \).

**Base case.**

The base case is that
\[ \text{HeightOS} (\text{os}) = 1 \Rightarrow \text{HeightOU} (\text{ou}) = 1 \]
and from the definition of \( \text{OSNorm0} \), it then follows that
\[ \text{OSNorm0} (\text{os}) \Rightarrow \text{OUNorm0} (\text{ou}) \]
so that again from theorem 5 there are four sub-cases for the construction of \( \text{ou} \), with the common features that, from the definition of \( \text{NormOU1} \),
\[ \text{ou}.\text{Rest} = \emptyset \Rightarrow \text{NormOU1} (\text{ou}) = \text{ou} \]
and that, because \( \text{ou}.\text{NextState} \neq \text{continues} \), each element \( \text{ou} \) gives rise to exactly one element \( \text{ou}_e \), and hence to exactly one element \( \text{ou}' \). These four sub-cases for the construction of \( \text{ou} \) are then as follows.

Base sub-case (i): \( \text{ou} = \text{EpsSem} \Rightarrow \text{ou} = \text{ou} \)
\[ \Rightarrow \text{OUNorm1} (\text{ou}) = \text{true} \]
\[ \Rightarrow \text{OUNorm1} (\text{ou}') = \text{true} \]
from the definition of \( \text{NormOS1Unit} \)
from theorem 8

Base sub-case (ii): \( \exists a : \text{PA} \cdot \text{ou} = \text{FinalActSem} (a) \)
\[ \Rightarrow \text{ou} = \text{NormOU1} (\text{ou}) \]
\[ \Rightarrow \text{ou} = \text{FinalActSem} (a) \]
\[ \Rightarrow \text{OUNorm1} (\text{ou}) = \text{true} \]
\[ \Rightarrow \text{OUNorm1} (\text{ou}') = \text{true} \]
from the definition of \( \text{NormOS1Unit} \)
from the definition of \( \text{NormOU1} \)
from theorem 8

Base sub-case (iii): \( \text{ou} = \text{PhiSem} \Rightarrow \text{ou} = \text{ou} \)
\[ \Rightarrow \text{OUNorm1} (\text{ou}) = \text{true} \]
\[ \Rightarrow \text{OUNorm1} (\text{ou}') = \text{true} \]
from the definition of \( \text{NormOS1Unit} \)
from theorem 8
Base sub-case (iv): \( \exists \ a : PA \bullet ou = \text{FinalAbActSem} (a) \)
\[
\Rightarrow oue = \text{NormOU1} (ou) \quad \text{from the definition of NormOS1Unit}
\]
\[
\Rightarrow oue = \text{FinalAbActSem} (a) \quad \text{from the definition of NormOU1}
\]
\[
\Rightarrow \text{OUNorm1} (oue) = \text{true} \quad \text{from theorem 8}
\]
\[
\Rightarrow \text{OUNorm1} (ou') = \text{true}
\]

Hence, the base case of the induction follows directly from the combination of these four sub-cases.

**Inductive case.**

As in the proof of theorem 16, the induction hypothesis is that, for any natural number \( n > 1 \), the theorem holds for all values \( os' : \text{OpSem} \) such that \( \text{HeightOS} (os') < n \), and this hypothesis applies to both forms of the theorem, although the second form is the one that is mainly used in the proof. Again, the induction step is to show from this that the theorem must also hold for any arbitrary value \( os : \text{OpSem} \) such that \( \text{HeightOS} (os) = n \), where the structure of the argument is based on its first form.

Therefore, again without loss of generality, suppose firstly that \( os \) is such that \( \text{HeightOS} (os) = n \), so that by the definition of \( \text{HeightOS} \), \( \text{HeightOU} (ou) \leq n \), and also suppose that \( \text{HeightOU} (ou) > 1 \), to avoid the problem simply reducing to the base case.

Then, let \( os1 : \text{OpSem} = ou.\text{Rest} \), so that \( os1 \neq \emptyset \) and hence \( 1 \leq \text{HeightOS} (os1) < n \), and let \( ou1 : \text{OpSUnit} \) be any arbitrary value such that \( ou1 \in os1 \).

From the definition of \( \text{OSNorm0} \), it therefore follows again that
\[
\text{OSNorm0} (os) \Rightarrow \text{OUNorm0} (ou) \Rightarrow \text{OSNorm0} (os1) \Rightarrow \text{OUNorm0} (ou1)
\]
and also, since \( \text{HeightOS} (os1) < n \), it follows from the induction hypothesis that
\[
\text{OSNorm0} (\text{NormOS1} (os1)) = \text{true}
\]

Again, as in the proof of theorem 16, the possibility arises here that a single element \( ou \) may give rise to more than one element \( oue \), and hence to more than one element \( ou' \). And also, for any single element \( ou \) there are the same three sub-cases that arise from the structure of \( \text{NormOS1Unit} \), as follows.

Inductive sub-case (i): \( ou.\text{DoesAct} = \text{true} \Rightarrow oue = \text{NormOU1} (ou) \)
\[
\Rightarrow oue = \text{Cont1ActSem} (ou.\text{TheAct}, \text{NormOS1} (os1)) \quad \text{since } os1 \neq \emptyset
\]

From the induction hypothesis,
\[
\text{OSNorm1} (\text{NormOS1} (os1)) = \text{true}
\]
\[
\Rightarrow \text{OUNorm1} (oue) = \text{true} \quad \text{from theorem 13}
\]
and this applies to any such elements \( oue \) and \( ou' \), and so the theorem holds for this sub-case.

Inductive sub-case (ii): \( ou.\text{DoesAct} = \text{false} \land ou.\text{NextState} = \text{continues} \)
\[
\Rightarrow oue \in \text{NormOS1} (os1)
\]
\[
\Rightarrow ou' \in \text{NormOS1} (os1)
\]

But, since \( \text{HeightOS} (os1) < n \), by the induction hypothesis \( \text{OSNorm1} (\text{NormOS1} (os1)) = \text{true} \). Hence, for this sub-case \( \text{OUNorm1} (ou') = \text{true} \), for any such elements \( oue \) and \( ou' \), and so the theorem holds for this sub-case.

Inductive sub-case (iii): \( ou.\text{DoesAct} = \text{false} \land ou.\text{NextState} \neq \text{continues} \)

In this case we would have
\[
\text{OUNorm0} (ou) \Rightarrow os1 = \emptyset
\]
But, as in the corresponding sub-case in the proof of theorem 16, since we have assumed that \( \text{HeightOU} (ou) > 1 \) in order to avoid reducing to the base case, this sub-case can not arise as part of the inductive case, and therefore does not need to be analysed further here.

Hence the inductive case again follows from the combination of these two applicable sub-cases, and so by induction the theorem holds for all values of \( n \geq 1 \), and so holds.

\[ \square \]
Finally, there is an obvious but useful property that ought to be established formally, which is that if a tree or forest is already in first normal form, then applying the relevant normalisation function to it will not change it. This property is expressed as the following theorem.

**Theorem 18.**

\[
\forall \text{os} : \text{OpSem} | \text{OSNorm01 (os)} \bullet \text{NormOS1 (os)} = \text{os} \\
\land \forall \text{ou} : \text{OpSUnit} | \text{OUNorm01 (ou)} \bullet \text{NormOU1 (ou)} = \text{ou}.
\]

**Proof.**

This proof has a similar overall structure to the proofs of theorems 16 and 17, in that it too is by induction over the heights of the forests \(\text{os}\) and the trees \(\text{ou}\), and within this it uses case analysis of the two functions \(\text{NormOS1}\) and \(\text{NormOU1}\).

The structure of the individual cases is simpler, though, as it is sufficient to work forwards for any object \(\text{os}\) from each individual component \(\text{ou}\), rather than having to work backwards from the elements of \(\text{NormOS1 (os)}\).

**Base case.**

The base case is that

\[\text{HeightOS (os)} = 1 \implies \text{HeightOU (ou)} = 1\]

and from the definition of \(\text{OSNorm0}\), it then follows that

\[\text{OSNorm0 (os)} \implies \text{OUNorm0 (ou)}\]

so that again from theorem 5 there are four sub-cases for the construction of \(\text{ou}\), viz:

\[\text{ou} = \text{EpsSem} \lor \text{ou} = \text{PhiSem} \lor (\exists \text{a : PA} \bullet \text{ou} = \text{FinalActSem (a)} \lor \text{ou} = \text{FinalAbActSem (a)})\]

The common feature of all of these sub-cases is that, from the definition of \(\text{NormOU1}\),

\[\text{ou.Rest} = \emptyset \implies \text{NormOU1 (ou)} = \text{ou}\]

Hence, the second half of the theorem holds directly for the base case, and so the first half follows directly from the definition of \(\text{NormOS1}\).

**Inductive case.**

As in the proofs of theorems 16 and 17, the induction hypothesis is that, for any natural number \(n > 1\), the theorem holds for all values \(\text{os'} : \text{OpSem}\) such that \(\text{HeightOS (os')} < n\), and as in the proof of theorem 17 this hypothesis applies to both forms of the theorem, although the second form is the one that is mainly used in the proof. Again, the induction step is to show from this that the theorem must also hold for any arbitrary value \(\text{os} : \text{OpSem}\) such that \(\text{HeightOS (os)} = n\), where the structure of the argument is based on its first form.

Therefore, again without loss of generality, suppose firstly that \(\text{os}\) is such that \(\text{HeightOS (os)} = n\), so that by the definition of \(\text{HeightOS}\), \(\text{HeightOU (ou)} \leq n\), and also suppose that \(\text{HeightOU (ou)} > 1\), to avoid the problem simply reducing to the base case.

Then, let \(\text{os1 : OpSem} = \text{ou.Rest}\), so that \(\text{os1} \neq \emptyset\) and hence \(1 \leq \text{HeightOS (os1)} < n\). It then follows that

\[\text{OUNorm1 (ou)} \implies \text{OSNorm1 (os1)} \implies \text{NormOS1 (os1)} = \text{os1}\]

from the induction hypothesis,

and also that

\[\text{OUNorm1 (ou)} \land \text{os1} \neq \emptyset \implies \text{ou.DoesAct}\]

Because of the conditions in \(\text{Cont1ActSem}\) there are two sub-cases to be considered, depending on the size of \(\text{os1}\), as follows.

**Sub-case (i): \# os1 = 1 \land \exists \text{ou1 : OpSUnit} \bullet \text{os1} = \{ \text{ou1} \},**

so that

\[\text{OSNorm1 (os1)} \implies \text{ou1.DoesAct}\]

\[\implies \text{NormOU1 (ou)} = \text{Cont1ActSem (ou.TheAct, NormOS1 (os1))}\]

\[\implies \text{NormOU1 (ou)} = \text{Cont1ActSem (ou.TheAct, ou.Rest)}\]

\[\implies \text{NormOU1 (ou)} = \text{ContActSem (ou.TheAct, ou.Rest)}\]

\[\implies \text{NormOU1 (ou)} = \text{ou}\]

**Sub-case (ii): \# os1 > 1, where it follows directly that**

\[\text{NormOU1 (ou)} = \text{Cont1ActSem (ou.TheAct, NormOS1 (os1))}\]

\[\implies \text{NormOU1 (ou)} = \text{Cont1ActSem (ou.TheAct, ou.Rest)}\]

\[\implies \text{NormOU1 (ou)} = \text{ContActSem (ou.TheAct, ou.Rest)}\]

\[\implies \text{NormOU1 (ou)} = \text{ou}\]
Hence, the second half of the theorem holds for both of these sub-cases, and so holds for the inductive case, and hence the first half of the theorem holds for the inductive case too, again directly from the definition of NormOS1. By induction, therefore, the theorem holds for all values of \( n \geq 1 \), and so holds.

6. Constructing Objects in Second Normal Form

The functions that transform objects into second normal form are defined in a similar fashion to those that transform them into first normal form, in that in principle there is a pair of such functions, one applying to objects of type \( \text{OpSem} \) and the other applying to objects of type \( \text{OpSUnit} \). In practice, though, the function that applies to objects of type \( \text{OpSem} \) has to be defined in two stages, that correspond roughly to the two parts of the normalisation condition.

Thus, the first stage is to define the function NormOS2, with signature \( \text{OpSem} \rightarrow \text{OpSem} \), which primarily transforms a set of elements by merging any two that perform the same action. This is defined as

\[
\text{NormOS2} (os) = \begin{cases} 
\text{if } \exists \text{ou1, ou2 : OpSUnit } | \text{ou1 } \in \text{os } \land \text{ou2 } \in \text{os } \land \text{ou1 } \neq \text{ou2} \\
\text{then NormOS2 } ( ((\text{os } - \text{ou1}) - \text{ou2}) \cup \text{MergeOU (ou1, ou2)}) \\
\text{else NormOS2b (os)} 
\end{cases}
\]

In defining this and the other functions, the assumption is made that objects to be put into second normal form will already be in zeroth and first strict normal forms, so that where appropriate the definitions can implicitly rely on these properties. Thus, the function MergeOU has signature \( \text{OpSUnit} \times \text{OpSUnit} \rightarrow \text{OpSUnit} \), and the way in which it is called from within NormOS2 guarantees that, for any pair of parameters \( \text{ou1} \) and \( \text{ou2} \), \( \text{ou1} \neq \text{ou2} \land \text{ou1.TheAct } \equiv \text{ou2.TheAct} \).

On the other hand, while this and the assumption of zeroth normal form also implies that \( \text{ou1.DoesAct } = \text{ou2.DoesAct} \) (so that it is only the next state and descendents that differ), no assumption is made about whether they are both \( \text{true} \) or both \( \text{false} \), and so \( \text{EpsSem} \) and \( \text{PhiSem} \) would be a legitimate pair of parameters to this function. Thus, while in principle the function simply merges the sets of descendents of the two parameter objects, in practice there are a number of cases that need to be recognised and dealt with specially, as discussed below. The actual definition of this function is therefore as follows.

\[
\text{MergeOU (ou1, ou2)} = \begin{cases} 
\text{if } \text{ou1.NextState } = \text{continues} \\
\text{then } \text{if } \text{ou2.NextState } = \text{continues} \\
\text{then } \text{ContActSem (ou1.TheAct, ou1.Rest } \cup \text{ou2.Rest)} \\
\text{elseif } \text{ou2.NextState } = \text{normend} \\
\text{then } \text{ContActSem (ou1.TheAct, ou1.Rest } \cup \text{\{ EpsSem \})} \\
\text{else ou1} \\
\text{fi} \\
\text{elsif } \text{ou1.NextState } = \text{normend} \\
\text{then if } \text{ou2.NextState } = \text{continues} \\
\text{then } \text{ContActSem (ou1.TheAct, ou2.Rest } \cup \text{\{ EpsSem \})} \\
\text{else ou1} \\
\text{fi} \\
\text{else ou2} \\
\text{fi}
\end{cases}
\]

The second stage in defining the function NormOS2 is then represented by the function NormOS2b, which has signature \( \text{OpSem} \rightarrow \text{OpSem} \). This transforms a set of elements by eliminating any element that simply terminates abnormally without performing any action, since it is possible that such an element might be introduced by other transformations, even though (as noted above) the guards on the transitions for alternations mean that such elements will not occur in derivation sequences. The function is therefore defined as

\[
\text{NormOS2b (os)} = \begin{cases} 
\text{if } \text{os } = \emptyset \\
\text{then } \emptyset \\
\text{elsif } (\# \text{os } = 1) \land (\exists \text{ou : OpSUnit } \bullet \text{ou } \in \text{os}) \\
\text{then } \{ \text{NormOU2 (ou)} \} \\
\text{else } \{ \forall \text{ou : OpSUnit } | \text{ou } \in \text{os } \land \text{ou } \neq \text{PhiSem } \bullet \text{NormOU2 (ou)} \} \\
\text{fi}
\end{cases}
\]
Here, the function \( \text{NormOU2} \) that is invoked is the other one of the required pair, and has signature \( \text{OpSUnit} \rightarrow \text{OpSUnit} \), and in terms of the individual tree structures it simply applies \( \text{NormOS2} \) recursively to its descendant forest.

For this purpose, though, having to handle the possibility of objects that are not in zeroth or first normal form would complicate the definition significantly, and this complication is not really necessary, since in practice this function and \( \text{NormOS2} \) will only ever need to be applied to objects that are in these forms. Hence, to simplify this definition the assumption is made that its parameter \( \text{ou} \) must be in both zeroth and first strict normal forms, meaning that if it continues then it must do an action, so that the function can then be defined as

\[
\text{NormOU2} \ (\text{ou}) \equiv \begin{cases} 
\text{if} \ (\text{ou}.\text{Rest} = \emptyset) \\
\text{then} \ \text{ou} \\
\text{else} \ \text{Cont1ActSem} \ (\text{ou}.\text{TheAct}, \text{NormOS2} \ (\text{ou}.\text{Rest}) ) \\
\end{cases}
\]

As noted above, a feature of these definitions is that \( \text{MergeOU} \) needs to handle certain cases specially, and in particular these cases include the ones where either of the parameters \( \text{ou1} \) or \( \text{ou2} \) are constructed as \( \text{FinalActSem} \ (a) \) for some action \( a \). The nature of this special case is that such an object is effectively replaced by one that has the same action, but a single descendent which is constructed as \( \text{EpsSem} \). Such an object would not be in first normal form, but this does not matter, as the descendent object \( \text{EpsSem} \) is then merged with the set of descendents from the other parameter, which from the preconditions for the function can not simply consist of \( \text{EpsSem} \), and so this merging must create a forest that is in first normal form. This is why it is sufficient to use \( \text{ContActSem} \) here in constructing the object with the merged descendents, rather than \( \text{Cont1ActSem} \), because the cases that would need to be handled specially by \( \text{Cont1ActSem} \) will not arise anyway.

It might be supposed that a similar approach should also be taken if either \( \text{ou1} \) or \( \text{ou2} \) were constructed instead as \( \text{FinalAbActSem} \ (a) \) for some action \( a \), but this is not necessary, as it would simply incorporate into the merged set of descendents an object \( \text{PhiSem} \) that would then have to be removed again by call of \( \text{NormOS2b} \) within \( \text{NormOS2} \). Of course, \( \text{NormOS2b} \) has to be capable of removing such elements, but introducing them unnecessarily would complicate the arguments needed in the proofs of the various properties of the functions. In particular, there is one case of removing such an element that could require special treatment, and this is the tree constructed as

\[
\text{ContActSem} \ (a, \ \{ \text{EpsSem}, \Phi\text{Sem} \})
\]

for some action \( a \). This tree is in first normal form, but the application of \( \text{NormOU2} \) to it will result in \( \text{NormOS2} \) being applied to the object \( \{ \text{EpsSem}, \Phi\text{Sem} \} \), to give the object \( \{ \text{EpsSem} \} \). Hence, to ensure that \( \text{NormOU2} \) will in this case create an object that is in first normal form, it does need to use \( \text{Cont1ActSem} \) rather than simply \( \text{ContActSem} \) in order to construct its result.

The other important feature of these definitions is that they implicitly assume that applications of \( \text{MergeOU} \) can be made in any arbitrary order, and so they rely on this function being both commutative and associative. These properties, together with the associated one of idempotence, are established formally as the following three theorems, of which the first defines the commutativity property.

**Theorem 19.**
\[
\forall \ \text{oua}, \ \text{oub} : \text{OpSUnit} \mid \text{OUNorm01} \ (\text{oua}) \land \text{OUNorm01} \ (\text{oub}) \land \text{oua}.\text{TheAct} = \text{oub}.\text{TheAct} \bullet \\
\text{MergeOU} \ (\text{oua}, \ \text{oub}) = \text{MergeOU} \ (\text{oub}, \ \text{oua}).
\]

**Proof.**
There are nine cases to be considered, corresponding to the different combinations of \( \text{ou1}.\text{NextState} \) and \( \text{ou2}.\text{NextState} \) that occur in the definition of \( \text{MergeOU} \). The fact that combinations involving \( \text{ou1}.\text{NextState} = \text{continues} \) or \( \text{ou2}.\text{NextState} = \text{continues} \) can only occur if \( \text{ou1}.\text{TheAct} \in \text{PA} \) does not affect the structure of these cases, which are as follows.

(i): \( \text{oua}.\text{NextState} = \text{continues} \land \text{oub}.\text{NextState} = \text{continues} \)
\[
\Rightarrow \text{MergeOU} \ (\text{oua}, \ \text{oub}) = \text{ContActSem} \ (\text{oua}.\text{TheAct}, \ \text{oua}.\text{Rest} \cup \ \text{oub}.\text{Rest}) \\
\land \text{MergeOU} \ (\text{oub}, \ \text{oua}) = \text{ContActSem} \ (\text{oub}.\text{TheAct}, \ \text{oub}.\text{Rest} \cup \ \text{oua}.\text{Rest}) \\
= \text{ContActSem} \ (\text{oua}.\text{TheAct}, \ \text{oua}.\text{Rest} \cup \ \text{oub}.\text{Rest})
\]

(ii): \( \text{oua}.\text{NextState} = \text{continues} \land \text{oub}.\text{NextState} = \text{normend} \)
\[
\Rightarrow \text{MergeOU} \ (\text{oua}, \ \text{oub}) = \text{ContActSem} \ (\text{oua}.\text{TheAct}, \ \text{oua}.\text{Rest} \cup \ \{ \text{EpsSem} \} ) \\
\land \text{MergeOU} \ (\text{oub}, \ \text{oua}) = \text{ContActSem} \ (\text{oub}.\text{TheAct}, \ \text{oub}.\text{Rest} \cup \ \{ \text{EpsSem} \} )
\]

(iii): \( \text{oua}.\text{NextState} = \text{continues} \land \text{oub}.\text{NextState} = \text{abnormend} \)
\[
\Rightarrow \text{MergeOU} \ (\text{oua}, \ \text{oub}) = \text{oua} \lor \text{MergeOU} \ (\text{oub}, \ \text{oua}) = \text{oua}
\]
(iv): \( \text{oua.NextState} = \text{normend} \land \text{oub.NextState} = \text{continues} \)
\[ \Rightarrow \text{MergeOU (oua, oub) = ContActSem (oua.TheAct, oua.Rest \cup \{ \text{EpsSem} \})} \land \text{MergeOU (oub, oua) = ContActSem (oub.TheAct, oub.Rest \cup \{ \text{EpsSem} \})} \]
\[ = \text{ContActSem (oua.TheAct, oua.Rest \cup \{ \text{EpsSem} \})} \]

(v): \( \text{oua.NextState} = \text{normend} \land \text{oub.NextState} = \text{normend} \)
\[ \Rightarrow \text{MergeOU (oua, oub) = oua \land \text{MergeOU (oub, oua) = oua}} \]

and
\[ \Rightarrow \text{oua.Rest} = \emptyset \land \text{oub.Rest} = \emptyset \land \text{oua.DoesAct = oub.DoesAct} \Rightarrow \text{oua = oub} \]

(vi): \( \text{oua.NextState} = \text{normend} \land \text{oub.NextState} = \text{abnormend} \)
\[ \Rightarrow \text{MergeOU (oua, oub) = oua \land \text{MergeOU (oub, oua) = oua}} \]

(vii): \( \text{oua.NextState} = \text{abnormend} \land \text{oub.NextState} = \text{continues} \)
\[ \Rightarrow \text{MergeOU (oua, oub) = oub \land \text{MergeOU (oub, oua) = oub}} \]

(viii): \( \text{oua.NextState} = \text{abnormend} \land \text{oub.NextState} = \text{normend} \)
\[ \Rightarrow \text{MergeOU (oua, oub) = oub \land \text{MergeOU (oub, oua) = oub}} \]

(ix): \( \text{oua.NextState} = \text{abnormend} \land \text{oub.NextState} = \text{abnormend} \)
\[ \Rightarrow \text{MergeOU (oua, oub) = oub \land \text{MergeOU (oub, oua) = oua}} \]
\[ \land \Rightarrow \text{oua.Rest} = \emptyset \land \text{oub.Rest} = \emptyset \land \text{oua.DoesAct = oub.DoesAct} \Rightarrow \text{oua = oub} \]

Hence the theorem holds for all nine of these cases, and so holds.

The next theorem defines the associativity property for \( \text{MergeOU} \).

**Theorem 20.**
\[ \forall \text{oua, oub, ouc : OpSUnit} \mid \text{OUNorm01 (oua)} \land \text{OUNorm01 (oub)} \land \text{OUNorm01 (ouc)} \land \text{oua.TheAct \equiv oub.TheAct \equiv ouc.TheAct} \]
\[ \Rightarrow \text{MergeOU (MergeOU (oua, oub), ouc) = MergeOU (oua, MergeOU (oub, ouc))} \]

**Proof.**
Here there are 27 cases to be considered, for the different combinations of \( \text{oua.NextState}, \text{oub.NextState} \) and \( \text{ouc.NextState} \). For each we let four objects of type OpSUnit be defined, as follows:
\[ \text{ouab} = \text{MergeOU (oua, oub)}, \text{oubc} = \text{MergeOU (oub, ouc)}, \text{oul} = \text{MergeOU (ouab, ouc)}, \text{our} = \text{MergeOU (oua, oubc)} \]

Then, the cases are as follows, where (to help illustrate the structure of the set) a blank line has been inserted after each group of nine that has the same value of \( \text{oua.NextState} \).

(i): \( \text{oua.NextState} = \text{continues} \land \text{oub.NextState} = \text{continues} \land \text{ouc.NextState} = \text{continues} \)
\[ \Rightarrow \text{ouab = ContActSem (oua.TheAct, oua.Rest \cup \{ \text{EpsSem} \})} \land \text{ouab.NextState = continues} \]
\[ \Rightarrow \text{oul = ContActSem (oua.TheAct, oua.Rest \cup \{ \text{EpsSem} \}} \land \text{oubc = ContActSem (oub.TheAct, oub.Rest \cup \{ \text{EpsSem} \})} \land \text{oubc.NextState = continues} \]
\[ \Rightarrow \text{our = ContActSem (oua.TheAct, oua.Rest \cup \{ \text{EpsSem} \}} \land \text{oubc.Rest = oua.Rest} \land \text{oua.DoesAct = oubc.DoesAct} \Rightarrow \text{oua = oub} \]

(ii): \( \text{oua.NextState} = \text{continues} \land \text{oub.NextState} = \text{continues} \land \text{ouc.NextState = normend} \)
\[ \Rightarrow \text{ouab = ContActSem (oua.TheAct, oua.Rest \cup \{ \text{EpsSem} \})} \land \text{ouab.NextState = continues} \]
\[ \Rightarrow \text{oul = ContActSem (oua.TheAct, oua.Rest \cup \{ \text{EpsSem} \}} \land \text{oubc = ContActSem (oub.TheAct, oub.Rest \cup \{ \text{EpsSem} \})} \land \text{oubc.NextState = normend} \]
\[ \Rightarrow \text{our = ContActSem (oua.TheAct, oua.Rest \cup \{ \text{EpsSem} \}} \land \text{oubc.Rest = oua.Rest} \land \text{oua.DoesAct = oubc.DoesAct} \Rightarrow \text{oua = oub} \]

(iii): \( \text{oua.NextState} = \text{continues} \land \text{oub.NextState} = \text{continues} \land \text{ouc.NextState = abnormend} \)
\[ \Rightarrow \text{ouab = ContActSem (oua.TheAct, oua.Rest \cup \{ \text{EpsSem} \})} \land \text{ouab.NextState = continues} \]
\[ \Rightarrow \text{oul = ouab \land \text{oubc = oub \land \text{oubc.NextState = normend}} \land \text{oua.Rest = oub.Rest} \land \text{oua.DoesAct = oubc.DoesAct} \Rightarrow \text{oua = oub} \]

(iv): \( \text{oua.NextState = normend} \land \text{oub.NextState = normend} \land \text{ouc.NextState = normend} \)
\[ \Rightarrow \text{ouab = ContActSem (oua.TheAct, oua.Rest \cup \{ \text{EpsSem} \})} \land \text{ouab.NextState = normend} \]
\[ \Rightarrow \text{oul = ContActSem (oua.TheAct, oua.Rest \cup \{ \text{EpsSem} \}} \land \text{oubc = ContActSem (oub.TheAct, oub.Rest \cup \{ \text{EpsSem} \})} \land \text{oubc.NextState = normend} \]
\[ \Rightarrow \text{our = ContActSem (oua.TheAct, oua.Rest \cup \{ \text{EpsSem} \}} \land \text{oubc.Rest = oua.Rest} \land \text{oua.DoesAct = oubc.DoesAct} \Rightarrow \text{oua = oub} \]

(v): \( \text{oua.NextState = normend} \land \text{oub.NextState = normend} \land \text{ouc.NextState = normend} \)
\[ \Rightarrow \text{ouab = ContActSem (oua.TheAct, oua.Rest \cup \{ \text{EpsSem} \})} \land \text{ouab.NextState = normend} \]
\[ \Rightarrow \text{oul = ContActSem (oua.TheAct, oua.Rest \cup \{ \text{EpsSem} \}} \land \text{oubc = oua \land \text{oubc.NextState = normend}} \land \text{our = ContActSem (oua.TheAct, oua.Rest \cup \{ \text{EpsSem} \})} \land \text{our.Rest = oua.Rest} \land \text{oua.DoesAct = oubc.DoesAct} \Rightarrow \text{oua = oub} \]

24
(vi): \( \text{oua.NextState} = \text{continues} \land \text{oub.NextState} = \text{normend} \land \text{ouc.NextState} = \text{abnormend} \)
\( \Rightarrow \text{ouab} = \text{ContActSem (oua.TheAct, oua.Rest } \cup \{ \text{EpsSem} \} \) \) \( \Rightarrow \text{ouab.NextState} = \text{continues} \)
\( \Rightarrow \text{oul} = \text{ouab} \land \text{oubc.NextState} = \text{normend} \)
\( \Rightarrow \text{our} = \text{ContActSem (oua.TheAct, oua.Rest } \cup \{ \text{EpsSem} \} \) 

(vii): \( \text{oua.NextState} = \text{continues} \land \text{oub.NextState} = \text{abnormend} \land \text{ouc.NextState} = \text{continues} \)
\( \Rightarrow \text{ouab} = \text{oua} \Rightarrow \text{oub.NextState} = \text{continues} \)
\( \Rightarrow \text{oul} = \text{ContActSem (oua.TheAct, oua.Rest } \cup \text{ouc.Rest}) \land \text{oubc} = \text{ouc} \Rightarrow \text{oubc.NextState} = \text{continues} \)
\( \Rightarrow \text{our} = \text{ContActSem (oua.TheAct, oua.Rest } \cup \text{ouc.Rest}) \)

(viii): \( \text{oua.NextState} = \text{continues} \land \text{oub.NextState} = \text{abnormend} \land \text{ouc.NextState} = \text{continues} \)
\( \Rightarrow \text{ouab} = \text{oua} \Rightarrow \text{oub.NextState} = \text{continues} \)
\( \Rightarrow \text{oul} = \text{ContActSem (oua.TheAct, oua.Rest } \cup \{ \text{EpsSem} \} \) 

(ix): \( \text{oua.NextState} = \text{continues} \land \text{oub.NextState} = \text{abnormend} \land \text{ouc.NextState} = \text{abnormend} \)
\( \Rightarrow \text{ouab} = \text{oua} \Rightarrow \text{oub.NextState} = \text{continues} \Rightarrow \text{oul} = \text{oua} \land \text{oubc} = \text{ouc} \Rightarrow \text{oubc.NextState} = \text{abnormend} \Rightarrow \text{our} = \text{oua} \)

(x): \( \text{oua.NextState} = \text{normend} \land \text{oub.NextState} = \text{continues} \land \text{ouc.NextState} = \text{continues} \)
\( \Rightarrow \text{ouab} = \text{ContActSem (oua.TheAct, oub.Rest } \cup \{ \text{EpsSem} \} \) \)
\( \Rightarrow \text{oul} = \text{ContActSem (oug.TheAct, ouc.Rest } \cup \{ \text{EpsSem} \} \) 

(xi): \( \text{oua.NextState} = \text{normend} \land \text{oub.NextState} = \text{continues} \land \text{ouc.NextState} = \text{normend} \)
\( \Rightarrow \text{ouab} = \text{ContActSem (oua.TheAct, oub.Rest } \cup \{ \text{EpsSem} \} \) \)
\( \Rightarrow \text{oul} = \text{ContActSem (oua.TheAct, oua.Rest } \cup \{ \text{EpsSem} \} \) 

(xii): \( \text{oua.NextState} = \text{normend} \land \text{oub.NextState} = \text{continues} \land \text{ouc.NextState} = \text{normend} \)
\( \Rightarrow \text{ouab} = \text{ContActSem (oua.TheAct, oub.Rest } \cup \{ \text{EpsSem} \} \) \)
\( \Rightarrow \text{oul} = \text{ContActSem (oua.TheAct, oua.Rest } \cup \{ \text{EpsSem} \} \) 

(xiii): \( \text{oua.NextState} = \text{normend} \land \text{oub.NextState} = \text{normend} \land \text{ouc.NextState} = \text{continues} \)
\( \Rightarrow \text{ouab} = \text{oua} \Rightarrow \text{oub.NextState} = \text{normend} \Rightarrow \text{oul} = \text{ouab} \land \text{oubc} = \text{ouc} \Rightarrow \text{oubc.NextState} = \text{normend} \Rightarrow \text{our} = \text{oua} \)

(xiv): \( \text{oua.NextState} = \text{normend} \land \text{oub.NextState} = \text{normend} \land \text{ouc.NextState} = \text{normend} \)
\( \Rightarrow \text{ouab} = \text{oua} \Rightarrow \text{oub.NextState} = \text{normend} \Rightarrow \text{oul} = \text{ouab} \land \text{oubc} = \text{ouc} \Rightarrow \text{oubc.NextState} = \text{normend} \Rightarrow \text{our} = \text{oua} \)

(xv): \( \text{oua.NextState} = \text{normend} \land \text{oub.NextState} = \text{normend} \land \text{ouc.NextState} = \text{normend} \)
\( \Rightarrow \text{ouab} = \text{oua} \Rightarrow \text{oub.NextState} = \text{normend} \Rightarrow \text{oul} = \text{ouab} \land \text{oubc} = \text{ouc} \Rightarrow \text{oubc.NextState} = \text{normend} \Rightarrow \text{our} = \text{oua} \)

(xvi): \( \text{oua.NextState} = \text{abnormend} \land \text{oub.NextState} = \text{abnormend} \land \text{ouc.NextState} = \text{continues} \)
\( \Rightarrow \text{ouab} = \text{oua} \Rightarrow \text{oub.NextState} = \text{normend} \Rightarrow \text{oul} = \text{ContActSem (oua.TheAct, ouc.Rest } \cup \{ \text{EpsSem} \} \) 

(xvii): \( \text{oua.NextState} = \text{abnormend} \land \text{oub.NextState} = \text{abnormend} \land \text{ouc.NextState} = \text{normend} \)
\( \Rightarrow \text{ouab} = \text{oua} \Rightarrow \text{oub.NextState} = \text{normend} \Rightarrow \text{oul} = \text{ouab} \land \text{oubc} = \text{ouc} \Rightarrow \text{oubc.NextState} = \text{normend} \Rightarrow \text{our} = \text{oua} \)

(xviii): \( \text{oua.NextState} = \text{abnormend} \land \text{oub.NextState} = \text{abnormend} \land \text{ouc.NextState} = \text{abnormend} \)
\( \Rightarrow \text{ouab} = \text{oua} \Rightarrow \text{oub.NextState} = \text{normend} \Rightarrow \text{oul} = \text{ouab} \land \text{oubc} = \text{ouc} \Rightarrow \text{oubc.NextState} = \text{abnormend} \Rightarrow \text{our} = \text{oua} \)

(xix): \( \text{oua.NextState} = \text{abnormend} \land \text{oub.NextState} = \text{continues} \land \text{ouc.NextState} = \text{continues} \)
\( \Rightarrow \text{ouab} = \text{oub} \Rightarrow \text{oub.NextState} = \text{continues} \)
NormOS1 and NormOU1, these three can all be expressed as theorems, although here only the first will have two
alternative statements, but the proofs of the first two theorems will have more complicated structures than those for theorems 16 and 17.

The reason for this is that, because of the structures of the functions, the proofs require two inductions: an outer one over the heights of the forests involved, and an inner one over the number of trees in a forest. For the inner one the structure of the function NormOS2 means that the base case is where each tree has a different action associated with its root node, and so two auxiliary functions are needed to define the appropriate metric for this induction.

The first of these functions is one to extract this “head” action, and so this function is called GetHead, and it has signature \( \text{OpSUnit} \rightarrow \text{Act} \). The definition of this function then assumes that the parameter is in first normal form, so that the case where it does not perform an action but does continue can be ignored, and hence the definition is

\[
\text{GetHead} (\text{ou}) \equiv \begin{cases} 
\text{if } \text{ou}.\text{DoesAct} & \text{then } \text{ou}.\text{TheAct} \\
\text{else if } \text{ou}.\text{NextState} = \text{abnormend} & \text{then } \emptyset \\
\text{else } & \varepsilon \\
\end{cases}
\]

The second auxiliary function then constructs the set of head actions for any object of type \( \text{OpSem} \) that is in first normal form, in a fashion that is analogous to the function SeqHeads that is used in the guards in clauses (ix) and (x) of the definition of the transition relation. Since this function is defined over the semantic structures rather than over DFA constructions it is called simply Heads, and it has signature \( \text{OpSem} \rightarrow P \text{Act} \), and is defined as

\[
\text{Heads} (\text{os}) \equiv \begin{cases} 
\text{if } \text{os} = \emptyset & \text{then } \emptyset \\
\text{else } \{ \forall \text{ou} : \text{OpSUnit} \mid \text{ou} \in \text{os} \ \text{GetHead} (\text{ou}) \} & \text{fi} \\
\end{cases}
\]

In this definition, the significance of the assumption that the parameter is in first normal form is that SeqHeads will only deliver the result \( \{ \varepsilon \} \) if actually applied to the silent action, and not to a sequence of the form \( \varepsilon ; s \), but the analogous effect for Heads will only occur for forest structures that are in first normal form. It is also worth noting that the analogy between the behaviour of the two functions is only precise if parameters to Heads are also in second normal form, since SeqHeads will never produce a result of the form \( \{ a, \phi \} \) for any \( a \in \text{PA} \), whereas Heads can produce such a result if applied to a forest that is not in second normal form. A formal proof of this analogy will, however, have to wait until the semantic functions have been fully defined, in section 8.

There are then three simple properties of these functions that will be useful in subsequent proofs, and that are expressed as the following theorems.

**Theorem 22.**

\[
\forall \text{ou} : \text{OpSUnit} \mid \text{OUNorm01} (\text{ou}) \bullet \\
(\text{GetHead} (\text{ou}) = \varepsilon \Rightarrow \text{ou} = \text{EpsSem}) \land (\text{GetHead} (\text{ou}) = \phi \Rightarrow \text{ou} = \text{PhiSem}).
\]

**Proof.**

From theorem 12, under the conditions of this theorem there are only three possible disjoint cases that can occur, as follows.

\[
\text{ou}.\text{DoesAct} = \text{true} \Rightarrow \exists a : \text{PA} \bullet a = \text{ou}.\text{TheAct} \Rightarrow \text{GetHead} (\text{ou}) = a \\
\text{ou} = \text{EpsSem} \Rightarrow \text{GetHead} (\text{ou}) = \varepsilon \\
\text{ou} = \text{PhiSem} \Rightarrow \text{GetHead} (\text{ou}) = \phi
\]

Hence the theorem holds for all three of these cases, and so holds.

**Theorem 23.**

\[
\forall a : \text{PA}, \ os : \text{OpSem} \mid \text{OSNorm01} (\text{os}) \bullet \\
\text{GetHead} (\text{Cont1ActSem} (a, \text{os})) = a.
\]

**Proof.**

As in the proof of theorem 13, the proof of this requires an analysis of the various cases for the size and construction of \( \text{os} \) in the definition of Cont1ActSem, and so there are three main cases to consider.
Case (i): \# os = 0 \Rightarrow os = \emptyset \Rightarrow \text{Cont1ActSem} (a, os) = \text{FinalActSem} (a) \\
\Rightarrow \text{GetHead} ( \text{Cont1ActSem} (a, os) ) = \text{GetHead} ( \text{FinalActSem} (a) ) = a

Case (ii): \# os = 1 \Rightarrow \exists ou : \text{OpSUnit} \bullet os = \{ ou \}
Again, there are two sub-cases of this, depending on the value of \text{ou}.\text{DoesAct}, as follows.
\text{ou}.\text{DoesAct} = \text{true} \Rightarrow (ou \neq \text{EpsSem}) \land (ou \neq \text{PhiSem}) \Rightarrow \text{Cont1ActSem} (a, os) = \text{ContActSem} (a, os)
\Rightarrow \text{GetHead} ( \text{Cont1ActSem} (a, os) ) = \text{GetHead} ( \text{ContActSem} (a, os) ) = a
\text{ou}.\text{DoesAct} = \text{false} \Rightarrow (ou = \text{EpsSem}) \lor (ou = \text{PhiSem})
\Rightarrow ( \text{Cont1ActSem} (a, os) = \text{FinalActSem} (a) ) \lor ( \text{Cont1ActSem} (a, os) = \text{FinalAbActSem} (a) )
\Rightarrow ( \text{GetHead} ( \text{Cont1ActSem} (a, os) ) = \text{GetHead} ( \text{FinalActSem} (a) ) ) \lor 
( \text{GetHead} ( \text{Cont1ActSem} (a, os) ) = \text{GetHead} ( \text{FinalAbActSem} (a) ) )
\Rightarrow \text{GetHead} ( \text{Cont1ActSem} (a, os) ) = a

Case (iii): \# os > 1 \Rightarrow \text{Cont1ActSem} (a, os) = \text{ContActSem} (a, os)
\Rightarrow \text{GetHead} ( \text{Cont1ActSem} (a, os) ) = \text{GetHead} ( \text{ContActSem} (a, os) ) = a

Hence the theorem holds for all cases, and so holds.

The third of these three properties involves the interaction between the two functions Heads and NormOS2b, and it should be noted that it does not require the condition OSNorm2a to hold for any forest os to which it applies, although in practice the operation of the function NormOS2 is such that it will only ever invoke NormOS2b for forests where this condition does hold. Thus, it is expressed as the following theorem.

Theorem 24.
\forall os : \text{OpSem} \ | \ os \neq \emptyset \land \text{OSNorm01} (os) \bullet 
( \# os = 1 \lor \text{PhiSem} \notin os \Rightarrow \text{Heads} (\text{NormOS2b} (os) ) = \text{Heads} (os) ) \\
\land ( \# os > 1 \land \text{PhiSem} \in os \Rightarrow \text{Heads} (\text{NormOS2b} (os) ) = \text{Heads} (os) - \{ \text{PhiSem} \} )

Proof.
From the conditions of this theorem there are three possible cases that can occur, as follows.
(i) \# os = 1 \Rightarrow \exists ou : \text{OpSUnit} \bullet os = \{ ou \}
\Rightarrow \text{Heads} (os) = \{ \text{GetHead} (ou) \}
and
\Rightarrow \text{NormOS2b} (os) = \{ \text{NormOU2} (ou) \}
\Rightarrow \text{Heads} (\text{NormOS2b} (os) ) = \text{Heads} (\{ \text{NormOU2} (ou) \} )
= \{ \text{GetHead} (ou) \} = \text{Heads} (os)

(ii) \text{PhiSem} \notin os
\Rightarrow \text{Heads} (\text{NormOS2b} (os) ) = \text{Heads} (\{ \forall ou : \text{OpSUnit} \ | \ ou \in os \land ou \neq \text{PhiSem} \bullet \text{NormOU2} (ou) \} )
= \text{Heads} (\{ \forall ou : \text{OpSUnit} \ | \ ou \in os \bullet \text{NormOU2} (ou) \} )
= \{ \forall ou : \text{OpSUnit} \ | \ ou \in os \bullet \text{GetHead} (\text{NormOU2} (ou) ) \} = \text{Heads} (os)

(iii) \# os > 1 \land \text{PhiSem} \in os
\Rightarrow \text{Heads} (\text{NormOS2b} (os) ) = \text{Heads} (\{ \forall ou : \text{OpSUnit} \ | \ ou \in os \land ou \neq \text{PhiSem} \bullet \text{NormOU2} (ou) \} )
= \text{Heads} (\{ \forall ou : \text{OpSUnit} \ | \ ou \in os \bullet \text{NormOU2} (PhiSem) \} )
= \{ \forall ou : \text{OpSUnit} \ | \ ou \in os \bullet \text{GetHead} (\text{NormOU2} (ou) ) \} - \{ \text{GetHead} (\text{PhiSem}) \}
= \text{Heads} (os) - \{ \text{PhiSem} \}

Hence the theorem holds for all three of these cases, and so holds.

Theorem 25.
\forall os : \text{OpSem} \ | \ \text{OSNorm01} (os) \bullet 
( \# \text{Heads} (os) = \# os \Rightarrow 
( \exists ou1, ou2 : \text{OpSUnit} \ | \ ou1 \in os \land ou2 \in os \land ou1 \neq ou2 \land ou1.\text{DoesAct} \land ou2.\text{DoesAct} \bullet 
ou1.\text{TheAct} = ou2.\text{TheAct} ) ) \land 
( \# \text{Heads} (os) < \# os \Rightarrow 
( \exists ou1, ou2 : \text{OpSUnit} \ | \ ou1 \in os \land ou2 \in os \land ou1 \neq ou2 \land ou1.\text{DoesAct} \land ou2.\text{DoesAct} \bullet 
ou1.\text{TheAct} = ou2.\text{TheAct} ) )


Proof.
The proof is by induction over \# os.

Base case.
There are effectively two base cases, one for the empty set and the other for a single element set. If the set os is empty then the theorem is trivially true, because no elements ou1 and ou2 can exist, and \# Heads (os) cannot be less than zero.

If the set os has a single element then again the theorem must be trivially true, because there can not be two different elements ou1 and ou2, and Heads (os) must also be a single element set, so its cardinality can not be less than the cardinality of os.

Inductive case.
The inductive case applies to any forest os such that \# os = n, for any arbitrary natural number n > 0, and the induction hypothesis is that the theorem holds for any such forest os. The induction step is then to show from this that the theorem therefore also holds for any forest os such that \# os = n + 1. There are two parts to this step, corresponding to the two parts of the conjunction in the theorem: one part for the case where \# Heads (os) = \# os and the other part for the case where \# Heads (os) < \# os. We will treat each separately, although the structure of the proof is similar for each part, as it involves constructing a new object os1 : OpSem, where os1 = os \cup \{ ou \} for any arbitrary object ou: OpSUnit which satisfies the conditions

\[
\text{NormOU0 (ou)} \land \text{NormOU1 (ou)} \land \neg (ou \in os),
\]

and then analysing the various possible cases for this construction. Hence, for each part it is straightforward from the definitions of NormOS0 and NormOS1 that we shall have

\[
\text{NormOS0 (os1)} \land \text{NormOS1 (os1)},
\]

so that the conditions of the theorem hold for os1, and since \neg (ou \in os) we shall also have \# os1 = \# os + 1 = n + 1.

Inductive case: \# Heads (os) = \# os.
As in the proof of theorem 22, there are just three separate sub-cases that need to be considered for the construction of ou, as follows.

Inductive sub-case (i): ou2.DoesAct = true.
Here there are two possibilities that need to be analysed, as follows.

\[
\begin{align*}
\text{ou}.\text{DoesAct} & \land \text{ou}.\text{TheAct} \in \text{Heads (os)} \\
\Rightarrow & \text{Heads (os1)} = \text{Heads (os)} \\
\Rightarrow & \# \text{Heads (os1)} < \# os1
\end{align*}
\]

and

\[
\Rightarrow \exists \text{ou1 : OpSUnit | } \text{ou1} \in os \land \text{ou1}.\text{DoesAct} \land \text{ou1}.\text{TheAct} = \text{ou1}.\text{Act}.
\]

Hence, this possibility the first implication in the theorem is true for os1, because the left-hand side is false, while the second implication is true for os1 because both sides are true.

The other possibility is

\[
\begin{align*}
\text{ou}.\text{DoesAct} & \land \text{ou}.\text{TheAct} \notin \text{Heads (os)} \\
\Rightarrow & \text{Heads (os1)} = \text{Heads (os)} + 1 \\
\Rightarrow & \# \text{Heads (os1)} = \# os1
\end{align*}
\]

and

\[
\Rightarrow \neg (\exists \text{ou1 : OpSUnit | } \text{ou1} \in os \land \text{ou1}.\text{DoesAct} \land \text{ou1}.\text{TheAct} = \text{ou1}.\text{Act}).
\]

since, from the induction hypothesis

\[
\neg (\exists \text{ou1, ou2 : OpSUnit | } \text{ou1} \in os \land \text{ou2} \in os \land \text{ou1} \neq \text{ou2} \land \text{ou1}.\text{DoesAct} \land \text{ou2}.\text{DoesAct} \land \text{ou1}.\text{TheAct = ou2}.\text{TheAct})
\]

Hence, for this possibility the first implication in the theorem is true for os1, because both sides are true, while the second implication is true for os1 because the left-hand side is false. Hence, for both possibilities the theorem will hold for os1.

Inductive sub-case (ii): \text{ou = EpsSem}

\[
\Rightarrow \neg (\exists \text{ou1, ou2 : OpSUnit | } \text{ou1} \in os \land \text{ou2} \in os \land \text{ou1} \neq \text{ou2} \land \text{ou1}.\text{DoesAct} \land \text{ou2}.\text{DoesAct} \land \text{ou1}.\text{TheAct = ou2}.\text{TheAct})
\]

Hence, for both possibilities the theorem will hold for os1.
Inductive sub-case (iii): \( \text{ou} = \text{PhiSem} \)
\[ \Rightarrow \text{PhiSem} \not\in \text{os} \Rightarrow \phi \not\in \text{Heads (os)}, \] because (from theorem 12) there could not be any other element of \( \text{os} \) that had \( \text{ou}.\text{DoesAct} = \text{false} \) and \( \text{ou}.\text{NextState} = \text{abnormend} \). Hence, we have
\[ \text{Heads (os1)} = \text{Heads (os)} + 1 \Rightarrow \# \text{Heads (os1)} = \# \text{os1} \]
and again, since \( \text{ou}.\text{DoesAct} = \text{false} \) we have from the induction hypothesis that
\[ \neg ( \exists \text{ou1}, \text{ou2} : \text{OpSUnit} \mid \text{ou1} \not\in \text{os} \land \text{ou2} \in \text{os} \land \text{ou1} \neq \text{ou2} \land \text{ou1}.\text{DoesAct} \land \text{ou2}.\text{DoesAct} \bullet \]
\[ \text{ou1}.\text{TheAct} = \text{ou2}.\text{TheAct} ) \]
\[ \Rightarrow \neg ( \exists \text{ou1}, \text{ou2} : \text{OpSUnit} \mid \text{ou1} \not\in \text{os} \land \text{ou2} \in \text{os} \land \text{ou1} \neq \text{ou2} \land \text{ou1}.\text{DoesAct} \land \text{ou2}.\text{DoesAct} \bullet \]
\[ \text{ou1}.\text{TheAct} = \text{ou2}.\text{TheAct} ) \]
Hence, for this sub-case too the first implication in the theorem is true for \( \text{os1} \), because both sides are true, while the second implication is false for \( \text{os1} \) because the left-hand side is false, and so for this sub-case the theorem will hold for \( \text{os1} \).

Inductive case: \( \# \text{Heads (os)} < \# \text{os} \).
Again, from the structure of the definition of GetHead, and from theorem 12, there are just the same three separate sub-cases that need to be considered for the construction of \( \text{ou} \), although they are numbered here to continue from the first inductive case, as follows.

Inductive sub-case (iv): \( \text{ou}.\text{DoesAct} = \text{true} \).
Here there are again two possibilities that need to be analysed, as follows.
\[ \text{ou}.\text{DoesAct} \land \text{ou}.\text{TheAct} \not\in \text{Heads (os)} \]
\[ \Rightarrow \text{Heads (os1)} = \text{Heads (os)} \]
\[ \Rightarrow \# \text{Heads (os1)} < \# \text{os1} \]
\[ \Rightarrow \exists \text{ou1} : \text{OpSUnit} \mid \text{ou1} \not\in \text{os} \land \text{ou1}.\text{DoesAct} \bullet \text{ou1}.\text{TheAct} = \text{ou}.\text{TheAct} \]
Hence, for this possibility the first implication in the theorem is again true for \( \text{os1} \), because the left-hand side is false, while the second implication is true for \( \text{os1} \) because both sides are true. Hence, for both possibilities the theorem holds for \( \text{os1} \).

The other possibility is
\[ \text{ou}.\text{DoesAct} \land \text{ou}.\text{TheAct} \in \text{Heads (os)} \]
\[ \Rightarrow \text{Heads (os1)} = \text{Heads (os)} + 1 \]
\[ \Rightarrow \# \text{Heads (os1)} < \# \text{os1} \]
since \( \# \text{os1} = \# \text{os} + 1 \)
\[ \Rightarrow \exists \text{ou1}, \text{ou2} : \text{OpSUnit} \mid \text{ou1} \not\in \text{os} \land \text{ou2} \in \text{os} \land \text{ou1} \neq \text{ou2} \land \text{ou1}.\text{DoesAct} \land \text{ou2}.\text{DoesAct} \bullet \]
\[ \text{ou1}.\text{TheAct} = \text{ou2}.\text{TheAct} \]
Hence, for this possibility as well the first implication in the theorem is true for \( \text{os1} \), because the left-hand side is false, while the second implication is true for \( \text{os1} \) because both sides are true. Hence, for both possibilities the theorem holds for \( \text{os1} \).

Inductive sub-case (v): \( \text{ou} = \text{EpsSem} \)
\[ \Rightarrow \text{EpsSem} \not\in \text{os} \Rightarrow \phi \not\in \text{Heads (os)}, \] because again from theorem 12 there could not be any other element of \( \text{os} \) that had \( \text{ou}.\text{DoesAct} = \text{false} \) and \( \text{ou}.\text{NextState} = \text{normend} \). Hence, we have
\[ (\text{Heads (os1)} = \text{Heads (os)} + 1) \land (\# \text{os1} = \# \text{os} + 1) \Rightarrow \# \text{Heads (os1)} < \# \text{os1} \]
and since \( \text{ou}.\text{DoesAct} = \text{false} \) we have from the induction hypothesis that
\[ \exists \text{ou1}, \text{ou2} : \text{OpSUnit} \mid \text{ou1} \not\in \text{os} \land \text{ou2} \in \text{os} \land \text{ou1} \neq \text{ou2} \land \text{ou1}.\text{DoesAct} \land \text{ou2}.\text{DoesAct} \bullet \]
\[ \text{ou1}.\text{TheAct} = \text{ou2}.\text{TheAct} \]
\[ \Rightarrow \exists \text{ou1}, \text{ou2} : \text{OpSUnit} \mid \text{ou1} \not\in \text{os} \land \text{ou2} \in \text{os} \land \text{ou1} \neq \text{ou2} \land \text{ou1}.\text{DoesAct} \land \text{ou2}.\text{DoesAct} \bullet \]
\[ \text{ou1}.\text{TheAct} = \text{ou2}.\text{TheAct} \]
Hence, for this sub-case the first implication in the theorem is true for \( \text{os1} \), because the left-hand side is false, while the second implication is true for \( \text{os1} \) because both sides are true, and so for this sub-case the theorem holds for \( \text{os1} \).

Inductive sub-case (vi): \( \text{ou} = \text{PhiSem} \)
\[ \Rightarrow \text{PhiSem} \not\in \text{os} \Rightarrow \phi \not\in \text{Heads (os)}, \] because again from theorem 12 there could not be any other element of \( \text{os} \) that had \( \text{ou}.\text{DoesAct} = \text{false} \) and \( \text{ou}.\text{NextState} = \text{abnormend} \). Hence, we have
\[ (\text{Heads (os1)} = \text{Heads (os)} + 1) \land (\# \text{os1} = \# \text{os} + 1) \Rightarrow \# \text{Heads (os1)} < \# \text{os1} \]
and again, since ou\,DoesAct = false we have from the induction hypothesis that
\[ \exists ou1, ou2 : OpSUnit \mid ou1 \in os \land ou2 \in os \land ou1 \neq ou2 \land ou1\,DoesAct \land ou2\,DoesAct \land ou1\,TheAct = ou2\,TheAct \]
\[ \Rightarrow \exists ou1, ou2 : OpSUnit \mid ou1 \in os1 \land ou2 \in os1 \land ou1 \neq ou2 \land ou1\,DoesAct \land ou2\,DoesAct \land ou1\,TheAct = ou2\,TheAct \]
Hence, for this sub-case too the first implication in the theorem is true for os1, because the left-hand side is false, while the second implication is true for os1 because both sides are true, and so the theorem holds for os1 for this sub-case as well. Hence it holds for all the sub-cases in this second inductive case, and so the whole inductive case follows from the combination of these with the sub-cases of the first inductive case, which between them cover all the possible constructions of an object os1 such that \# os1 = \# os + 1 = n + 1. Hence, by induction the theorem holds for all values of n \geq 1, and so holds.

These properties are then used in the proof of the theorem that NormOS2 constructs objects in zeroth and first strict normal forms if it is applied to objects that are in these forms, and this theorem is as follows.

**Theorem 26.**
\[
\forall os : OpSem, ou' : OpSUnit \mid ou' \in NormOS2 (os) \Rightarrow OSNorm01 (os) \Rightarrow OUNorm0 (ou') \land OUNorm1 (ou')
\]
or, alternatively
\[
\forall os : OpSem \Rightarrow OSNorm01 (os) \Rightarrow OSNorm0 (NormOS2 (os)) \land OSNorm1 (NormOS2 (os))
\]
where the alternative form follows directly from the first form and the definitions of OSNorm0 and OSNorm1 in terms of OUNorm0 and OUNorm1 respectively.

**Proof.**
The proof focuses on the first form of the theorem, and it involves a double induction, where the outer induction is over the height of the forest os, and the inner induction is over the metric \# os − \# Heads (os) for which the properties have been established in theorem 25.

**Outer Base Case:** HeightOS (os) = 1
There are two parts to the proof of this case, corresponding to the base and recursive cases of the inner induction.

**Inner Base Case:** \# os − \# Heads (os) = 0
In this case, from theorem 25 the conditions for NormOS2 (os) to invoke MergeOU can not be satisfied, and so instead it simply invokes NormOS2b (os) directly.

For this invocation there are two slightly different cases, depending on the size of os, but effectively the argument is the same for both. Thus, if \# os = 1 then NormOU2 is invoked directly for this element, which must therefore be in both zeroth and first normal forms. If \# os > 1 then NormOU2 is invoked directly for each element that is not equal to PhiSem, but again each of these elements must be in both zeroth and first normal forms.

For each of these invocations of NormOU2 it must follow from the outer base case that HeightOU (ou) = 1, and hence that ou\,Rest = \emptyset, and hence that NormOU2 = ou, where from the conditions of this theorem OUNorm0 (ou) \land OUNorm1 (ou) both hold. Thus, each element ou' of the set produced by NormOS2b must be such an element ou, and also an element of the set produced by NormOS2, so that OUNorm0 (ou') \land OUNorm1 (ou').

Hence, the theorem holds for this inner base case.

**Inner Inductive Case:** \# os > \# Heads (os)
For this inner inductive case the induction hypothesis is that the theorem holds for the conditions of the outer base case and for some natural number ni such that \# os − \# Heads (os) = ni, and then the inductive step is to show that therefore it also holds for \# os − \# Heads (os) = ni + 1. The analysis of this step depends on the fact that the operation of NormOS2 reduces any case where \# os − \# Heads (os) = ni + 1 to a case where \# os − \# Heads (os) = ni by replacing the pair of elements ou1 and ou2 by a single element constructed as MergeOU (ou1, ou2).

In this replacement, it follows directly from the way in which OSNorm0 and OSNorm1 are defined that the removal of the two elements ou1 and ou2 from os has no effect on whether the rest of its elements are in zeroth or first normal form, and similarly the addition of an element has no effect on this if the new element is in these forms. (As an aside, we may observe that if either of these functions involved dependencies between the different elements of os then it would be
necessary to prove this result by induction over the size of \( os \), but since they do not this step is not required.) Thus, all that needs to be established in order to prove this inner inductive case is that
\[
\forall ou_1, ou_2 : OpSUnit \&\& \text{OUNorm0} (ou_1) \&\& \text{OUNorm1} (ou_1) \&\& \text{OUNorm0} (ou_2) \&\& \text{OUNorm1} (ou_2) \\
\Rightarrow \text{OUNorm0} (\text{MergeOU} (ou_1, ou_2))
\]
and
\[
\forall ou_1, ou_2 : OpSUnit \&\& \text{OUNorm0} (ou_1) \&\& \text{OUNorm1} (ou_1) \&\& \text{OUNorm0} (ou_2) \&\& \text{OUNorm1} (ou_2) \\
\Rightarrow \text{OUNorm1} (\text{MergeOU} (ou_1, ou_2)).
\]

Since the conditions of the outer base case ensure that \( ou_1.\text{NextState} \neq \text{continues} \&\& \text{ou2.\text{NextState} \neq \text{continues} \), then
the result of \( \text{MergeOU} (ou_1, ou_2) \) must be either \( ou_1 \) or \( ou_2 \), and so this result follows immediately. Thus, the theorem also holds for this inner inductive case, and hence for both parts of the outer base case.

**Outer Inductive Case:** \( \text{HeightOS} (os) > 1 \)

For this outer inductive case the induction hypothesis is that, for any natural number \( no > 1 \), the theorem holds for all values \( os' : \text{OpSem} \) such that \( \text{HeightOS} (os') < no \). The induction step is then to show from this that the theorem must also hold for any arbitrary value \( os : \text{OpSem} \) such that \( \text{HeightOS} (os) = no \). Without loss of generality, therefore, suppose that any \( os \) must be such that \( \text{HeightOS} (os) = no \), so that by the definition of \( \text{HeightOS} \) it will follow directly that \( \text{HeightOU} (ou ) \leq no \). Also without loss of generality, assume that \( \text{HeightOU} (ou ) > 1 \), since otherwise the problem will simply reduce to the base case.

Again, there are two parts to the proof of this case, corresponding to the base and recursive cases of the inner induction.

**Inner Base Case:** \( \# os – \# \text{Heads} (os) = 0 \)

As before, from theorem 25 the conditions for \( \text{NormOS2} (os) \) to invoke \( \text{MergeOU} \) can not be satisfied in this case, and so instead it simply invokes \( \text{NormOS2b} (os) \) directly. This again gives rise to two slightly different cases, depending on the size of \( os \), but as before the argument is effectively the same for both, and is that each invocation of \( \text{NormOU2} \) is for an element which must be in both zeroth and first normal forms.

For each of these invocations of \( \text{NormOU2} \), the conditions of the outer inductive case must ensure that \( \text{ou.Rest} \neq \emptyset \), and so \( 1 \leq \text{HeightOS} (\text{ou.Rest}) < \text{no} \). Thus, from the induction hypothesis \( \text{OSNorm0} (\text{ou.Rest}) \&\& \text{OSNorm1} (\text{ou.Rest}) \), and hence from theorem 13 it follows that for each element \( \text{ou1 : OpSUnit} \) of the set produced by the invocations of \( \text{Cont1ActSem} \) in \( \text{NormOS2b} \) we have \( \text{OUNorm0} (\text{ou1}) \&\& \text{OUNorm1} (\text{ou1}) \), and this element \( \text{ou1} \) is also then an element \( os' : \text{OpSem} \) of the set produced by \( \text{NormOS2} \), so that \( \text{OUNorm0} (os') \&\& \text{OUNorm1} (os') \).

Hence, the theorem holds for this inner base case too.

**Inner Inductive Case:** \( \# os > \# \text{Heads} (os) \)

Again, for this inner inductive case the induction hypothesis is that the theorem holds for the conditions of the outer inductive case and for some natural number \( ni \) such that \( \# os – \# \text{Heads} (os) = ni \). The inductive step is then again to show that it also holds for \( \# os – \# \text{Heads} (os) = ni+1 \), and as before the analysis of this depends on the operation of \( \text{NormOS2} \) reducing any case where \( \# os – \# \text{Heads} (os) = ni+1 \) to a case where \( \# os – \# \text{Heads} (os) = ni \), by replacing the pair of elements \( ou1 \) and \( ou2 \) by a single element constructed as \( \text{MergeOU} (ou1, ou2) \).

Again, it follows directly that in this replacement the removal of the two elements \( ou1 \) and \( ou2 \) from \( os \) has no effect on whether the rest of its elements are in zeroth or first normal form, and similarly the addition of an element has no effect on this if the new element is in these forms. Thus, as before what needs to be established in order to prove this inner inductive case is that
\[
\forall ou_1, ou_2 : \text{OpSUnit} \&\& \text{OUNorm0} (ou_1) \&\& \text{OUNorm1} (ou_1) \&\& \text{OUNorm0} (ou_2) \&\& \text{OUNorm1} (ou_2) \\
\Rightarrow \text{OUNorm0} (\text{MergeOU} (ou_1, ou_2))
\]
and
\[
\forall ou_1, ou_2 : \text{OpSUnit} \&\& \text{OUNorm0} (ou_1) \&\& \text{OUNorm1} (ou_1) \&\& \text{OUNorm0} (ou_2) \&\& \text{OUNorm1} (ou_2) \\
\Rightarrow \text{OUNorm1} (\text{MergeOU} (ou_1, ou_2)).
\]

Here, though, the conditions of the outer inductive case mean that we may have either \( ou1.\text{NextState} = \text{continues} \) or \( ou2.\text{NextState} = \text{continues} \) or both, and so all the different cases in the definition of \( \text{MergeOU} \) need to be considered. For each of them the first of these results either follows directly from \( \text{OUNorm0} (ou_1) \) and \( \text{OUNorm0} (ou_2) \), or it follows from theorem 6, where the same conditions ensure that the conditions of this theorem hold.

Similarly, for each case the second of these results either follows directly from \( \text{OUNorm1} (ou_1) \) and \( \text{OUNorm1} (ou_2) \), or it follows from theorem 13, where the left-hand side of the implication together with the construction of the set of descendents ensure that the conditions of this theorem hold.
Consequently, the theorem also holds for this inner inductive case, and so by induction over \( n_i \) it holds for both parts of the outer inductive case as well, for all values of \( n_i \). Therefore, by the outer induction, it holds for all values of \( n_0 \) too, and so holds.

The second property that then needs to be shown for \( \text{NormOS2} \) is that, if it is applied to objects that are in zeroth and first strict normal forms, then the objects that it constructs are in strict second normal form. This property is expressed as the following theorem, where the way in which \( \text{OSNorm2} \) is defined means that (unlike theorem 26) there is only one form to its statement.

**Theorem 27.**

\[
\forall \mathit{os} : \text{OpSem} \bullet \text{OSNorm01} (\mathit{os}) \Rightarrow \text{OSNorm2} (\text{NormOS2} (\mathit{os})).
\]

**Proof.**

The proof is very similar in structure to that of theorem 26, in that it again involves a double induction, where the outer induction is over the height of the forest \( \mathit{os} \), and the inner induction is over the metric \( \# \mathit{os} – \# \text{Heads} (\mathit{os}) \).

**Outer Base Case:** \( \text{HeightOS} (\mathit{os}) = 1 \)

There are two parts to the proof of this case, corresponding to the base and recursive cases of the inner induction.

**Inner Base Case:** \( \# \mathit{os} – \# \text{Heads} (\mathit{os}) = 0 \)

In this case, from the conditions of the theorem and from theorem 25 it must be the case that \( \text{OSNorm2a} (\mathit{os}) \) holds. Also, as in the proof of theorem 26, the conditions for \( \text{NormOS2} (\mathit{os}) \) to invoke \( \text{MergeOU} \) can not be satisfied, and so instead it simply invokes \( \text{OSNorm2b} (\mathit{os}) \) directly.

Then for this invocation we need to show three things, one for each term in the definition of \( \text{OSNorm2} \), as follows.

(i) that \( \text{OSNorm2a} (\text{NormOS2b} (\mathit{os})) \) holds, as a consequence of \( \text{OSNorm2a} (\mathit{os}) \),

(ii) that \( \text{OSNorm2b} (\text{NormOS2b} (\mathit{os})) \) holds, and

(iii) that \( \forall \mathit{ou} : \text{OpSUnit} \mid \mathit{ou} \in \text{NormOS2b} (\mathit{os}) \bullet \text{OUNorm2} (\mathit{ou}) \) holds.

For (i) we observe that

\[
\forall \mathit{ou} : \text{OpSUnit} \bullet \mathit{ou}.\text{TheAct} = (\text{NormOU2} (\mathit{ou})).\text{TheAct}
\]

and this is a general result that follows from the two cases in the definition of \( \text{NormOU2} \), and is not just restricted to the present case, where \( \text{HeightOS} (\mathit{os}) = 1 \).

Hence, there are three cases that arise from the structure of \( \text{NormOS2b} \) and that need to be considered, as follows.

\[
\begin{align*}
\# \mathit{os} = 0 \lor \# \mathit{os} = 1 \Rightarrow \text{NormOS2b} (\mathit{os}) = \mathit{os} \Rightarrow (\text{OSNorm2a} (\mathit{os}) \Rightarrow \text{OSNorm2a} (\text{NormOS2b} (\mathit{os}))) \\
\# \mathit{os} > 1 \land \text{PhiSem} \notin \mathit{os} \Rightarrow \text{Heads} (\text{NormOS2b} (\mathit{os})) = \text{Heads} (\mathit{os}) \quad \text{from theorem 24} \\
& \Rightarrow (\text{OSNorm2a} (\mathit{os}) \Rightarrow \text{OSNorm2a} (\text{NormOS2b} (\mathit{os}))) \\
\end{align*}
\]

and

\[
\begin{align*}
\# \mathit{os} > 1 \land \text{PhiSem} \in \mathit{os} \Rightarrow \text{Heads} (\text{NormOS2b} (\mathit{os})) = (\text{Heads} (\mathit{os}) – \phi) \\
& \Rightarrow (\text{OSNorm2a} (\mathit{os}) \Rightarrow \text{OSNorm2a} (\text{NormOS2b} (\mathit{os}))) \\
\end{align*}
\]

For (ii), since the conditions of this theorem require that \( \mathit{os} \) is in zeroth normal form, it follows from theorem 5 that \( \text{PhiSem} \) is the only object that must not appear in \( \mathit{os} \) in order for \( \text{OSNorm2b} (\mathit{os}) \) to be satisfied. Hence, from the structure of \( \text{NormOS2b} \) there are the same three cases that need to be considered, as follows.

\[
\begin{align*}
\# \mathit{os} = 0 \lor \# \mathit{os} = 1, \\
\# \mathit{os} > 1 \land \text{PhiSem} \notin \mathit{os} \\
\end{align*}
\]

where it follows directly from the definition of \( \text{OSNorm2b} \) that \( \text{OSNorm2b} (\text{NormOS2b} (\mathit{os})) \)

\[
\begin{align*}
\# \mathit{os} > 1 \land \text{PhiSem} \notin \mathit{os} \\
\end{align*}
\]

where since \( \text{PhiSem} \notin \mathit{os} \) it follows directly from theorem 24 that \( \text{OSNorm2b} (\text{NormOS2b} (\mathit{os})) \)

and

\[
\begin{align*}
\# \mathit{os} > 1 \land \text{PhiSem} \in \mathit{os} \Rightarrow \text{Heads} (\text{NormOS2b} (\mathit{os})) = (\text{Heads} (\mathit{os}) – \phi) \\
\end{align*}
\]

where from theorem 24 the object \( \text{PhiSem} \) is removed from \( \text{NormOS2b} (\mathit{os}) \), and so it again follows that

\( \text{OSNorm2b} (\text{NormOS2b} (\mathit{os})) \).

For (iii), since \( \text{HeightOS} (\mathit{os}) = 1 \) it follows that

\[
\begin{align*}
\forall \mathit{ou} : \text{OpSUnit} \mid \mathit{ou} \in \mathit{os} & \bullet \mathit{ou}.\text{Rest} = \emptyset \Rightarrow \\
\forall \mathit{ou} : \text{OpSUnit} \mid \mathit{ou} \in \text{NormOS2b} (\mathit{os}) & \bullet \mathit{ou}.\text{Rest} = \emptyset \Rightarrow \\
\forall \mathit{ou} : \text{OpSUnit} \mid \mathit{ou} \in \text{NormOS2b} (\mathit{os}) & \bullet \text{OUNorm2} (\mathit{ou}).
\end{align*}
\]
Hence, it therefore follows that

\[
\text{OSNorm2a (NormOS2b (os)) \land \forall ou : OpSUnit | ou \in \text{NormOS2b (os)} \bullet \text{OUNorm2 (ou)} \Rightarrow \text{OSNorm2 (NormOS2b (os))}
\]

and so the theorem holds for this inner base case.

**Inner Inductive Case:** # os > # Heads (os)

As in the proof of theorem 26, for this inner inductive case the induction hypothesis is that the theorem holds for the conditions of the outer base case and for some natural number \(ni\) such that # os – # Heads (os) = ni, and then the induction step is to show that therefore it also holds for # os – # Heads (os) = ni + 1.

Here, though, the analysis of this step depends essentially on the fact that the recursive calls of NormOS2 eventually reduce any case where # os – # Heads (os) = ni to the base case for this induction, since it is only this base case that satisfies the condition OSNorm2a (os), and hence leads (as in the inner base case) to OSNorm2a (NormOS2b (os)) holding. This follows directly as a consequence of the fact that each recursive call replaces the pair of elements \(ou1\) and \(ou2\) by a single element constructed as MergeOU (ou1, ou2), so that # os is reduced by 1 while # Heads (os) is unchanged, and hence any case where # os – # Heads (os) = ni + 1 is reduced to one where # os – # Heads (os) = ni.

The rest of the argument then duplicates that of the inner base case, in that any inductive case reduces eventually to an invocation of NormOS2b (os), where OSNorm2a (os) holds. For this it is again necessary to show that:

(i) \(\text{OSNorm2a (NormOS2b (os))}\) holds,

(ii) \(\text{OSNorm2b (NormOS2b (os))}\) holds, and

(iii) \(\forall ou : \text{OpSUnit | ou \in \text{NormOS2b (os)} \bullet \text{OUNorm2 (ou)}\) holds,

and exactly the same arguments are used to establish each of these terms, and hence to establish that the theorem holds for this inner inductive case as well, and hence for both parts of the outer base case.

**Outer Inductive Case:** HeightOS (os) > 1

As in the proof of theorem 26, the induction hypothesis for this outer inductive case is that, for any natural number \(no > 1\), the theorem holds for all values \(os' : \text{OpSem}\) such that HeightOS (os') < no. The induction step is then to show from this that the theorem must also hold for any arbitrary value \(ou : \text{OpSem}\) such that HeightOS (os) = no. Again, suppose without loss of generality that any os must be such that HeightOS (os) = no, and so HeightOU (ou) ≤ no, and also assume that HeightOU (ou) > 1, to avoid the problem simply reducing to the base case.

Again, there are two parts to the proof of this case, corresponding to the base and recursive cases of the inner induction, and the arguments for each part are very similar to those for the outer base case.

**Inner Base Case:** # os – # Heads (os) = 0

Again, theorem 25 and the conditions of this theorem ensure that OSNorm2a (os) holds, and the conditions of this inner base case ensure that NormOS2 (os) invokes NormOS2b (os) directly, for which it is necessary to show that:

(i) \(\text{OSNorm2a (NormOS2b (os))}\) holds,

(ii) \(\text{OSNorm2b (NormOS2b (os))}\) holds, and

(iii) \(\forall ou : \text{OpSUnit | ou \in \text{NormOS2b (os)} \bullet \text{OUNorm2 (ou)}\)

Since we noted above that the result \(ou : \text{OpSUnit } \bullet ou,.TheAct = (\text{NormOU2 (ou)} .TheAct\) holds for all values of HeightOS (os), the arguments for (i) and (ii) are identical to those for the outer base case. For (iii) the argument here depends directly on the induction hypothesis, as follows. For any \(ou : \text{OpSUnit}\) such that \(ou \in os\), let \(os1 : \text{OpSem}\) denote \(ou,.Rest\). Then we have

\[
\text{OSNorm2 (NormOS2 (os1))}
\]

from the induction hypothesis

\[
\Rightarrow \text{OSNorm2 (NormOU2 (ou),Rest )}
\]

by the definition of NormOU2

\[
\Rightarrow \text{OUNorm2 (NormOU2 (ou))}
\]

by the definition of OUNorm2

\[
\Rightarrow \forall ou' : \text{OpSUnit | ou' \in NormOS2b (os) } \bullet ou' = \text{NormOU2 (ou)}
\]

by the definition of NormOS2b

\[
\Rightarrow \forall ou' : \text{OpSUnit | ou' \in NormOS2b (os) } \bullet \text{OUNorm2 (ou')}\]

Hence, all of the required properties (i), (ii) and (iii) hold, and so the theorem holds for this inner base case.

**Inner Inductive Case:** # os > # Heads (os)

The induction hypothesis for this inner inductive case is the same as for the inner inductive case of the outer base case, and so is the induction step, although of course the set of objects os that satisfy the conditions of the theorem will be different from the outer base case. The induction step again leads to the same three properties, which are established using exactly the same arguments as for the inner base case of this outer inductive case. Hence, the theorem holds for this inner
The third property to be established for these normalisation functions is the analogue of theorem 18, namely that they do not alter objects that are already normalised, and this is expressed as the following theorem.

**Theorem 28.**

\[
\forall os : \text{OpSem} \mid \text{OSNorm012} (os) \bullet \text{NormOS2} (os) = os \\
\land \forall ou : \text{OpSUnit} \mid \text{OUNorm012} (ou) \bullet \text{NormOU2} (ou) = ou.
\]

**Proof.**

The proof of this theorem has a simpler structure than those of theorems 26 and 27, as there is no need for an inner induction over the metric \# os - \# Heads (os), since this must be zero from the definition of OSNorm2a and theorem 25. Hence, we just have a single induction over the heights of the forests os and its constituent elements ou.

**Base case.**

The base case is that

- \( \text{HeightOS} (os) = 1 \Rightarrow \text{HeightOU} (ou) = 1 \Rightarrow \text{ou.Rest} = \emptyset \)

As in the proofs of theorems 26 and 27, the conditions for NormOS2 (os) to invoke MergeOU can not be satisfied, and so it simply invokes NormOS2b (os) directly. There are then two sub-cases to be considered, depending on the size of os.

Sub-case (i): \( \# os = 1 \Rightarrow os = \{ ou \} \)

\[
\Rightarrow \text{NormOS2} (os) = \{ \text{NormOU2} (ou) \} \\
\Rightarrow \text{NormOS2} (os) = \{ ou \} = os
\]

Sub-case (ii): \( \# os > 1 \), so that

\[
\Rightarrow \text{NormOS2} (os) = \{ \forall ou : \text{OpSUnit} \mid ou \in os \bullet \text{NormOU2} (ou) \} \\
\Rightarrow \text{NormOS2} (os) = \{ \forall ou : \text{OpSUnit} \mid ou \in os \bullet ou \} = os
\]

Hence, both halves of the theorem hold for both of these sub-cases, and so hold for the base case.

**Inductive case.**

As in the proofs of the outer inductive cases of theorems 26 and 27, the induction hypothesis is that, for any natural number \( n > 1 \), the theorem holds for all values os' : OpSem such that HeightOS (os') < n. The induction step is then to show from this that the theorem must also hold for any arbitrary value os : OpSem such that HeightOS (os) = n. Again, suppose without loss of generality that any os is such that HeightOS (os) = n, and so HeightOU (ou) ≤ n, and also suppose that HeightOU (ou) > 1, to avoid the problem simply reducing to the base case.

As in the base case, the conditions for NormOS2 (os) to invoke MergeOU can not be satisfied, and so it simply invokes NormOS2b (os) directly, and there are the same two sub-cases to be considered for the size of os.

Sub-case (i): \( \# os = 1 \Rightarrow os = \{ ou \} \)

\[
\Rightarrow \text{NormOS2} (os) = \{ \text{NormOU2} (ou) \} \\
\Rightarrow \text{NormOS2} (os) = \{ ou \} \text{ from OUNorm1 (ou)}
\]

and

\[
\Rightarrow \text{NormOU2} (ou) = \text{Cont1ActSem} (ou.TheAct, \text{NormOS2} (ou.Rest)) \\
\Rightarrow \text{NormOU2} (ou) = \text{Cont1ActSem} (ou.TheAct, ou.Rest) \text{ from the induction hypothesis}
\]

Then

\[
\text{OSNorm0} (os) \land \text{OSNorm1} (os) \Rightarrow \text{OUNorm0} (ou) \land \text{OUNorm1} (ou) \Rightarrow \text{ou.Rest} \neq \{ \text{EpsSem} \} \land \text{ou.Rest} \neq \{ \text{PhiSem} \} \\
\Rightarrow \text{Cont1ActSem} (ou.TheAct, ou.Rest) = \text{ContActSem} (ou.TheAct, ou.Rest) \\
\Rightarrow \text{NormOU2} (ou) = \text{ContActSem} (ou.TheAct, ou.Rest) \\
\Rightarrow \text{NormOU2} (ou) = ou \\
\Rightarrow \text{NormOS2} (os) = \{ ou \} = os
\]
An Operational Semantics for the Dataflow Algebra  A. J. Cowling

Sub-case (ii): \( \# \text{os} > 1 \), so that
\[
\text{OSNorm2b}(\text{os}) \Rightarrow \phi\text{Norm} \notin \text{os} \\
\implies \text{NormOS2}(\text{os}) = \{ \forall \text{ou} : \text{OpSUnit} \mid \text{ou} \in \text{os} \bullet \text{NormOU2}(\text{ou}) \} \\
\text{ou.Rest} \neq \emptyset \Rightarrow \text{ou.DoesAct}\text{from OUNorm1(ou)}
\]
and
\[
\text{NormOS2}(\text{os}) = \{ \forall \text{ou} : \text{OpSUnit} \mid \text{ou} \in \text{os} \bullet \text{ou} \} \\
\text{ou.Rest} \neq \emptyset \Rightarrow \text{NormOU2}(\text{ou}) = \text{Cont1ActSem(ou.TheAct, NormOS2(ou.Rest))} \\
\implies \text{NormOU2}(\text{ou}) = \text{Cont1ActSem(ou.TheAct, ou.Rest)} \\
\text{from the induction hypothesis}
\]

Hence, both halves of the theorem again hold for both of these sub-cases, and so hold for the inductive case. By induction, therefore, the theorem holds for all values of \( n \geq 1 \), and so holds.

This therefore establishes the obvious properties that are required for the normalisation functions, but in addition to these we also need to show that the transformations carried out by these functions are consistent with the properties of the DFA, as these need to be reflected within the semantics. Ultimately, of course, the demonstration of this will come when the semantic functions have been defined, and particularly in the form of the proofs that the axioms of the algebra are sound with respect to the semantics. At this stage, though, there is an important property that can be established, and this is that these transformations do not alter the set of possible actions that can occur as the next step in executing any construction in the DFA, as these actions are represented in the semantic structures.

In stating this property formally, the forbidden action needs to be treated as a special case. In the transition system the situations where the forbidden action could not occur as a possible action are covered by the guards in clauses (ix) and (x) of the definition of the transition relation, but (as noted previously) the normalisation functions do not rely on this property. Consequently, they are defined in a way that allows for the possibility that a semantic structure in zeroth normal form might include nodes that correspond to occurrences of the forbidden action that actually should not be executed. Thus, part of the process of transforming semantic structures into second normal form (specifically, the function NormOS2b) includes the removal of any such elements if they have arisen, and so the required property has to be stated in a form that allows for this. The required form is therefore the following theorem, which in some respects can be regarded as a generalisation of theorem 24.

\textbf{Theorem 29.}
\[
\forall \text{os} : \text{OpSem} \mid \text{OSNorm1( os)} \bullet \\
( \# \text{os} \leq 1 \Rightarrow \text{Heads(NormOS2(os))} = \text{Heads(os)} ) \land \\
( \# \text{os} > 1 \Rightarrow \text{Heads(NormOS2(os))} = \text{Heads(os)} - \{ \phi \} ).
\]

\textbf{Proof.}
Again, this proof involves a double induction, but unlike the proofs of theorems 26 and 27 the outer induction does not need to be over the height of the forest \text{os}, because the arguments used do not depend on this. Here, therefore, the outer induction is over the metric \( \# \text{os} - \# \text{Heads} (\text{os}) \), and then the inner induction is over \( \# \text{Heads} (\text{os}) \), where the first term of the result essentially acts as the base cases for this, and the second term of the result as the inductive cases.

\textbf{Outer Base Case:} \( \# \text{os} - \# \text{Heads}(\text{os}) = 0 \)

There are two parts to the proof of this case, corresponding to the base and recursive cases of the inner induction. As in the proofs of theorems 26 and 27, for this outer base case the conditions for NormOS2(\text{os}) to invoke MergeOU can not be satisfied, and so instead it simply invokes NormOS2b(\text{os}) directly.

\textbf{Inner Base Case:} \( \# \text{os} \leq 2 \)

For the invocation of NormOS2b(\text{os}) by NormOS2(\text{os}) we have six sub-cases, depending on the size of \text{os} and whether or not it contains either or both of the elements PhiSem or EpsSem, as follows.

Sub-case (i): \( \# \text{os} = 0 \), which gives
\[
\text{os} = \emptyset \Rightarrow \text{Heads}(\text{os}) = \emptyset \\
\text{os} = \emptyset \Rightarrow \text{NormOS2(\text{os})} = \emptyset \Rightarrow \text{Heads(NormOS2(\text{os}))} = \emptyset = \text{Heads(\text{os})}
\]
Sub-case (ii): \( \# os = 1 \), which gives
\[ \exists ou: \text{OpSUnit} \cup os = \{ ou \} \]

There are then two possibilities, depending on the value of \( ou.\text{DoesAct} \), and while the structure of the argument is the same for each the details are slightly different.

\( ou.\text{DoesAct} = \text{true} \Rightarrow \text{Heads (os) = \{ ou.TheAct \}} \)
\[ \Rightarrow \text{NormOS2 (os) = \{ NormOU2 (ou) \} = \{ Cont1ActSem (ou.TheAct, NormOS2 (ou.Rest)) \}} \]
\[ \Rightarrow \text{Heads (NormOS2 (os)) = Heads (os) \quad \text{from theorem 23}} \]

\( ou.\text{DoesAct} = \text{false} \Rightarrow ou = \text{EpsSem} \vee ou = \text{PhiSem} \quad \text{from theorem 12} \)
\[ \Rightarrow \text{Heads (os) = \{ GetHead (ou) \}} \]
\[ \Rightarrow \text{NormOS2 (os) = \{ NormOU2 (ou) \} = \{ ou \}} \]
\[ \Rightarrow \text{Heads (NormOS2 (os)) = \{ GetHead (ou) \} = Heads (os)} \]

Sub-case (iii): \( \# os = 2 \wedge \text{PhiSem} \notin os \wedge \text{EpsSem} \notin os \), which from theorem 12 gives
\[ \exists ou1, ou2: \text{OpSUnit} \mid ou1.\text{TheAct} \equiv/ \equiv ou2.\text{TheAct} \wedge ou1 \neq \text{PhiSem} \wedge ou2 \neq \text{PhiSem} \]
\[ os = \{ ou1, ou2 \} \]
\[ \Rightarrow \text{Heads (os) = \{ ou1.TheAct, ou2.TheAct \}} \]
\[ \Rightarrow \text{NormOS2 (os) = \{ NormOU2 (ou1), NormOU2 (ou2) \}} \]
\[ \Rightarrow \text{Heads (NormOS2 (os)) = \{ ou1.TheAct, ou2.TheAct \} = Heads (os) – \{ \phi \}} \]

Sub-case (iv): \( \# os = 2 \wedge \text{PhiSem} \in os \wedge \text{EpsSem} \notin os \), which (again from theorem 12) gives
\[ \exists ou: \text{OpSUnit} \mid ou \neq \text{PhiSem} \]
\[ os = \{ ou, \text{PhiSem} \} \]
\[ \Rightarrow \text{Heads (os) = \{ ou.TheAct, \phi \}} \]
\[ \Rightarrow \text{NormOS2 (os) = \{ NormOU2 (ou) \}} \]
\[ \Rightarrow \text{Heads (NormOS2 (os)) = \{ ou.TheAct \} = Heads (os) – \{ \phi \}} \]

Sub-case (v): \( \# os = 2 \wedge \text{PhiSem} \notin os \wedge \text{EpsSem} \in os \), which by the same argument from theorem 12 gives
\[ \exists ou: \text{OpSUnit} \mid ou \neq \text{PhiSem} \]
\[ os = \{ ou, \text{EpsSem} \} \]
\[ \Rightarrow \text{Heads (os) = \{ ou.TheAct, \epsilon \}} \]
\[ \Rightarrow \text{NormOS2 (os) = \{ NormOU2 (ou), EpsSem \}} \]
\[ \Rightarrow \text{Heads (NormOS2 (os)) = \{ ou.TheAct \} = Heads (os) – \{ \phi \}} \]

Sub-case (vi): \( \# os = 2 \wedge \text{PhiSem} \in os \wedge \text{EpsSem} \in os \), which gives
\[ os = \{ \text{EpsSem, PhiSem} \}
\[ \Rightarrow \text{Heads (os) = \{ \epsilon, \phi \}} \]
\[ \Rightarrow \text{NormOS2 (os) = \{ EpsSem \}} \]
\[ \Rightarrow \text{Heads (NormOS2 (os)) = \{ \epsilon \} = Heads (os) – \{ \phi \}} \]

Hence, the theorem holds for all six of these sub-cases, and so holds for this inner base case.

Inner Inductive Case: \( \# os > 2 \)

Here, the induction hypothesis is that the result holds for any \( \# os = ni \) such that \( ni \geq 2 \), and the inductive step is to show that therefore the result must also hold for \( \# os = ni + 1 \). We therefore have
\[ \# os > 2 \Rightarrow \exists ou: \text{OpSUnit} \mid ou \in os \cup \text{ou.\text{DoesAct} = true} \]
since from theorem 12 there can only be two possible elements of os that do not meet the condition \( ou.\text{DoesAct} = \text{true} \), namely EpsSem and PhiSem. Hence, let \( os1 = os – \{ ou \} \), so that \( \# os = \# os1 + 1 \).

Then, for the invocation of NormOS2b (os) by NormOS2 (os) we have the following argument.
\[ \text{Heads (os) = Heads (os1) \cup \{ ou.TheAct \}} \]
\[ \text{NormOS2 (os) = NormOS2 (os1) \cup \{ NormOU2 (ou) \}} \]
\[ \Rightarrow \text{Heads (NormOS2 (os)) = Heads (NormOS2 (os1)) \cup \{ ou.TheAct \}} \]
\[ = [ Heads (os1) – \{ \phi \}] \cup \{ ou.TheAct \} \quad \text{from the induction hypothesis} \]
\[ = [ Heads (os1) \cup \{ ou.TheAct \}] – \{ \phi \} \quad \text{since \{ ou.TheAct \} \cap \{ \phi \} = \emptyset} \]
\[ = \text{Heads (os) – \{ \phi \}} \]

Hence, the theorem holds for this inner inductive case, and so by induction it holds for all \( ni \), and hence holds for the outer base case.

Outer Inductive Case: \( \# os > \# \text{Heads (os)} \)

In similar fashion to the inner inductive case in the proofs of theorems 26 and 27, for this case the induction hypothesis is that the theorem holds for some natural number \( no \) such that \( \# os – \# \text{Heads (os)} = no \), and then the inductive step is to show that therefore it also holds for \( \# os – \# \text{Heads (os)} = no + 1 \).
Again, the analysis of this step depends essentially on the fact that the recursive calls of NormOS2 eventually reduce any case where # os − # Heads (os) = no to the base case for this induction. As before, this is because each recursive call replaces the pair of elements \( ou1 \) and \( ou2 \) by a single element constructed as MergeOU (\( ou1, ou2 \)), and it again follows immediately that this replacement reduces # os by 1, and that # Heads (os) is unchanged.

Hence, if we let \( os1 \) denote the parameter that is eventually passed to the call of NormOS2b, then it follows directly that \( \text{Heads (NormOS2 (os))} = \text{Heads (NormOS2 (os1))} \), and that \( \text{Heads (os)} = \text{Heads (os1)} \).

But \( os1 \) satisfies the conditions of the outer base case, for which the theorem holds, and hence it holds directly for this case too, without needing to appeal explicitly to the induction hypothesis, although its use is implicit in the way in which we have analysed the whole sequence of calls of MergeOU.

Hence, the theorem holds for both of the outer cases, and so holds.

There are then two other simple properties of second normal form that are indicated by the proofs of these last three theorems, and that are sufficiently useful to be worth stating formally, as the following theorems.

**Theorem 30.**

\[ \forall os : \text{OpSem} \mid \text{OSNorm012 (os) } \bullet \mbox{ # Heads (os) } = \# os. \]

**Proof.**

The proof is essentially by contradiction. There are three cases for the relationship between # Heads (os) and # os that notionally could arise, as follows.

Case (i): # Heads (os) > # os, but this can not occur, as the definition of Heads (os) does not allow for more elements to be constructed than os has.

Case (ii): # Heads (os) < # os. For this to occur, from the definition of Heads (os) it would have to be the case that \( \exists ou1, ou2 : \text{OpSUnit} \mid ou1 \in os \land ou2 \in os \land ou1 \neq ou2 \bullet \text{GetHead (ou1)} = \text{GetHead (ou2)} \).

From theorem 12, it would then have to be the case in this that

\[ \text{ou1 = \text{EpsSem} \land ou2 = \text{EpsSem} \land ou1 = \text{PhiSem} \land ou2 = \text{PhiSem} } \]

and hence that \( \text{ou1.DoesAct = true} \land \text{ou2.DoesAct = true} \). But, from the definition of GetHead, this would imply that \( \text{ou1.TheAct = ou2.TheAct} \), which gives a contradiction with the definition of OSNorm2 (os). Hence, this case can not occur.

Case (iii): # Heads (os) = # os. Since the other two cases can not occur, this must hold, and therefore the theorem holds.

**Theorem 31.**

\[ \forall os : \text{OpSem} \mid \text{OSNorm012 (os) } \bullet \mbox{ ( os } 
eq \{ \text{PhiSem} \} \Rightarrow \text{PhiSem } \notin \text{os} ) \land ( \text{Heads (os)} \neq \{ \phi \} \Rightarrow \phi \notin \text{Heads (os)} ). \]

**Proof.**

The proof has three cases for # os, as follows.

Case (i): # os = 0 ⇒ ( os \neq \{ \text{PhiSem} \} \land \text{PhiSem } \notin \text{os} )

and  
\( \Rightarrow \text{Heads (os)} = \emptyset \Rightarrow ( \text{Heads (os)} \neq \{ \phi \} \land \phi \notin \text{Heads (os)} ) \)

Case (ii): # os = 1 ⇒ \exists ou : \text{OpSUnit } \bullet \mbox{ os } = \{ \text{ou} \} \land \text{Heads (os)} = \{ \text{GetHead (ou)} \}.

Since OSNorm0 (os) \land OSNorm1 (os) \land OSNorm2 (os), from theorems 5, 6 and 12 there are just three possible sub-cases for the value of os, as follows.

Sub-case (ii)(a): \( \text{ou = PhiSem } \Rightarrow \text{Heads (os)} = \{ \phi \} \), which is consistent with the theorem, since the left-hand sides of both conjuncts are false.
Sub-case (ii)(b): $ou = \text{EpsSem} \Rightarrow (os \neq \{\PhiSem\} \land \PhiSem \notin os)$ and $\Rightarrow \text{Heads}(os) = \{\varepsilon\} \Rightarrow (\text{Heads}(os) \neq \{\phi\} \land \phi \notin \text{Heads}(os))$

Sub-case (ii)(c): $Ea: PA \bullet ou = \text{FinalActSem}(a) \lor ou = \text{FinalAbActSem}(a) \lor (os1: \text{OpSem} \bullet ou = \text{ContActSem}(a, os1))$ and $\Rightarrow (os \neq \{\PhiSem\} \land \PhiSem \notin os)$ and $\Rightarrow \text{Heads}(os) = \{a\} \Rightarrow (\text{Heads}(os) \neq \{\phi\} \land \phi \notin \text{Heads}(os))$

Case (iii): $\#os > 1 \Rightarrow os \neq \{\PhiSem\}$ and $\Rightarrow \PhiSem \notin os$ from $\text{OSNorm2}(os)$ and $\Rightarrow \#\text{Heads}(os) = \#os > 1$ from theorem 30 and $\Rightarrow \text{Heads}(os) \neq \{\phi\}$ from theorem 22 and $\Rightarrow \phi \notin \text{Heads}(os)$

Hence the theorem holds for all cases and sub-cases, and therefore holds.

Having established these properties of the normal forms and the transformation functions that produce them, the next step is to use these normal forms in order to define a set of semantic functions for the various constructions of the DFA. Before doing this, though, there are various auxiliary functions that are required, and these need to be defined first, and their properties established.

### 7. Auxiliary Functions for the Semantics

An obvious set of auxiliary functions to define at this stage consists of those that act effectively as complements to the function $\text{Heads}$, in the sense that they represent how the execution of a DFA construction will proceed after some action has been performed. This set contains firstly a group of three functions to model the control state of the next configuration, and so they are called respectively $\text{ContinuesAfter}$, $\text{EndsAfter}$ and $\text{AbEndsAfter}$, and each has the same signature, namely $\text{OpSem} \times \text{Act} \rightarrow \text{Boolean}$. Their definitions assume that their first parameter is in second normal form (and hence also in zeroth and first normal forms), and they are

\[
\begin{align*}
\text{ContinuesAfter}(os, a) & \equiv \text{if } \exists\ ou : \text{OpSUnit} \mid ou \in os \bullet \text{GetHead}(ou) = a \text{ then ou.NextState = continues else false fi} \\
\text{EndsAfter}(os, a) & \equiv \text{if } \exists\ ou : \text{OpSUnit} \mid ou \in os \bullet \text{GetHead}(ou) = a \text{ then ou.NextState = normend else false fi} \\
\text{AbEndsAfter}(os, a) & \equiv \text{if } \exists\ ou : \text{OpSUnit} \mid ou \in os \bullet \text{GetHead}(ou) = a \text{ then ou.NextState = abnormend else false fi}
\end{align*}
\]

Note that it is not necessary here to impose the precondition $a \in \text{Heads}(os)$, although the results returned by the functions are not particularly meaningful if this precondition does not hold. Where this precondition does hold we have the property that exactly one of the three functions delivers the result $true$, and otherwise all three of them deliver the result $false$.

The other function in this set then applies to the case where the execution does continue after some action, and it produces the forest that represents how the execution continues. This function is therefore called $\text{RestAfter}$, and it has the signature $\text{OpSem} \times \text{Act} \rightarrow \text{OpSem}$. Again, its definition assumes that the first parameter is in zeroth, first and second normal forms, and is

\[
\text{RestAfter}(os, a) \equiv \text{if } \exists\ ou : \text{OpSUnit} \mid ou \in os \bullet \text{GetHead}(ou) = a \text{ then ou.Rest else } \emptyset \text{ fi}
\]

Again, it is not necessary here to impose the precondition $a \in \text{Heads}(os)$, nor the precondition that the execution does actually continue, as given by $\text{ContinuesAfter}(os, a) = true$, since if either of these does not hold then the function simply delivers the empty set. On the other hand, the empty forest is not the valid semantics for any construction in the DFA, and so if either of these preconditions does not hold then the result that is returned by the function will not have the significance that is intended for it.
The way in which these functions complement Heads can then be expressed formally as the property defined by the following theorem, where the conditions of the theorem mean that it is sufficient to use the function ContActSem, as the cases that might need to be handled specially by Cont1ActSem could not arise.

**Theorem 32.**

\[ \forall os : \text{OpSem} \mid \text{OSNorm012 (os)} \bullet \\
\quad os = \{ \forall a : \text{Act} \mid a \in \text{Heads (os)} \bullet \\
\quad \quad \text{if EndsAfter } (os, a) \text{ then if } a \in \text{PA} \text{ then FinalActSem } (a) \text{ else EpsSem } \text{ fi} \\
\quad \quad \text{elsif AbEndsAfter } (os, a) \text{ then if } a \in \text{PA} \text{ then FinalAbActSem } (a) \text{ else PhiSem } \text{ fi} \\
\quad \quad \text{else ContActSem } (a, \text{RestAfter } (os, a) ) \}
\]

**Proof.**

The proof involves three main cases, for the different possibilities for the value of \(a\).

Case (i): \(a = \varepsilon\)

\[ \Rightarrow \exists ou : \text{OpSUnit} \mid ou \in os \bullet ou = \text{EpsSem} \quad \text{from theorem 22} \]

\[ \Rightarrow ou.\text{NextState} = \text{normend} \]

\[ \Rightarrow \text{EndsAfter } (os, a) = \text{true} \]

and since \(a \notin \text{PA}\), the theorem holds for this case.

Case (ii): \(a = \phi\)

\[ \Rightarrow \exists ou : \text{OpSUnit} \mid ou \in os \bullet ou = \text{PhiSem} \quad \text{from theorem 22} \]

\[ \Rightarrow ou.\text{NextState} = \text{abnormend} \]

\[ \Rightarrow \text{EndsAfter } (os, a) = \text{false} \land \text{AbEndsAfter } (os, a) = \text{true} \]

and since \(a \notin \text{PA}\), the theorem holds for this case.

Case (iii): \(a \in \text{PA}\)

\[ \Rightarrow \exists ou : \text{OpSUnit} \mid ou \in os \bullet ou.\text{DoesAct} = \text{true} \land ou.\text{TheAct} = a. \]

There are then three sub-cases to be considered, depending on the value of ou.\text{NextState}.

Sub-case (iii)(a): ou.\text{NextState} = \text{normend}

\[ \Rightarrow ou = \text{FinalActSem } (a) \quad \text{from theorem 5} \]

\[ \Rightarrow \text{EndsAfter } (os, a) = \text{true} \]

and since \(a \in \text{PA}\), the theorem holds for this sub-case.

Sub-case (iii)(b): ou.\text{NextState} = \text{abnormend}

\[ \Rightarrow ou = \text{FinalAbActSem } (a) \quad \text{from theorem 5} \]

\[ \Rightarrow \text{EndsAfter } (os, a) = \text{false} \land \text{AbEndsAfter } (os, a) = \text{true} \]

and since \(a \in \text{PA}\), the theorem holds for this sub-case.

Sub-case (iii)(c): ou.\text{NextState} = \text{continues}

\[ \Rightarrow ou.\text{Rest} \neq \emptyset \quad \text{since OSNorm0 (os)} \]

Since we also have OSNorm2 (os), theorem 30 implies that this element ou must be unique, and hence

\[ \text{RestAfter } (os, a) = ou.\text{Rest} \]

Combining this with ou.\text{DoesAct} = \text{true} and ou.\text{TheAct} = a gives

\[ ou = \text{ContActSem } (a, \text{RestAfter } (os, a)) \]

\[ \Rightarrow \text{EndsAfter } (os, a) = \text{false} \land \text{AbEndsAfter } (os, a) = \text{false} \]

and so the theorem holds for this sub-case too.

Hence, the theorem holds for all three of these sub-cases, and so it holds for case (iii), and hence for all three of the main cases, and so it holds.

\[ \blacksquare \]

Another function that can be defined, and which will be useful in proving some of the theorems for consistency of the semantics with the axioms of the algebra, is one that in some respects is analogous to RestAfter, ContinuesAfter and the others, but it actually returns the element of type OpSUnit that corresponds to an element of Heads. This function is
therefore called GetUnit, and it has signature $\text{Act} \times \text{OpSem} \rightarrow \text{OpSUnit}$, but because there is no null value defined for the type $\text{OpSUnit}$ it is necessary to impose a formal precondition on it, so that its definition is as follows.

$$a \in \text{Heads (os)} \Rightarrow \text{GetUnit (a, os)} \equiv \exists \ ou : \text{OpSUnit} \mid \text{ou.TheAct} = a \land \text{ou} \in \text{os} \bullet \text{ou}$$

It should also be noted that this definition may be non-deterministic, if there is more than one such element $\text{ou}$ in $\text{os}$, but it can be guaranteed that this element is unique (and hence the function deterministic) if $\text{OSNorm2 (os)}$ holds. This function can then be used to define an equivalence relation over the semantic structures, which can be used as an alternative to the equality relation if the structures are in second normal form. This equivalence relation is represented by a pair of functions, one called $\text{OSEqN2}$ and the other called $\text{OUEqN2}$. These functions have signatures $\text{OpSem} \times \text{OpSem} \rightarrow \text{Boolean}$ and $\text{OpSUnit} \times \text{OpSUnit} \rightarrow \text{Boolean}$ respectively, and they are defined as follows.

$$\text{OSEqN2 (os1, os2)} \equiv \text{if } \text{os1} = \emptyset \land \text{os2} = \emptyset \text{ then } \text{true}
\text{elseif } \text{Heads (os1)} = \text{Heads (os2)}
\text{then } \forall a : \text{Act} \mid a \in \text{Heads (os1)} \bullet \text{OUEqN2 (GetUnit (a, os1), GetUnit (a, os2))}
\text{else } \text{false}
\text{fi}$$

$$\text{OUEqN2 (ou1, ou2)} \equiv \text{ou1.TheAct} = \text{ou2.TheAct}
\land \text{ou1.NextState} = \text{ou2.NextState}
\land \text{OSEqN2 (ou1.Rest, ou2.Rest)}$$

Then, the fact that this gives an alternative definition of the equality relationship for structures that are in second normal form is expressed as the following theorem.

**Theorem 33.**

$$\forall os1, os2 : \text{OpSem} \mid \text{OSNorm012 (os1)} \land \text{OSNorm012 (os2)} \bullet \text{OSEqN2 (os1, os2)} \Leftrightarrow os1 = os2.$$

**Proof.**
The proof uses the implicit recursive definition of equality between sets in terms of equality between their elements, although a formal statement of this would require a separate notation for the test for an empty set from the usual equality symbol. If we avoid this by notating this test informally, then for the type $\text{OpSem}$ this definition can be stated as follows.

$$(\text{os1} = \text{os2}) \Leftrightarrow \text{if } \text{os1} \text{ is empty } \land \text{os2} \text{ is empty } \text{ then } \text{true}
\text{else } \exists \ ou1, ou2 : \text{OpSUnit} \mid \text{ou1} \in \text{os1} \land \text{ou2} \in \text{os2} \bullet \text{ou1} = \text{ou2} \land (\text{os1} – \{\text{ou1}\}) = (\text{os2} – \{\text{ou2}\})
\text{fi}$$

The proof is then essentially by induction over the height of the forest $\text{os1}$, and within this it uses case analysis of the structure of the arbitrary elements $\text{ou1}$ and $\text{ou2}$, as these are processed by $\text{OUEqN2}$. Here the conditions $\text{OSNorm2 (os1)}$ and $\text{OSNorm2 (os2)}$ ensure that, for any element $a$ of $\text{Heads (os1)}$, these elements $\text{ou1}$ and $\text{ou2}$ are unique, and so the case analysis is primarily over the possible forms for the value of $a$.

**Base case.**
The base case is that

$$\text{HeightOS (os1)} = 1 \Rightarrow \text{HeightOU (ou1)} = 1$$

From theorem 5, since $\text{OSNorm0 (os1)} \Rightarrow \text{OUNorm0 (ou1)}$ and $\text{OSNorm0 (os2)} \Rightarrow \text{OUNorm0 (ou2)}$, there are just four sub-cases for the constructions of each of $\text{ou1}$ and $\text{ou2}$, which have the common features that $\text{ou1.Rest} = \emptyset$ and hence $\text{ou1.NextState} \neq \text{continues}$, and similarly $\text{ou2.Rest} = \emptyset$ and hence $\text{ou2.NextState} \neq \text{continues}$. These four sub-cases for the construction of $\text{ou1}$ are derived from the three possible cases for the structure of $a$, as follows.

- **Base sub-case (i):** $a = \varepsilon$ \Rightarrow $\text{ou1} = \text{EpsSem} \land \text{ou2} = \text{EpsSem}$
  \Rightarrow $(\text{OUEqN2 (ou1, ou2)} = \text{true}) \land ((\text{ou1} = \text{ou2}) = \text{true})$

- **Base sub-case (ii):** $a = \phi$ \Rightarrow $\text{ou1} = \text{PhiSem} \land \text{ou2} = \text{PhiSem}$
  \Rightarrow $(\text{OUEqN2 (ou1, ou2)} = \text{true}) \land ((\text{ou1} = \text{ou2}) = \text{true})$

- **Base sub-case (iii):** $a \in \text{PA}$
  \Rightarrow $(\text{ou1} = \text{FinalActSem (a)}) \lor (\text{ou1} = \text{FinalAbActSem (a)})$
  \land (\text{ou2} = \text{FinalActSem (a)}) \lor (\text{ou2} = \text{FinalAbActSem (a)})$
This gives four possible combinations for the constructions of \( \text{ou1} \) and \( \text{ou2} \), but these can be reduced to just two sub-cases, one in which they have the same construction and one in which they have different constructions. The analyses for these are as follows.

\[
\begin{align*}
\text{ou1} = \text{ou2} & \Rightarrow \text{OUEqN2 (ou1, ou2)} \land \text{OUEqN2 (ou1, ou2)} \Rightarrow \text{ou1} = \text{ou2} \\
\text{ou1} \neq \text{ou2} & \Rightarrow \neg \text{OUEqN2 (ou1, ou2)} \land \neg \text{OUEqN2 (ou1, ou2)} \Rightarrow \text{ou1} \neq \text{ou2}
\end{align*}
\]

Hence, the base case of the induction follows directly from the combination of these three sub-cases.

**Inductive case.**

For the inductive case we have the same three sub-cases for the construction of \( \text{a} \), but the first two are not inductive, and the analysis of them is the same as for the base case. Thus, the only case that needs to be analysed by induction is \( \text{a} \in \text{PA} \).

For this the induction hypothesis is that, for any natural number \( n > 1 \), the theorem holds for all values \( \text{os}' : \text{OpSem} \) such that \( \text{HeightOS (os')} < n \), and so the induction step is then to show from this that the theorem must also hold for any arbitrary value \( \text{os1} : \text{OpSem} \) such that \( \text{HeightOS (os1)} = n \). Without loss of generality, therefore, suppose that any \( \text{os1} \) must be such that \( \text{HeightOS (os1)} = n \), so that by the definition of \( \text{HeightOS} \), for any arbitrary element \( \text{ou1} \) of \( \text{os1} \) it must be the case that \( \text{HeightOU (ou1)} \leq n \).

Then, the constructions of \( \text{ou1} \) and \( \text{ou2} \) must be such that

\[
\exists \text{os3} : \text{OpSem} \ni \text{ou1} = \text{ContActSem (a, os3)} \land \exists \text{os4} : \text{OpSem} \ni \text{ou2} = \text{ContActSem (a, os4)}
\]

so that \( \text{HeightOS (os3)} < n \land \text{HeightOS (os4)} < n \), and we then have

\[
(\text{ou1.NextState } = \text{continues } = \text{ou2.NextState}) \land (\text{ou1.TheAct } = \text{a } = \text{ou2.TheAct})
\]

Hence

\[
\text{OUEqN2 (ou1, ou2)} = \text{OSEqN2 (os3, os4)} \\
\Rightarrow \text{os3} = \text{os4}
\]

from the induction hypothesis

Hence the theorem holds for the inductive case as well, and so by induction it holds for all values of \( n \geq 1 \), and so holds.

The final auxiliary function to be introduced is one to represent the sequential composition of any arbitrary pair of DFA sequences \( s1 \) and \( s2 \). Defining this actually requires a pair of functions, one called \( \text{SeqCompOS} \) that will operate over the forest corresponding to \( s1 \), and the other called \( \text{SeqCompOU} \) that will operate over individual trees in this forest. The function \( \text{SeqCompOS} \) has signature \( \text{OpSem} \times \text{OpSem} \rightarrow \text{OpSem} \), and it is defined as follows

\[
\text{SeqCompOS (os1, os2)} \equiv \begin{cases} 
\text{if } \text{os1} = \emptyset & \text{then } \text{os2} \\
\text{elsif } \text{os2} = \emptyset & \text{then } \text{os1} \\
\text{else} & \text{FlattenOS ( } \{ \forall \text{ou} : \text{OpSUnit} \ni \text{ou } \in \text{os1} \ni \text{SeqCompOU (ou, os2)} \} ) \\
\end{cases}
\]

Then, the function \( \text{SeqCompOU} \) has signature \( \text{OpSUnit} \times \text{OpSem} \rightarrow \text{OpSem} \), and is defined as follows

\[
\text{SeqCompOU (ou, os)} \equiv \begin{cases} 
\text{if } \text{ou}.\text{DoesAct} \\
\text{then if } \text{ou.NextState} = \text{continues} \\
\text{then } \{ \text{Cont1ActSem (ou.TheAct, SeqCompOS (ou.Rest, os))} \} \\
\text{elsif } \text{ou.NextState} = \text{normend} \\
\text{then } \{ \text{Cont1ActSem (ou.TheAct, os)} \} \\
\text{else } \{ \text{ou} \} \\
\text{fi} \\
\text{elsif } \text{ou.NextState} = \text{continues} \\
\text{then } \text{SeqCompOS (ou.Rest, os)} \\
\text{elsif } \text{ou.NextState} = \text{normend} \\
\text{then } \text{os} \\
\text{else } \{ \text{ou} \} \\
\text{fi} \\
\text{fi}
\end{cases}
\]
Both of these definitions assume that the parameters to the functions are in zeroth normal form, and provided that this is the case then they produce results in zeroth normal form. They do not, though, necessarily assume that their parameters are in either first or second strict normal form, and so the definition of \( \text{SeqCompOU} \) is formulated to cover explicitly the case where \( \text{ou.}\text{DoesAct} = \text{false} \) and \( \text{ou.}\text{NextState} = \text{continues} \), which could not occur if its parameters were in first normal form. On the other hand, \( \text{SeqCompOU} \) has to handle the cases where either its parameter \( \text{os} \) or the result of the recursive call that makes of \( \text{SeqCompOS} \) has a single element that does not perform an action, and for these the use of the function \( \text{Cont1ActSem} \) rather than \( \text{ContActSem} \) ensures that, if its parameters are in first normal form, then the result is too. By contrast, though, the functions do not try to ensure that their results are in second normal form, even if their parameters are.

This property of the functions with respect to zeroth and first normal forms is expressed as the following theorem.

**Theorem 34.**

\[
\forall \text{os1, os2 : OpSem} \quad \text{OSNorm01(os1)} \land \text{OSNorm01(os2)} \\
\Rightarrow \text{OSNorm0(SeqCompOS(os1, os2))} \land \text{OSNorm1(SeqCompOS(os1, os2))}.
\]

**Proof.**

For the two trivial cases given by \( \text{os1} = \emptyset \) or \( \text{os2} = \emptyset \) the proof follows immediately from the conditions of the theorem. For the non-trivial cases the proof is essentially by induction over the height of the forest \( \text{os1} \), and within this it uses case analysis of the structure of an arbitrary element \( \text{ou} \) of \( \text{os1} \), as this is reflected in the structure of the function \( \text{SeqCompOU} \). In applying this analysis, both the base and inductive cases of the proof rely on the observations that, from the definitions of \( \text{OSNorm0} \) and \( \text{OSNorm1} \),

\[
\text{OSNorm0(SeqCompOS(os1, os2))} \Leftrightarrow \forall \text{ou : OpSUnit} \mid \text{ou} \in \text{os1} \Rightarrow \text{OUNorm0(SeqCompOU(ou, os2))}
\]

and

\[
\text{OSNorm1(SeqCompOS(os1, os2))} \Leftrightarrow \forall \text{ou : OpSUnit} \mid \text{ou} \in \text{os1} \Rightarrow \text{OUNorm1(SeqCompOU(ou, os2))}
\]

Hence, in both the base case and the inductive case, it is sufficient to prove that the result of the theorem holds for any arbitrary element \( \text{ou} \) of \( \text{os1} \), and then rely on this observation to extend the result to the whole of \( \text{os1} \).

**Base case.**

The base case is that

\[
\text{HeightOS(os1)} = 1 \Rightarrow \text{HeightOU(ou)} = 1
\]

From theorem 5, since \( \text{OSNorm0(os1)} \Rightarrow \text{OUNorm0(ou)} \), there are just four sub-cases for the construction of \( \text{ou} \), which have the common feature that \( \text{ou.Rest} = \emptyset \), and hence \( \text{ou.NextState} \neq \text{continues} \). These four sub-cases for the construction of \( \text{ou} \) are as follows.

Base sub-case (i):

\[
\text{ou} = \text{EpsSem} \Rightarrow \text{SeqCompOU(ou, os2)} = \text{os2}
\]

\[
\Rightarrow \text{OSNorm0(SeqCompOS(os1, os2))} \quad \text{since OSNorm0(os2)}
\]

and

\[
\Rightarrow \text{OSNorm1(SeqCompOS(os1, os2))} \quad \text{since OSNorm1(os2)}
\]

Base sub-case (ii):

\[
\exists a : \text{PA} \mid \text{ou} = \text{FinalActSem(a)}
\]

\[
\Rightarrow \text{SeqCompOU(ou, os2)} = \{ \text{Cont1ActSem(ou.TheAct, os2)} \}
\]

\[
\Rightarrow \text{OSNorm0(SeqCompOS(os1, os2))} \land \text{OSNorm1(SeqCompOS(os1, os2))} \quad \text{from theorem 13}
\]

Base sub-case (iii):

\[
\text{ou} = \text{PhiSem} \Rightarrow \text{SeqCompOU(ou, os2)} = \{ \text{ou} \}
\]

and

\[
\text{OSNorm0(os1)} \Rightarrow \text{OUNorm0(ou)} \Rightarrow \text{OSNorm0(SeqCompOS(os1, os2))}
\]

and

\[
\text{OSNorm1(os1)} \Rightarrow \text{OUNorm1(ou)} \Rightarrow \text{OSNorm1(SeqCompOS(os1, os2))}
\]

Base sub-case (iv):

\[
\exists a : \text{PA} \mid \text{ou} = \text{FinalAbActSem(a)}
\]

\[
\Rightarrow \text{SeqCompOU(ou, os2)} = \{ \text{ou} \}
\]

and

\[
\text{OSNorm0(os1)} \Rightarrow \text{OUNorm0(ou)} \Rightarrow \text{OSNorm0(SeqCompOS(os1, os2))}
\]

and

\[
\text{OSNorm1(os1)} \Rightarrow \text{OUNorm1(ou)} \Rightarrow \text{OSNorm1(SeqCompOS(os1, os2))}
\]

Hence, the base case of the induction follows directly from the combination of these four sub-cases.

**Inductive case.**

For any natural number \( n > 1 \), the induction hypothesis is that the theorem holds for all values \( \text{os'} : \text{OpSem} \) such that \( \text{HeightOS(os')} < n \), and so the induction step is then to show from this that the theorem must also hold for any arbitrary value \( \text{os1 : OpSem} \) such that \( \text{HeightOS(os1)} = n \). Without loss of generality, therefore, suppose that any \( \text{os1} \) must be such that \( \text{HeightOS(os1)} = n \), so that by the definition of \( \text{HeightOS} \), for any arbitrary element \( \text{ou} \) of \( \text{os1} \) it must be the
case that \( \text{HeightOU} (ou) \leq n \). Also without loss of generality, assume that \( \text{HeightOU} (ou) > 1 \), since otherwise the problem will simply reduce to the base case.

From the definitions of \( \text{OSNorm0} \) and \( \text{OSNorm1} \) it therefore follows that
\[
\text{OSNorm0} (os1) \Rightarrow \text{OUNorm0} (ou) \Rightarrow \text{OSNorm0} (ou.\text{Rest}), \quad \text{and since } ou.\text{Rest} \neq \emptyset, \quad \text{that}
\]
\[
\text{OUNorm0} (ou) \Rightarrow ou.\text{NextState} = \text{continues} \quad \text{and}
\]
\[
\text{OUNorm1} (ou) \Rightarrow ou.\text{DoesAct}.
\]

Hence,
\[
\text{SeqCompOU} (ou, os2) = \text{Cont1ActSem} (ou.\text{TheAct}, \text{SeqCompOS} (ou.\text{Rest}, os2))
\]

Then, since \( \text{HeightOS} (ou.\text{Rest}) \leq n \), we have from the induction hypothesis
\[
\text{OSNorm0} (\text{SeqCompOS} (ou.\text{Rest}, os2)) \land \text{OSNorm1} (\text{SeqCompOS} (ou.\text{Rest}, os2))
\]
\[
\Rightarrow \text{OSNorm0} (\text{SeqCompOU} (ou, os2)) \land \text{OSNorm1} (\text{SeqCompOU} (ou, os2)) \quad \text{from theorem 13}
\]
\[
\Rightarrow \text{OSNorm0} (\text{SeqCompOS} (os1, os2)) \land \text{OSNorm1} (\text{SeqCompOS} (os1, os2))
\]

Hence the inductive case holds, and so by induction the theorem holds for all values of \( n \geq 1 \), and so holds.


8. Semantic Properties of the Auxiliary Functions

The properties defined in this section concern the inter-relationships between the various functions that have been defined already, such as distributive and associative properties that reflect the equivalent properties in the semantics of DFA expressions, as given by the axioms of the DFA. Algebraically, these properties also allow expressions involving the functions to be manipulated, in a way that corresponds to the notion of introduction and elimination rules in logic, and indeed these properties are used in this way in many of the subsequent proofs.

The first such property to be defined here derives directly from the definition of \( \text{NormOU2} \), and provides a form of elimination rule for calls of \( \text{NormOU2} \). It is expressed as the following theorem.

**Theorem 35.**
\[
\forall a : PA, os : \text{OpSem} \mid \text{IsActive} (os) \land os \neq \{ \text{EpsSem}, \text{PhiSem} \} \Rightarrow \text{NormOU2} (\text{ContActSem} (a, os)) = \text{ContActSem} (a, \text{NormOS2} (os)).
\]

**Proof.**
\[
\text{IsActive} (os) \Rightarrow os \neq \emptyset \Rightarrow \text{NormOU2} (\text{ContActSem} (a, os)) = \text{Cont1ActSem} (a, \text{NormOS2} (os))
\]
and
\[
\text{IsActive} (os) \land os \neq \{ \text{EpsSem}, \text{PhiSem} \} \Rightarrow \exists ou : \text{OpUnit}, a : PA \mid ou \in os \land \text{GetHead} (ou) = a
\]
\[
\Rightarrow \text{NormOS2} (os) \neq \{ \text{EpsSem} \} \land \text{NormOS2} (os) \neq \{ \text{PhiSem} \}
\]
\[
\Rightarrow \text{Cont1ActSem} (a, \text{NormOS2} (os)) = \text{ContActSem} (a, \text{NormOS2} (os))
\]
\[
\Rightarrow \text{NormOU2} (\text{ContActSem} (a, os)) = \text{ContActSem} (a, \text{NormOS2} (os)).
\]

The next two properties concern the function \( \text{SeqCompOS} \), and the circumstances under which it maintains the property that its parameters are active. These are expressed formally as the following theorems.

**Theorem 36.**
\[
\forall os1, os2 : \text{OpSem} \mid \text{OSNorm012} (os1) \land \text{OSNorm012} (os2) \Rightarrow \text{IsActive} (os1) \Rightarrow \text{IsActive} (\text{SeqCompOS} (os1, os2)).
\]

**Proof.**
For the trivial case where \( os2 = \emptyset \), \( \text{SeqCompOS} (os1, os2) = os1 \), and so the proof follows immediately from the conditions of the theorem. For the non-trivial case the proof has two main cases, depending on the value of \( \# os1 \). In it, we let:
os : OpSem be defined as os = SeqCompOS (os1, os2),
ou1 : OpSUnit denote any arbitrary element of os1,
a : PA denote GetHead (ou1), and
ou : OpSUnit denote any arbitrary element of os, such that GetHead (ou) = a.

Then, the two main cases are as follows.

Case (i): # os1 = 1 ⇒ os1 = { ou1 }
From theorem 15 there are five possible constructions for ou1, of which two (EpsSem and PhiSem) are excluded by the conditions of this theorem, so that the remaining ones give rise to three sub-cases, as follows.

(i): ou1 = FinalActSem (a) ⇒ ou = Cont1ActSem (a, os2) ⇒ os ≠ { EpsSem } ∧ os ≠ { PhiSem }
(ii): ou1 = FinalAbActSem (a) ⇒ ou = FinalAbActSem (a) ⇒ os ≠ { EpsSem } ∧ os ≠ { PhiSem }
(iii): ∃ os1x : OpSem • ou1 = ContActSem (a, os1x) ⇒ ou = Cont1ActSem (a, SeqCompOS (os1x, os2))
and this gives three possible constructions for ou, depending on the value of SeqCompOS (os1x, os2), as follows:

SeqCompOS (os1x, os2) = { EpsSem } ⇒ ou = FinalActSem (a)
⇒ os ≠ { EpsSem } ∧ os ≠ { PhiSem }
SeqCompOS (os1x, os2) = { PhiSem } ⇒ ou = FinalAbActSem (a)
⇒ os ≠ { EpsSem } ∧ os ≠ { PhiSem }
otherwise ou = ContActSem (a, SeqCompOS (os1x, os2)) ⇒ os ≠ { EpsSem } ∧ os ≠ { PhiSem }

Case (ii): # os1 > 1 ⇒ # os > 1 ⇒ os ≠ { EpsSem } ∧ os ≠ { PhiSem }

Hence the theorem holds for all cases and sub-cases, and therefore holds.

Theorem 37.
∀ os1, os2 : OpSem | OSNorm012 (os1) ∧ OSNorm012 (os2) •
os1 ≠ { PhiSem } ∧ IsActive (os2) ⇒ IsActive (SeqCompOS (os1, os2)).

Proof.
As in the proof of theorem 36, for the trivial case where os1 = ∅, SeqCompOS (os1, os2) = os2, and so the proof follows immediately from the conditions of the theorem. For the non-trivial case the proof has two main cases, depending on the value of # os1. In it, we let:
os : OpSem be defined as os = SeqCompOS (os1, os2),
ou1 : OpSUnit denote any arbitrary element of os1,
a : PA denote GetHead (ou1), and
ou : OpSUnit denote any arbitrary element of os, such that GetHead (ou) = a.

Then, the two main cases are as follows.

Case (i): # os1 = 1 ⇒ os1 = { ou1 }
From theorem 15 there are again five possible constructions for ou1, but here one (PhiSem) is excluded by the conditions of this theorem, so that the remaining ones give rise to four sub-cases, as follows.

(i): ou1 = EpsSem ⇒ SeqCompOU (ou1, os2) = os2 ⇒ os = os2 ⇒ IsActive (os)
(ii): ou1 = FinalActSem (a) ⇒ ou = Cont1ActSem (a, os2) ⇒ os ≠ { EpsSem } ∧ os ≠ { PhiSem }
(iii): ou1 = FinalAbActSem (a) ⇒ ou = FinalAbActSem (a) ⇒ os ≠ { EpsSem } ∧ os ≠ { PhiSem }
(iv): ∃ os1x : OpSem • ou1 = ContActSem (a, os1x) ⇒ ou = Cont1ActSem (a, SeqCompOS (os1x, os2))
and this gives three possible constructions for ou, depending on the value of SeqCompOS (os1x, os2), as follows:

SeqCompOS (os1x, os2) = { EpsSem } ⇒ ou = FinalActSem (a)
⇒ os ≠ { EpsSem } ∧ os ≠ { PhiSem }
SeqCompOS (os1x, os2) = { PhiSem } ⇒ ou = FinalAbActSem (a)
⇒ os ≠ { EpsSem } ∧ os ≠ { PhiSem }
otherwise ou = ContActSem (a, SeqCompOS (os1x, os2)) ⇒ os ≠ { EpsSem } ∧ os ≠ { PhiSem }

Case (ii): # os1 > 1 ⇒ # os > 1 ⇒ os ≠ { EpsSem } ∧ os ≠ { PhiSem }

Hence the theorem holds for all cases and sub-cases, and therefore holds.

The next group of properties define how various functions distribute over the operation of set union, as this is applied to forests. The first of them is for the operation Heads, and is expressed as the following theorem.
Theorem 38.
\( \forall \text{os1, os2} : \text{OpSem} \ni \text{Heads (os1 } \cup \text{ os2)} = \text{Heads (os1)} \cup \text{Heads (os2)}. \)

Proof.
\( \forall \text{os1, os2} : \text{OpSem} \ni \\
\text{Heads (os1 } \cup \text{ os2)} = \{ \forall \text{ou} : \text{OpSUnit} | \text{ou} \in \text{os1 } \lor \text{ou} \in \text{os2} \ni \text{GetHead (ou)} \} \\
\lor \{ \forall \text{ou} : \text{OpSUnit} | \text{ou} \in \text{os1} \ni \text{GetHead (ou)} \} \\
= \text{Heads (os1)} \cup \text{Heads (os2)} \)

The next of these distributive properties is for the transformation to first normal form, and is expressed as the following theorem.

Theorem 39.
\( \forall \text{os1, os2} : \text{OpSem} | \text{OSNorm0 (os1)} \land \text{OSNorm0 (os2)} \land \text{os1} \neq \emptyset \land \text{os2} \neq \emptyset \\
\text{NormOS1 (os1 } \cup \text{ os2)} = \text{NormOS1 (os1)} \cup \text{NormOS1 (os2)}. \)

Proof.
\( \text{NormOS1 (os1 } \cup \text{ os2)} = \text{FlattenOS (} \{ \forall \text{ou} : \text{OpSUnit} | \text{ou} \in \text{os1 } \lor \text{ou} \in \text{os2} \ni \text{NormOS1Unit (ou)} \} ) \\
\lor \{ \forall \text{ou} : \text{OpSUnit} | \text{ou} \in \text{os1} \ni \text{NormOS1Unit (ou)} \} \\
= \text{FlattenOS (} \{ \forall \text{ou} : \text{OpSUnit} | \text{ou} \in \text{os1} \ni \text{NormOS1Unit (ou)} \} ) \lor \\
\text{FlattenOS (} \{ \forall \text{ou} : \text{OpSUnit} | \text{ou} \in \text{os2} \ni \text{NormOS1Unit (ou)} \} ) \\
= \text{NormOS1 (os1)} \cup \text{NormOS1 (os2)} \)

There is an analogous result for second normal form, but it is not strictly a distributive property, since the distributed version still requires an overall call of the function NormOS2, and so it might be more appropriate to describe it as an introduction rule for NormOS2 into set unions. Before defining this result, though, there is an intermediate result that is required for its proof, and this is expressed as the following theorem.

Theorem 40.
\( \forall \text{os} : \text{OpSem} | \text{OSNorm01 (os)} \ni \text{IsActive (os)} \Rightarrow \\
\text{NormOS2 (os)} = \text{NormOS2 (os – { PhiSem })} \\
\land \text{NormOS2 (os } \cup \text{ { PhiSem })} = \text{NormOS2 (os)} \\
\land \text{NormOS2 (os } \cup \text{ { EpsSem })} = \text{NormOS2 (os)} \cup \text{ { EpsSem }}. \)

Proof.
The proof uses the result of theorem 33, in that it analyses the construction of typical elements of the forests for each of the three possibilities for the head element. Throughout it, we let the following variables be defined as:
\( \text{os1} = \text{NormOS2 (os)}, \text{os1r} = \text{NormOS2 (os – { PhiSem })}, \)
\( \text{os2} = \text{os } \cup \text{ { PhiSem }}, \text{os2n} = \text{NormOS2 (os2)}, \)
\( \text{os3} = \text{NormOS2 (os } \cup \text{ { EpsSem })}, \text{os3r} = \text{os1 } \cup \text{ { EpsSem }}, \)
\( \text{h} = \text{Heads (os)}, \text{h1} = \text{Heads (os1)}, \text{h1r} = \text{Heads (os1r)}, \text{h2} = \text{Heads (os2)}, \text{h2n} = \text{Heads (os2n)}, \)
\( \text{h3} = \text{Heads (os3)}, \text{h3r} = \text{Heads (os3r)}, \)
and as appropriate (meaning, when these objects actually exist), then for any action a we let
\( \text{ou} : \text{OpSUnit be such that ou } \in \text{os } \land \text{GetHead (ou)} = a, \)
\( \text{ou1} : \text{OpSUnit be such that ou1 } \in \text{os1 } \land \text{GetHead (ou1a)} = a \Rightarrow \text{OUNorm2 (ou1)}, \)
\( \text{ou1r} : \text{OpSUnit be such that ou1r } \in \text{os1r } \land \text{GetHead (ou1ra)} = a \Rightarrow \text{OUNorm2 (ou1r)}, \)
\( \text{ou2} : \text{OpSUnit be such that ou2 } \in \text{os2 } \land \text{GetHead (ou2a)} = a, \)
\( \text{ou2n} : \text{OpSUnit be such that ou2n } \in \text{os2n } \land \text{GetHead (ou2na)} = a \Rightarrow \text{OUNorm2 (ou2n)}, \)
\( \text{ou3} : \text{OpSUnit be such that ou3 } \in \text{os3 } \land \text{GetHead (ou3a)} = a \Rightarrow \text{OUNorm2 (ou3)}, \) and
\( \text{ou3r} : \text{OpSUnit be such that ou3r } \in \text{os3r } \land \text{GetHead (ou3ra)} = a. \)

Then the three cases for the possible construction of a are as follows.

Case (i): \( a \in PA \)
\[ a \in h \Rightarrow a \in h1 \land a \in h1r \land a \in h2 \land a \in h2n \land a \in h3 \land a \in h3r \]
and \( \Rightarrow \text{ou1r } = \text{ou1 } \land \text{ou2n } = \text{ou1 } \land \text{ou3r } = \text{ou3}. \)
Case (ii): \( a = \phi \)
For this case there are two possibilities, depending on whether or not \( \phi \) is in \( h \).
Sub-case (ii)(a) \( \phi \in h \land \text{IsActive} \ (os) \)
\[ \Rightarrow \# os > 1 \Rightarrow h_1 = h - \{ \phi \} \]
\[ \Rightarrow \phi \notin h_1 \Rightarrow \phi \notin h_3r \]
and
\[ \Rightarrow os2 = os \Rightarrow h_2 = h_1 = h \Rightarrow h_2n = h_1 \Rightarrow \phi \notin h_2n \]
and
\[ \Rightarrow h_3 = (h \cup \{ \varepsilon \}) - \{ \phi \} \]
\[ \Rightarrow \phi \notin h_3 \]
Then \( \phi \in h \land \text{OSNorm0} \ (os) \land \text{OSNorm1} \ (os) \Rightarrow ou = \text{PhiSem} \)
\[ \Rightarrow \phi \notin \text{Heads (os} - \{ \text{PhiSem} \}) \Rightarrow \phi \notin h_1r \]
Sub-case (ii)(b) \( \phi \notin h \)
\[ \Rightarrow \phi \notin h_1 \land \phi \notin h_1r \land \phi \notin h_3 \land \phi \notin h_3r \]
and
\[ \phi \notin h \land \text{IsActive} \ (os) \Rightarrow \# os2 > 1 \Rightarrow h_2n = h_2 - \{ \phi \} \]
\[ \Rightarrow \phi \notin h_2n \]
Sub-case (iii)(b) \( \phi \notin h \)
\[ \Rightarrow \phi \notin h_1 \land \phi \notin h_1r \land \phi \notin h_2 \land \phi \notin h_3 \land \phi \notin h_3r. \]
Hence, for both possibilities \( \phi \notin h_1 \land \phi \notin h_1r \) and \( \phi \notin h_2n \) and \( \phi \notin h_3 \land \phi \notin h_3r \).
Case (iii): \( a = \varepsilon \)
Again, for this case there are two possibilities, depending on whether or not \( \varepsilon \) is in \( h \).
Sub-case (iii)(a) \( \varepsilon \in h \)
\[ \Rightarrow \varepsilon \in h_1 \land \varepsilon \in h_1r \land \varepsilon \in h_2 \land \varepsilon \in h_2n \land \varepsilon \in h_3 \land \varepsilon \in h_3r \]
and
\[ \Rightarrow ou = \text{EpsSem} \]
\[ \Rightarrow ou1 = \text{EpsSem} \land ou1r = \text{EpsSem} \land ou2 = \text{EpsSem} \land ou2n = \text{EpsSem} \land ou3 = \text{EpsSem} \land ou3r = \text{EpsSem} \]
Sub-case (iii)(b) \( \varepsilon \notin h \)
\[ \Rightarrow \varepsilon \notin h_1 \land \varepsilon \notin h_1r \land \varepsilon \notin h_2 \land \varepsilon \notin h_2n \]
and
\[ \Rightarrow \varepsilon \in h_3 \land \varepsilon \in h_3r \land ou3 = \text{EpsSem} \land ou3r = \text{EpsSem}. \]
Hence the theorem holds for all three cases, and so holds.

Given this intermediate result, then the introduction rule for \( \text{NormOS2} \) into set unions can be defined. There are three possible forms for it, depending on whether \( \text{NormOS2} \) is introduced into one parameter of the set union, or the other, or both. The following theorem defines the first form, where it is introduced into one parameter, and then the other two forms follow from this.

**Theorem 41.**
\[
\forall os1, os2 : \text{OpSem} \mid os1 \neq \emptyset \land \text{OSNorm01 (os1)} \land os2 \neq \emptyset \land \text{OSNorm01 (os2)} \Rightarrow \\
\text{NormOS2 (os1} \cup \text{os2)} = \text{NormOS2 (NormOS2 (os1} \cup \text{os2)}).
\]

**Proof.**
The proof here is considerably more complicated than for the proof of theorem 39, as it is by induction over the heights of the forests \( os1 \) and \( os2 \). Throughout it, we let the following variables be defined as:
\[
os = os1 \cup os2, osn = \text{NormOS2 (os)}, osn1 = \text{NormOS2 (os1)}, osr = osn1 \cup os2, osm = \text{NormOS2 (osr)}, h1 = \text{Heads (os1)}, h2 = \text{Heads (os2)}, h = \text{Heads (os)}, hn = \text{Heads (osn)}, h1n = \text{Heads (os1n)}, hr = \text{Heads (osr)}, hnr = \text{Heads (osrn)},
\]
and as appropriate (meaning, when these objects actually exist), then for any action \( a \) we let
\[
ou1a, ou1b : \text{OpSUnit} \text{ be such that } ou1a \in os1 \land ou1b \in os1 \land \\
\text{GetHead (ou1a)} = a \land \text{GetHead (ou1b)} = a \land ou1a \neq ou1b
\]
\[
ou2a, ou2b : \text{OpSUnit} \text{ be such that } ou2a \in os2 \land ou2b \in os2 \land \\
\text{GetHead (ou2a)} = a \land \text{GetHead (ou2b)} = a \land ou2a \neq ou2b
\]
\[
ou : P \text{ OpSUnit be such that } ou \subseteq os \land \forall ou' : \text{OpSUnit} \mid ou' \in ou \land \text{GetHead (ou') = a,}
\]
\[
oun : \text{OpSUnit be such that } oun \in osn \land \text{GetHead (oun)} = a \Rightarrow \text{OUNorm2 (oun)},
\]
\[
oun1 : \text{OpSUnit be such that } oun1 \in osn1 \land \text{GetHead (oun1)} = a \Rightarrow \text{OUNorm2 (oun1)}
\]
\[
oour : P \text{ OpSUnit be such that } our \subseteq osr \land \forall ou' : \text{OpSUnit} \mid ou' \in our \land \text{GetHead (ou') = a, and}
\]
\[
oourn : \text{OpSUnit be such that } ourn \in osrn \land \text{GetHead (ourn)} = a \Rightarrow \text{OUNorm2 (ourn)}.
\]

Then, each of the base and inductive cases gives rise to three sub-cases, depending on whether an arbitrary element \( a \) is in either \( h_1 \) or \( h_2 \), or both, since the case where it is in neither can be ignored. Within each of these sub-cases the argument then needs to consider three possibilities for the value of \( a \), namely that it may be in \( PA \) or may be either of the two
constants ε or φ, where each of these possibilities corresponds to different constructions of the elements ou1a, ou2a, etc, and so there may also be different combinations of these possibilities that need to be analysed.

For one or two of these possibilities or combinations of possibilities, this analysis shows directly that osn = osrn, which is the required result. For most of them, though, it consists of showing for that element a that it either is or is not in both h and hr, and hence either is or is not in both hn and hrn, so that hn and hrn have the same elements. Also, for each element a that is in both hn and hrn the analysis then shows too that oun = ourn, so that GetUnit (a, osn) = GetUnit (a, osrn), from which it then follows by theorem 33 that osn = osrn.

Where the analysis needs to show that oun = ourn, it is based on the fact that, because both of these are in second normal form, the outermost calls of NormOS2 that have produced them will have combined different elements using MergeOU. Thus, for each case the analysis involves calculating the effects of these calls of MergeOU, as given by the intermediate values ou, oun, ourn, and oun, and relying on the properties of MergeOU given by theorems 19, 20 and 21 where it is being applied to a set of elements ou or our. To make the working clear, the elements of these sets will be written in an order that reflects the elements from which they are derived, with semi-colons being used to separate those derived from elements of os1 from those derived from elements of os2, even where this results in duplicate values appearing in the sets.

**Base case.**

The base case is that HeightOS (os1) = 1 ∧ HeightOS (os2) = 1, so that for any arbitrary element a the possible constructions of ou1a and ou1b are given by theorem 5. The three sub-cases and the three possibilities within each for the value of a are as follows.

**Sub-case (i):**    a ∈ h1 ∧ a ≠ h2

For the possibility a ∈ PA, from theorem 5 the only possible constructions of ou1a and ou1b are FinalActSem (a) and FinalAbActSem (a), and either or both of these could occur. Whichever, we have a ∈ h, a ∈ hn, a ∈ h1n, a ∈ hr and a ∈ hrn.

If FinalActSem (a) occurs, then ou = { FinalActSem (a) }, oun = FinalActSem (a), ou1n = FinalActSem (a), ou2a = FinalActSem (a), and oun = FinalActSem (a), so that oun = oun.

If FinalAbActSem (a) occurs, then ou = { FinalAbActSem (a) }, oun = FinalAbActSem (a), ou1n = FinalAbActSem (a), and oun = FinalAbActSem (a), so that oun = oun.

If both occur, then ou = { FinalActSem (a), FinalAbActSem (a) }, oun = FinalActSem (a), ou1n = FinalActSem (a), and oun = FinalAbActSem (a), so that oun = oun.

For the possibility a = ε, the only possible construction of ou1a is EpsSem, for which we have:

ε ∈ h, ε ∈ hn, ε ∈ h1n, ε ∈ hr and ε ∈ hrn, and then

ou = { EpsSem }, oun = EpsSem, ou1n = EpsSem, and oun = EpsSem, so that oun = oun.

For the possibility a = φ, the only possible construction of ou1a is PhiSem, and since φ ∈ h2 it follows that there is at least one element ou2a ≠ PhiSem. There are then two sub-cases of this, depending on # os1.

- # os1 = 1 ⇒ os = { PhiSem } ∪ os2 and ⇒ os1n = { PhiSem } ⇒ osr = { PhiSem } ∪ os2
  ⇒ os = osr ⇒ osn = osrm.
- # os1 > 1 ⇒ # os > 1 ⇒ φ ∈ hrn from theorem 29

and ⇒ φ ≠ h1n ⇒ φ ≠ hr ⇒ φ ≠ hrn.

Hence the theorem holds for all the possible values of a in this sub-case.

**Sub-case (ii):**    a ≠ h1 ∧ a ∈ h2

In this case, for any a the only contribution to ou or to our comes from ou2a or ou2b, and irrespective of the actual constructions of ou2a or ou2b this contribution is the same for both ou and our. Hence ou = our ⇒ oun = ourn.

**Sub-case (iii):**    a ∈ h1 ∧ a ∈ h2

As in sub-case (i), for the possibility a ∈ PA the only possible constructions for ou1a and ou1b are FinalActSem (a) and FinalAbActSem (a), and either or both of these could occur, and there are also the same three possibilities for ou2a and ou2b, giving nine different combinations to be considered. Common to all of them is that

a ∈ h, a ∈ hn, a ∈ h1n, a ∈ hr and a ∈ hrn.

The calculations for these nine different combinations are as follows.
An Operational Semantics for the Dataflow Algebra

A. J. Cowling

(a) If FinalActSem (a) occurs in os1 and in os2, then ou = \{ FinalActSem (a); FinalActSem (a) \}, ou1n = FinalActSem (a), ou1n = FinalActSem (a), our = \{ FinalActSem (a); FinalActSem (a) \}, and oum = FinalActSem (a), so that oum = oum.

(b) If FinalActSem (a) occurs in os1 and FinalAbActSem (a) occurs in os2, then ou = \{ FinalActSem (a); FinalAbActSem (a) \}, oum = FinalActSem (a), ou1n = FinalActSem (a), our = \{ FinalActSem (a); FinalAbActSem (a) \}, and oum = FinalActSem (a), so that oum = oum.

(c) If FinalActSem (a) occurs in os1 and both occur in os2, then ou = \{ FinalActSem (a); FinalActSem (a), FinalAbActSem (a) \}, oum = FinalActSem (a), ou1n = FinalActSem (a), our = \{ FinalActSem (a); FinalActSem (a), FinalAbActSem (a) \}, and oum = FinalActSem (a), so that oum = oum.

(d) If FinalAbActSem (a) occurs in os1 and FinalActSem (a) occurs in os2, then ou = \{ FinalAbActSem (a); FinalActSem (a) \}, oum = FinalActSem (a), ou1n = FinalAbActSem (a), our = \{ FinalAbActSem (a); FinalActSem (a) \}, and oum = FinalActSem (a), so that oum = oum.

(e) If FinalAbActSem (a) occurs in os1 and in os2, then ou = \{ FinalAbActSem (a); FinalAbActSem (a) \}, oum = FinalAbActSem (a), ou1n = FinalAbActSem (a), our = \{ FinalAbActSem (a); FinalAbActSem (a) \}, and oum = FinalAbActSem (a), so that oum = oum.

(f) If both occur in os1 and FinalActSem (a) occurs in os2, then ou = \{ FinalActSem (a), FinalActSem (a) \}, oum = FinalActSem (a), ou1n = FinalActSem (a), our = \{ FinalActSem (a), FinalActSem (a) \}, and oum = FinalActSem (a), so that oum = oum.

(g) If both occur in os1 and FinalActSem (a) occurs in os2, then ou = \{ FinalActSem (a); FinalActSem (a) \}, FinalActSem (a); FinalActSem (a) \}, oum = FinalActSem (a), ou1n = FinalActSem (a), our = \{ FinalActSem (a); FinalActSem (a) \}, and oum = FinalActSem (a), so that oum = oum.

(h) If both occur in os1 and FinalAbActSem (a) occurs in os2, then ou = \{ FinalActSem (a), FinalActSem (a) \}, FinalAbActSem (a) \}, oum = FinalActSem (a), ou1n = FinalAbActSem (a), our = \{ FinalActSem (a), FinalAbActSem (a) \}, and oum = FinalActSem (a), so that oum = oum.

(i) If both occur in os1 and both occur in os2, then ou = \{ FinalActSem (a); FinalActSem (a), FinalActSem (a), FinalActSem (a) \}, FinalActSem (a) \}, oum = FinalActSem (a), ou1n = FinalActSem (a), ou1n = FinalActSem (a), FinalActSem (a), FinalActSem (a) \}, and oum = FinalActSem (a), so that oum = oum.

Hence, for each of these nine combinations oum = oum, and so the theorem holds for the possibility a ∈ PA.

For the possibility a = ε, from theorem 5 the only possible constructions of ou1a and ou2a are EpsSem. Thus, we have ε ∈ h, ε ∈ h, ε ∈ h1, ε ∈ hr and ε ∈ hrr, and then ou = \{ EpsSem; EpsSem \}, oum = EpsSem, ou1n = EpsSem, our = \{ EpsSem; EpsSem \}, and oum = EpsSem, so that oum = oum.

For the possibility a = φ, from theorem 5 the only possible constructions of ou1a and ou2a are PhiSem, and there are then four combinations, depending on # os1 and # os2.

(a) # os1 = 1 ∧ # os2 = 1 ⇒ os1 = \{ PhiSem \} ∧ os2 = \{ PhiSem \} ⇒ os = \{ PhiSem \} and ⇒ os1n = \{ PhiSem \} ∧ osr = \{ PhiSem \} ⇒ os = osr ⇒ oum = oum.

(b) # os1 = 1 ∧ # os2 > 1 ⇒ os1 = \{ PhiSem \} ∧ os2 = \{ PhiSem \} ⇒ os = osr ⇒ oum = oum.

(c) # os1 > 1 ∧ # os2 = 1 ⇒ os1 = \{ PhiSem \} ∧ os2 = \{ PhiSem \} ⇒ osn = NormOS2 (os1 \{ PhiSem \}) = NormOS2 (os1) from theorem 40 and ⇒ osr = NormOS2 (os1 \{ PhiSem \}) = NormOS2 (NormOS2 (os1) \{ PhiSem \}) = NormOS2 (NomOS2 (os1)) = NormOS2 (os1) from theorem 40 ⇒ osn = osrn.

(d) # os1 > 1 ∧ # os2 = 1 ⇒ # os > 1 ⇒ φ ∈ h

and ⇒ φ ∈ h1 ⇒ \# h1 ≥ 1 ⇒ hr > 1 ⇒ φ ∈ hrr from theorem 29.
Hence the theorem holds for all the possible values of \( a \) in this sub-case too, and so holds for the base case.

**Inductive case.**
The inductive case is \((\text{HeightOS} (\text{os1}) = n \land \text{HeightOS} (\text{os2}) \leq n) \lor (\text{HeightOS} (\text{os1}) \leq n \land \text{HeightOS} (\text{os2}) = n)\) for any natural number \( n > 1 \). The induction hypothesis for this case is that the theorem holds for all \( \text{os1} \) and \( \text{os2} \) such that \( \text{HeightOS} (\text{os1}) < n \land \text{HeightOS} (\text{os2}) < n \), and the induction step is therefore to prove from this that it also holds for this inductive case. Within the three sub-cases for which of the sets of heads contain \( a \), and the three possible constructions of \( a \) for each, there are some possibilities and combinations of them that simply duplicate ones in the base case, and so the detail of these will not be repeated in what follows.

Sub-case (i): \( a \in h1 \land a \notin h2 \)

For the possibility \( a \in \text{PA} \), in addition to the only possible constructions of \( \text{ou1a} \) and \( \text{ou1b} \) as \( \text{FinalActSem} (a) \) and \( \text{FinalAbActSem} (a) \) which occurred in the base case, we could have either or both of

\[
\exists \text{os1x} : \text{OpSem} \bullet \text{ou1a} = \text{ContActSem} (a, \text{os1x}) \quad \text{and} \quad \exists \text{os1y} : \text{OpSem} \bullet \text{ou1b} = \text{ContActSem} (a, \text{os1y})
\]

and, of course, we could have further elements \( \text{ou1c} \) and so on, to any arbitrary number.

The various combinations of \( \text{FinalActSem} (a) \) and \( \text{FinalAbActSem} (a) \) on their own do not need to be considered further here, as they are identical to those in the base case, and so the only combinations that do need to be analysed here are those that involve one or more elements constructed by \( \text{ContActSem} \). For these combinations the set of elements to be combined could in principle require an induction over the size of the set, but in practice we can make this induction implicit by defining suitable constructions over the sets of elements that are used in the argument, as follows. Firstly, the notion of defining \( \text{ou} \) and \( \text{our} \) as sets is extended to treating the relevant sub-sets of \( \text{os1} \) as variables that are indexed over the element \( a \), which are denoted by \( \text{os1s}[a] \), where

\[
\text{os1s}[a] = \{ \forall \text{ou1a} : \text{OpSUnit} | \text{ou1a} \in \text{os1} \land \text{GetHead} (\text{ou1a}) = a \bullet \text{ou1a} \}
\]

Then, the analysis here is primarily concerned with the case that

\[
\exists \text{ou1a} : \text{OpSUnit} \bullet \{ \text{EpsSem} \}
\]

and for this case it can avoid considering the internal structure of any such elements \( \text{os1x} \), although for all of them it will rely on the property that

\[
\text{OSNorm1} (\text{os1}) \Rightarrow \text{IsActive} (\text{os1x})
\]

Under these conditions, the repeated applications of \( \text{MergeOU} \) to all of the elements of \( \text{os1s}[a] \) will produce an object that will be denoted \( \text{ou1m} : \text{OpSUnit} \), and which will be constructed as

\[
\text{ou1m} = \text{ContActSem} (a, \text{os1s}[a])
\]

where the object \( \text{os1s}[a] \) is essentially the union of the descendant trees of each of the objects in \( \text{os1s}[a] \), as these objects are modified by the operation of \( \text{MergeOU} \). Using \( \text{FlattenOS} \) to construct the union, this object can therefore be defined to also cover the cases of \( \text{FinalActSem} (a) \) and \( \text{FinalAbActSem} (a) \), as

\[
\text{os1a} = \text{FlattenOS} \{ \forall \text{ou1a} : \text{OpSUnit} | \text{ou1a} \in \text{os1s}[a] \bullet
\begin{cases}
\text{if} & \exists \text{os1x} : \text{OpSem} \bullet \text{ou1a} = \text{ContActSem} (a, \text{os1x}) \\
\text{else if} & \text{ou1a} = \text{ContActSem} (a, \text{os1s}[a]) \\
\text{else} & \emptyset
\end{cases}
\}
\]

Then, the subsequent application of \( \text{NormOU2} \) to such an object \( \text{ou1m} \) will, from theorem 35, result in the application of \( \text{NormOS2} \) to the object \( \text{os1a} \), and this will have two effects. One, which we do not need to analyse, will be to recursively invoke \( \text{MergeOU} \) for its elements, and the other will be to remove any element constructed as \( \text{PhiSem} \). To represent this second effect, we define an object \( \text{os1Un}[a] : \text{OpSem} \) to represent a partly normalised version of \( \text{os1a} \), where this object is defined as

\[
\text{os1Un}[a] = \text{os1a} - \{ \text{PhiSem} \}
\]

so that

\[
\text{NormOS2} (\text{os1Un}[a]) = \text{NormOS2} (\text{os1a})
\]

from theorem 40.

Given these constructions, then the required analysis is that \( a \in h, a \in hn, a \in h1n, a \in hr \) and \( a \in hm \), and the values of the corresponding elements can be calculated as follows.

\[
\begin{align*}
\text{ou} &= \{ \text{os1s}[a] \}, \\
\text{oun} &= \text{NormOU2} (\text{ContActSem} (a, \text{os1s}[a]) ) \\
&= \text{ContActSem} (a, \text{NormOS2} (\text{os1s}[a]) ) \\
&= \text{ContActSem} (a, \text{NormOS2} (\text{os1Un}[a]) ) \\
\text{ou1n} &= \text{ContActSem} (a, \text{NormOS2} (\text{os1Un}[a]) ) = \text{ContActSem} (a, \text{NormOS2} (\text{os1Un}[a]) ) , \\
\text{our} &= \{ \text{ContActSem} (a, \text{NormOS2} (\text{os1Un}[a]) ) \}
\end{align*}
\]

from theorem 35.
This general case, along with the possible constructions for os1s[a] that were analysed in the base case, thus establishes the theorem for all the elements that can arise for the possibility of \( a \in \text{PA} \).

For the possibilities \( a = \varepsilon \) or \( a = \phi \) the arguments are identical to the base case, and do not need to be repeated here. Hence, the theorem holds for all the possible values of \( a \) in this sub-case.

Sub-case (ii): \( a \not\in h1 \wedge a \in h2 \)
As in the base case, for any \( a \) the only contribution to \( ou \) or to \( our \) comes from \( ou2a \) or \( ou2b \), and irrespective of the actual constructions of \( ou2a \) or \( ou2b \) this contribution is the same for both \( ou \) and \( our \). Hence \( ou = our \Rightarrow ou = ourn \).

Sub-case (iii): \( a \in h1 \wedge a \in h2 \)
Similarly to sub-case (i), for the possibility \( a \in \text{PA} \), in addition to the possible constructions of \( ou1a \), \( ou1b \), \( ou2a \) and \( ou2b \) as \( \text{FinalActSem} (a) \) and \( \text{FinalAbActSem} (a) \) which occurred in the base case, we could have either or both of

\[ \exists os1 : \text{OpSem} \Rightarrow ou1a = \text{ContActSem} (a, os1x) \text{ and } \exists os1y : \text{OpSem} \Rightarrow ou1b = \text{ContActSem} (a, os1y) \]

and similarly either or both of

\[ \exists os2x : \text{OpSem} \Rightarrow ou2a = \text{ContActSem} (a, os2x) \text{ and } \exists os2y : \text{OpSem} \Rightarrow ou2b = \text{ContActSem} (a, os2y) \]

and again we could have further elements \( ou1c \), \( ou2c \) and so on, to any arbitrary number.

To analyse these we introduce objects \( os2s[a] : \text{OpSem} \), \( ou2m : \text{OpSUnit} \), \( os2U[a] : \text{OpSem} \) and \( os2Un[a] : \text{OpSem} \), with constructions that are exactly analogous to those of \( os1s[a] \), \( ou1m \), \( os1U[a] \) and \( os2U[a] \). Then, since each of the subsets \( os1s[a] \) and \( os2s[a] \) contains at least one element constructed by \( \text{ContActSem} \) (for which the height is therefore greater than one), and possibly also elements constructed by \( \text{FinalActSem} \) or \( \text{FinalAbActSem} \), once the main combination of \( os1s[a] \) and \( os2s[a] \) has been analysed, the only other ones that need to be considered are firstly the three between \( os1s[a] \) and either or both of FinalActSem (a) and FinalAbActSem (a), and secondly the equivalent three combinations for \( os2s[a] \).

As in the base case, for all seven of these combinations we have \( a = h \), \( a \in h1n \), \( a \in hr \) and \( a \in hmn \), and so it is only the calculations for the element values that need to be given. These are as follows.

(a): For \( os1s[a] \) with \( os2s[a] \),
\[
ou = \{ os1s[a], os2s[a] \},
\]
\[
oun = \text{NormOU2} (\text{ContActSem} (a, os1U[a] \cup os2U[a] ))
\]
\[
\text{from theorem 35}
\]
\[
= \text{ContActSem} (a, \text{NormOS2} (os1U[a] \cup os2U[a] ))
\]
\[
\text{from theorem 35}
\]
\[
o1n = \text{ContActSem} (a, \text{NormOS2} (os1U[a] ))
\]
\[
\text{from theorem 40,}
\]
\[
o1m = \text{ContActSem} (a, \text{NormOS2} (os1U[a] ) \cup \{ \text{EpsSem} \})
\]
\[
\text{from theorem 40,}
\]
\[
o1n = \text{ContActSem} (a, \text{NormOS2} (os1U[a] ))
\]
\[
= \text{ContActSem} (a, \text{NormOS2} (os1U[a] ) \cup \{ \text{EpsSem} \})
\]
\[
\text{from theorem 28}
\]

(b): For \( os1s[a] \) with \( \text{FinalActSem} (a) \),
\[
ou = \{ os1s[a], \text{FinalActSem} (a) \},
\]
\[
oun = \text{NormOU2} (\text{MergeOU} (\text{ContActSem} (a, os1U[a] ), \text{FinalActSem} (a) ))
\]
\[
\text{from theorem 35}
\]
\[
= \text{ContActSem} (a, \text{NormOS2} (os1U[a] ) \cup \{ \text{EpsSem} \})
\]
\[
\text{from theorem 40,}
\]
\[
o1n = \text{ContActSem} (a, \text{NormOS2} (os1U[a] ) \cup \{ \text{EpsSem} \})
\]
\[
\text{from theorem 35}
\]
\[
o1n = \text{ContActSem} (a, \text{NormOS2} (os1U[a] ))
\]
\[
= \text{ContActSem} (a, \text{NormOS2} (os1U[a] ) \cup \{ \text{EpsSem} \})
\]
\[
\text{from theorem 28}
\]

(c): For \( os1s[a] \) with \( \text{FinalAbActSem} (a) \),
\[
ou = \{ os1s[a], \text{FinalAbActSem} (a) \},
\]
oun = NormOU2 (MergeOU (ContActSem (a, os1U[a] ), FinalAbActSem (a) ) )
    = NormOU2 (ContActSem (a, os1U[a] ) )
    = ContActSem (a, NormOS2 (os1U[a] ) )
from theorem 35
ou1n = ContActSem (a, NormOS2 (os1U[a] ) )
    = ContActSem (a, NormOS2 (os1U[a] ) )
from theorem 40,
our = { ContActSem (a, NormOS2 (os1U[a] ) ); FinalAbActSem (a) },
ourn = NormOU2 (MergeOU (ContActSem (a, NormOS2 (os1U[a] ) ), FinalAbActSem (a) ) )
    = NormOU2 (ContActSem (a, NormOS2 (os1U[a] ) ) )
    = ContActSem (a, NormOS2 (os1U[a] ) )
from theorem 35
    = ContActSem (a, NormOS2 (os1U[a] ) )
from theorem 28
= oun.

(d): For os1s[a] with both FinalActSem (a) and FinalAbActSem (a),
ou = { os1s[a]; FinalActSem (a), FinalAbActSem (a) },
ourn = NormOU2 (MergeOU (ContActSem (a, NormOS2 (os1U[a] ) ), FinalAbActSem (a) ) )
    = NormOU2 (MergeOU (ContActSem (a, os1U[a] ), FinalActSem (a) ) )
    = NormOU2 (ContActSem (a, os1U[a] ) )
    = ContActSem (a, NormOS2 (os1U[a] ) )
from theorem 35
ou1n = ContActSem (a, NormOS2 (os1U[a] ) )
    = ContActSem (a, NormOS2 (os1U[a] ) )
from theorem 40,
our = { ContActSem (a, NormOS2 (os1U[a] ) ); FinalActSem (a) , FinalAbActSem (a) }
ourn = NormOU2 (MergeOU (ContActSem (a, NormOS2 (os1U[a] ) ); FinalActSem (a) , FinalAbActSem (a) ) )
    = NormOU2 (ContActSem (a, NormOS2 (os1U[a] ) ) )
    = ContActSem (a, NormOS2 (os1U[a] ) )
from theorem 35
    = ContActSem (a, NormOS2 (os1U[a] ) )
from theorem 28
= oun.

(e): For os2s[a] with FinalActSem (a),
ou = { FinalActSem (a); os2s[a] },
ourn = NormOU2 (MergeOU (FinalActSem (a), ContActSem (a, os2U[a] ) ) )
ou1n = FinalActSem (a),
our = { FinalActSem (a); ContActSem (a, os1U[a] ) },
ourn = NormOU2 (MergeOU (FinalActSem (a), ContActSem (a, os1U[a] ) ) )
    = FinalActSem (a)
ou1n = FinalActSem (a),
urn = NormOU2 (MergeOU (FinalActSem (a), ContActSem (a, os1U[a] ) ) )
    = NormOU2 (MergeOU (FinalActSem (a), ContActSem (a, os2U[a] ) ) )
    = NormOU2 (ContActSem (a, NormOS2 (os1U[a] ) ) )
from theorem 40
    = NormOU2 (ContActSem (a, NormOS2 (os1U[a] ) ) )
from theorem 35
    = NormOU2 (ContActSem (a, NormOS2 (os1U[a] ) ) )
from theorem 28
= oun.

(f): For os2s[a] with FinalAbActSem (a),
ou = { FinalAbActSem (a); os2s[a] },
ourn = NormOU2 (MergeOU (FinalAbActSem (a), ContActSem (a, os2U[a] ) ) )
ou1n = FinalAbActSem (a),
our = { FinalAbActSem (a); ContActSem (a, os1U[a] ) },
ourn = NormOU2 (MergeOU (FinalAbActSem (a), ContActSem (a, os1U[a] ) ) )
    = FinalAbActSem (a)
ou1n = FinalAbActSem (a),
urn = NormOU2 (MergeOU (FinalAbActSem (a), ContActSem (a, os1U[a] ) ) )
    = NormOU2 (MergeOU (FinalAbActSem (a), ContActSem (a, os2U[a] ) ) )
    = NormOU2 (ContActSem (a, os2U[a] ) )
from theorem 35
    = NormOU2 (ContActSem (a, os2U[a] ) )
from theorem 28
= oun.

(g): For os2s[a] with both FinalActSem (a) and FinalAbActSem (a),
ou = { FinalActSem (a) , FinalAbActSem (a); os2s[a] },
ourn = NormOU2 (MergeOU (FinalActSem (a), FinalAbActSem (a) ) , ContActSem (a, os2U[a] ) )
ou1n = FinalActSem (a),
our = { FinalActSem (a) ; FinalAbActSem (a); os2s[a] },
ourn = NormOU2 (MergeOU (FinalActSem (a), FinalAbActSem (a) ) , ContActSem (a, os2U[a] ) )
    = NormOU2 (MergeOU (FinalActSem (a), ContActSem (a, os2U[a] ) ) )
    = NormOU2 (FinalActSem (a) )
= oun.

These seven combinations, along with the ones that were analysed as part of the base case, thus establish the theorem for all the elements that can arise for the possibility of a ∈ PA.

For the possibilities a = ε or a = φ the arguments are identical to the base case, and do not need to be repeated here. Hence, the theorem holds for all the possible values of a in this sub-case, too, and so holds for all the sub-cases in this inductive case. Therefore, by induction the theorem holds for all values of n ≥ 1, and so holds.
The other two versions of this result that are required then follow immediately from this theorem, since set union is commutative. The first of these two forms is where NormOS2 is introduced into the second parameter to the set union, and the second of them is where it is introduced into both. These forms are expressed as the following two theorems.

**Theorem 42.**
\[
\forall os1, os2 : \text{OpSem} \mid os1 \neq \emptyset \land \text{OSNorm01}(os1) \land os2 \neq \emptyset \land \text{OSNorm01}(os2) \implies \\
\text{NormOS2}(os1 \cup os2) = \text{NormOS2}((os1 \cup \text{NormOS2}(os2))) .
\]

**Proof.**
The proof follows immediately from rewriting the right hand side as \(\text{NormOS2}((\text{NormOS2}(os2) \cup os1))\) and applying theorem 41.

**Theorem 43.**
\[
\forall os1, os2 : \text{OpSem} \mid os1 \neq \emptyset \land \text{OSNorm01}(os1) \land os2 \neq \emptyset \land \text{OSNorm01}(os2) \implies \\
\text{NormOS2}(os1 \cup os2) = \text{NormOS2}((\text{NormOS2}(os1) \cup \text{NormOS2}(os2))) .
\]

**Proof.**
The proof follows immediately from applying firstly theorem 41 and then theorem 42.

There are then useful results for \(\text{NormOU2}\) that are analogous to these, and indeed that are derived from them, and which are defined as the following theorems.

**Theorem 44.**
\[
\forall ou1, ou2 : \text{OpSUnit} \mid \text{OUNorm01}(ou1) \land \text{OUNorm01}(ou2) \land ou1.\text{TheAct} = ou2.\text{TheAct} \implies \\
\text{NormOU2}(\text{MergeOU}(\text{NormOU2}(ou1), ou2)) = \text{NormOU2}(\text{MergeOU}(ou1, ou2)).
\]

**Proof.**
The proof requires nine cases, for the different possible combinations of \(ou1.\text{NextState}\) and \(ou2.\text{NextState}\), as follows.

(i): \(ou1.\text{NextState} = \text{continues} \land ou2.\text{NextState} = \text{continues} \implies ou1.\text{Rest} \neq \emptyset \land ou2.\text{Rest} \neq \emptyset\)

\[
\implies \text{NormOU2}(\text{MergeOU}(\text{ContActSem}(ou1.\text{TheAct}, \text{NormOS2}(ou1.\text{Rest})), \\
\text{ContActSem}(ou2.\text{TheAct}, ou2.\text{Rest})))
\]

from theorem 35

\[
= \text{NormOU2}(\text{ContActSem}(ou1.\text{TheAct}, \text{NormOS2}(ou1.\text{Rest} \cup ou2.\text{Rest})))
\]

from theorem 35

\[
= \text{ContActSem}(ou1.\text{TheAct}, \text{NormOS2}(ou1.\text{Rest} \cup ou2.\text{Rest})))
\]

from theorem 41

\[
= \text{NormOU2}(\text{ContActSem}(ou1.\text{TheAct}, ou1.\text{Rest} \cup ou2.\text{Rest})))
\]

from theorem 35

\[
= \text{NormOU2}(\text{ContActSem}(ou1.\text{TheAct}, ou1.\text{Rest} \cup \{\text{EpsSem}\})))
\]

from theorem 28

(ii): \(ou1.\text{NextState} = \text{continues} \land ou2.\text{NextState} = \text{normend}\)

\[
\implies ou1.\text{Rest} \neq \emptyset \land ou2.\text{Rest} = \emptyset \land ou2 = \text{FinalActSem}(ou2.\text{TheAct})
\]

\[
\implies \text{NormOU2}(\text{MergeOU}(\text{NormOU2}(ou1), ou2))
\]

\[
= \text{NormOU2}(\text{MergeOU}(\text{ContActSem}(ou1.\text{TheAct}, \text{NormOS2}(ou1.\text{Rest})), \\
\text{FinalActSem}(ou2.\text{TheAct})))
\]

from theorem 35

\[
= \text{NormOU2}(\text{ContActSem}(ou1.\text{TheAct}, \text{NormOS2}(ou1.\text{Rest} \cup \{\text{EpsSem}\})))
\]

from theorem 35

\[
= \text{NormOU2}(\text{ContActSem}(ou1.\text{TheAct}, ou1.\text{Rest} \cup \{\text{EpsSem}\})))
\]

from theorem 40

\[
= \text{NormOU2}(\text{ContActSem}(ou1.\text{TheAct}, ou1.\text{Rest} \cup \{\text{EpsSem}\})))
\]

from theorem 35

\[
= \text{NormOU2}(\text{ContActSem}(ou1.\text{TheAct}, ou1.\text{Rest} \cup \{\text{EpsSem}\})))
\]

from theorem 28

\[
= \text{NormOU2}(\text{MergeOU}(ou1, ou2)).
\]

(iii): \(ou1.\text{NextState} = \text{continues} \land ou2.\text{NextState} = \text{abnormend}\)

\[
\implies ou1.\text{Rest} \neq \emptyset \land ou2.\text{Rest} = \emptyset \land ou2 = \text{FinalAbActSem}(ou2.\text{TheAct})
\]

\[
\implies \text{NormOU2}(\text{MergeOU}(\text{NormOU2}(ou1), ou2))
\]

\[
= \text{NormOU2}(\text{MergeOU}(\text{ContActSem}(ou1.\text{TheAct}, \text{NormOS2}(ou1.\text{Rest})), \\
\text{FinalAbActSem}(ou2.\text{TheAct})))
\]

from theorem 35

\[
= \text{NormOU2}(\text{ContActSem}(ou1.\text{TheAct}, \text{NormOS2}(ou1.\text{Rest} \cup \{\text{EpsSem}\})))
\]

from theorem 35

\[
= \text{NormOU2}(\text{ContActSem}(ou1.\text{TheAct}, ou1.\text{Rest} \cup \{\text{EpsSem}\})))
\]

from theorem 35

\[
= \text{NormOU2}(\text{ContActSem}(ou1.\text{TheAct}, ou1.\text{Rest} \cup \{\text{EpsSem}\})))
\]

from theorem 28

\[
= \text{NormOU2}(\text{MergeOU}(ou1, ou2)).
\]
(iv): \[ \text{ou}_1.\text{NextState} = \text{normend} \land \text{ou}_2.\text{NextState} = \text{continues} \]
\[ \Rightarrow \text{ou}_1.\text{Rest} = \emptyset \land \text{ou}_2.\text{Rest} \neq \emptyset \land \text{ou}_1 = \text{FinalActSem} (\text{ou}_1.\text{TheAct}) \]
\[ \Rightarrow \text{NormOU}_2 (\text{MergeOU} (\text{NormOU}_2 (\text{ou}_1), \text{ou}_2)) \]
\[ = \text{NormOU}_2 (\text{MergeOU} (\text{FinalActSem} (\text{ou}_1.\text{TheAct}), \text{ContActSem} (\text{ou}_2.\text{TheAct}, \text{ou}_2.\text{Rest})) ) \]
\[ = \text{NormOU}_2 (\text{MergeOU} (\text{ou}_1, \text{ou}_2)). \]

(v): \[ \text{ou}_1.\text{NextState} = \text{normend} \land \text{ou}_2.\text{NextState} = \text{normend} \]
\[ \Rightarrow \text{ou}_1.\text{Rest} = \emptyset \land \text{ou}_2.\text{Rest} = \emptyset \land \text{ou}_1 = \text{FinalActSem} (\text{ou}_1.\text{TheAct}) \land \text{ou}_2 = \text{FinalActSem} (\text{ou}_2.\text{TheAct}) \]
\[ \Rightarrow \text{NormOU}_2 (\text{MergeOU} (\text{NormOU}_2 (\text{ou}_1), \text{ou}_2)) \]
\[ = \text{NormOU}_2 (\text{MergeOU} (\text{FinalActSem} (\text{ou}_1.\text{TheAct}), \text{FinalActSem} (\text{ou}_2.\text{TheAct}))) \]
\[ = \text{NormOU}_2 (\text{MergeOU} (\text{ou}_1, \text{ou}_2)). \]

(vi): \[ \text{ou}_1.\text{NextState} = \text{normend} \land \text{ou}_2.\text{NextState} = \text{abnormend} \]
\[ \Rightarrow \text{ou}_1.\text{Rest} = \emptyset \land \text{ou}_2.\text{Rest} = \emptyset \land \text{ou}_1 = \text{FinalAbActSem} (\text{ou}_1.\text{TheAct}) \land \text{ou}_2 = \text{FinalAbActSem} (\text{ou}_2.\text{TheAct}) \]
\[ \Rightarrow \text{NormOU}_2 (\text{MergeOU} (\text{NormOU}_2 (\text{ou}_1), \text{ou}_2)) \]
\[ = \text{NormOU}_2 (\text{MergeOU} (\text{FinalAbActSem} (\text{ou}_1.\text{TheAct}), \text{ContActSem} (\text{ou}_2.\text{TheAct}, \text{ou}_2.\text{Rest}))) \]
\[ = \text{NormOU}_2 (\text{MergeOU} (\text{ou}_1, \text{ou}_2)). \]

(vii): \[ \text{ou}_1.\text{NextState} = \text{abnormend} \land \text{ou}_2.\text{NextState} = \text{continues} \]
\[ \Rightarrow \text{ou}_1.\text{Rest} = \emptyset \land \text{ou}_2.\text{Rest} \neq \emptyset \land \text{ou}_1 = \text{FinalAbActSem} (\text{ou}_1.\text{TheAct}) \land \text{ou}_2 = \text{FinalActSem} (\text{ou}_2.\text{TheAct}) \]
\[ \Rightarrow \text{NormOU}_2 (\text{MergeOU} (\text{NormOU}_2 (\text{ou}_1), \text{ou}_2)) \]
\[ = \text{NormOU}_2 (\text{MergeOU} (\text{FinalAbActSem} (\text{ou}_1.\text{TheAct}), \text{ContActSem} (\text{ou}_2.\text{TheAct}, \text{ou}_2.\text{Rest}))) \]
\[ = \text{NormOU}_2 (\text{MergeOU} (\text{ou}_1, \text{ou}_2)). \]

(ix): \[ \text{ou}_1.\text{NextState} = \text{abnormend} \land \text{ou}_2.\text{NextState} = \text{abnormend} \]
\[ \Rightarrow \text{ou}_1.\text{Rest} = \emptyset \land \text{ou}_2.\text{Rest} = \emptyset \land \text{ou}_1 = \text{FinalAbActSem} (\text{ou}_1.\text{TheAct}) \land \text{ou}_2 = \text{FinalAbActSem} (\text{ou}_2.\text{TheAct}) \]
\[ \Rightarrow \text{NormOU}_2 (\text{MergeOU} (\text{NormOU}_2 (\text{ou}_1), \text{ou}_2)) \]
\[ = \text{NormOU}_2 (\text{MergeOU} (\text{FinalAbActSem} (\text{ou}_1.\text{TheAct}), \text{FinalAbActSem} (\text{ou}_2.\text{TheAct}))) \]
\[ = \text{NormOU}_2 (\text{MergeOU} (\text{ou}_1, \text{ou}_2)). \]

Hence the theorem holds for all nine cases, and so holds.

Similarly, since \text{MergeOU} is commutative, it follows immediately that from this theorem that there are two further versions of this result, one where \text{NormOU}_2 is introduced into the second parameter to \text{MergeOU} and the other where it is introduced into both. These are expressed as the following theorems.

Theorem 45.
\[ \forall \text{ou}_1, \text{ou}_2 : \text{OpSUnit} \mid \text{OUNorm}_01 (\text{ou}_1) \land \text{OUNorm}_01 (\text{ou}_2) \land \text{ou}_1.\text{TheAct} = \text{ou}_2.\text{TheAct} \quad \text{NormOU}_2 (\text{MergeOU} (\text{ou}_1, \text{NormOU}_2 (\text{ou}_2) ) ) = \text{NormOU}_2 (\text{MergeOU} (\text{ou}_1, \text{ou}_2) ). \]

Proof.
The proof follows immediately from rewriting the left hand side as \text{NormOU}_2 (\text{MergeOU} (\text{NormOU}_2 (\text{ou}_2), \text{ou}_1)) and applying theorem 44.

Theorem 46.
\[ \forall \text{ou}_1, \text{ou}_2 : \text{OpSUnit} \mid \text{OUNorm}_01 (\text{ou}_1) \land \text{OUNorm}_01 (\text{ou}_2) \land \text{ou}_1.\text{TheAct} = \text{ou}_2.\text{TheAct} \quad \text{NormOU}_2 (\text{MergeOU} (\text{NormOU}_2 (\text{ou}_1), \text{NormOU}_2 (\text{ou}_2) ) ) \]
\[ = \text{NormOU}_2 (\text{MergeOU} (\text{ou}_1, \text{ou}_2) ). \]

Proof.
The proof follows immediately from applying firstly theorem 44 and then theorem 45.
The next main results that are required are to define similar sorts of distributive properties, but this time also involving calls of \( \text{SeqCompOS} \). Before these can be established, though, we again need another result for use in their proofs, to provide an analogue of theorem 35 for expressions involving \( \text{MergeOU} \). This result is expressed as the following theorem.

**Theorem 47.**

\[
\forall a : \text{PA}, \text{os1, os2 : OpSem} \mid \text{os1} \neq \emptyset \land \text{OSNorm012 (os1)} \land \text{os2} \neq \emptyset \land \text{OSNorm012 (os2)} \implies \\
\text{NormOU2 (MergeOU (Cont1ActSem (a, os1), Cont1ActSem (a, os2)))} \\
= \text{Cont1ActSem (a, NormOS2 (os1 \cup os2))}.
\]

**Proof.**

The proof has nine cases, for the different combinations of values for \( \text{os1} \) and \( \text{os2} \) that arise from the definition of the function \( \text{Cont1ActSem} \). In each case we let the three objects \( \text{ou, oum, our : OpSUnit} \) be defined as

\[
\text{oum} = \text{MergeOU (Cont1ActSem (a, os1), Cont1ActSem (a, os2))} \\
\text{ou} = \text{NormOU2 (oum) and} \\
\text{our} = \text{Cont1ActSem (a, NormOS2 (os1 \cup os2))}
\]

so that the result of the theorem can be expressed as \( \text{ou} = \text{our} \). Then the cases are as follows, where a number of them rely on the results that, from the definitions of \( \text{NormOS2} \) and \( \text{NormOU2} \),

\[
\text{NormOS2 (\{ EpsSem \})} = \{ EpsSem \}, \\
\text{NormOS2 (\{ PhiSem \})} = \{ PhiSem \}, \\
\text{NormOS2 (\{ FinalActSem (a) \})} = \{ FinalActSem (a) \}, \text{ and} \\
\text{NormOS2 (\{ FinalAbActSem (a) \})} = \{ FinalAbActSem (a) \}.
\]

(i): \( \text{os1} = \{ EpsSem \} \land \text{os2} = \{ EpsSem \} \)

\[
\implies \text{oum} = \text{MergeOU (FinalActSem (a), FinalActSem (a))} = \text{FinalActSem (a)}
\]

and

\[
\implies \text{ou} = \text{NormOU2 (FinalActSem (a))} = \text{FinalActSem (a)}
\]

and

\[
\implies \text{our} = \text{Cont1ActSem (a, NormOS2 (os1 \cup os2))} = \text{ou}
\]

(ii): \( \text{os1} = \{ EpsSem \} \land \text{os2} = \{ PhiSem \} \)

\[
\implies \text{oum} = \text{MergeOU (FinalActSem (a), FinalAbActSem (a))} = \text{FinalActSem (a)}
\]

and

\[
\implies \text{ou} = \text{NormOU2 (FinalActSem (a))} = \text{FinalActSem (a)}
\]

and

\[
\implies \text{our} = \text{Cont1ActSem (a, NormOS2 (os1 \cup \{ EpsSem \}))} = \text{Cont1ActSem (a, \{ EpsSem \})} = \text{ou}
\]

from theorem 40

(iii): \( \text{os1} = \{ EpsSem \} \land \text{os2} \neq \{ EpsSem \} \land \text{os2} \neq \{ EpsSem \} \)

\[
\implies \text{oum} = \text{MergeOU (FinalActSem (a), ContActSem (a, os2))} = \text{ContActSem (a, os2 \cup \{ EpsSem \})}
\]

and

\[
\implies \text{ou} = \text{NormOU2 (ContActSem (a, os2 \cup \{ EpsSem \}))} = \text{ContActSem (a, NormOS2 (os2 \cup \{ EpsSem \}))}
\]

and

\[
\implies \text{our} = \text{Cont1ActSem (a, NormOS2 (\{ EpsSem \} \cup os2))} = \text{ou}
\]

(iv): \( \text{os1} = \{ PhiSem \} \land \text{os2} = \{ EpsSem \} \) is symmetric with case (ii), and so the detail of the argument does not need to be repeated here.

(v): \( \text{os1} = \{ PhiSem \} \land \text{os2} = \{ PhiSem \} \)

\[
\implies \text{oum} = \text{MergeOU (FinalAbActSem (a), FinalAbActSem (a))} = \text{FinalAbActSem (a)}
\]

and

\[
\implies \text{ou} = \text{NormOU2 (FinalAbActSem (a))} = \text{FinalAbActSem (a)}
\]

and

\[
\implies \text{our} = \text{Cont1ActSem (a, NormOS2 (os2))} = \text{ou}
\]

(vi): \( \text{os1} = \{ PhiSem \} \land \text{os2} \neq \{ EpsSem \} \land \text{os2} \neq \{ EpsSem \} \)

\[
\implies \text{oum} = \text{MergeOU (FinalAbActSem (a), ContActSem (a, os2))} = \text{ContActSem (a, os2)}
\]

and

\[
\implies \text{ou} = \text{NormOU2 (ContActSem (a, os2))} = \text{Cont1ActSem (a, NormOS2 (os2))}
\]

and

\[
\implies \text{our} = \text{Cont1ActSem (a, NormOS2 (\{ PhiSem \} \cup os2))}
\]
Given this result, then there are two forms of distributive property that involve $H$.

Hence the theorem holds for all nine cases, and so holds.

\[ \text{Theorem 48.} \]
\[
\forall \text{os1, os2 : OpSem} | \text{os1} \neq \emptyset \wedge \text{OSNorm012 (os1)} \wedge \text{os2} \neq \emptyset \wedge \text{OSNorm012 (os2)} \bullet \\
\text{NormOS2 (SeqCompOS (NormOS2 (os1 \cup os2), os3))} \\
= \text{NormOS2 (SeqCompOS (os1, os3) \cup SeqCompOS (os2, os3))}. \\
\]

\text{Proof.} \\
The proof here is by induction over the maximum height of the forests os1 and os2. Throughout it, we let the following variables of type $\text{P Act}$ be defined as:
\[
\text{h1 = Heads (os1), h2 = Heads (os2), h3 = Heads (os3),} \\
h = \text{Heads (os1 \cup os2)} = \text{h1 \cup h2}, \\
\]
and as appropriate (meaning, when these objects actually exist, as in the proof of theorem 41), then for any action $a$ we let:
\[
\text{ou1 : OpSUnit be such that ou1 \in os1 \wedge GetHead (ou1) = a,} \\
\text{ou2 : OpSUnit be such that ou2 \in os2 \wedge GetHead (ou2) = a,} \\
\text{ou3 : OpSUnit be such that ou3 \in os3 \wedge GetHead (ou3) = a,} \\
\text{ou : OpSUnit be such that ou \in os \wedge GetHead (ou) = a,} \\
\text{ouc : OpSUnit be such that ouc \in osc \wedge GetHead (ouc) = a,} \\
\text{our1 : OpSUnit be such that our1 \in osr1 \wedge GetHead (our1) = a,} \\
\text{our2 : OpSUnit be such that our2 \in osr2 \wedge GetHead (our2) = a,} \\
\text{our : OpSUnit be such that our \in osr \wedge GetHead (our) = a.} \\
\]

In principle, since each of os1 and os2 are in second normal form, the argument can be structured in terms of analysing the elements of them and of the result sets that have a particular head $a$, and in each of the base and inductive cases this gives rise essentially to three sub-cases, depending on whether $a$ is in either $h1$ or $h2$, or both, since the case where it is in neither can be ignored. Within each of these sub-cases there are then three further possibilities for the value of $a$, namely that it may be in PA or may be either of the two constants $\varepsilon$ or $\Phi$, although (again since each of os1 and os2 are in second normal form) the case of $a = \varepsilon$ can be ignored where $\#$ os1 > 1 or $\#$ os2 > 1. Each of these possibilities then corresponds to different constructions of the elements ou1 and ou2, and hence to different combinations of these possibilities that need to be analysed.

In practice, though, particularly where os1 or os2 are singleton sets, it is often simpler to structure the argument in terms of computing the results osc and osr directly, to show that they are equal. Where $\#$ os1 > 1 or $\#$ os2 > 1, though, the argument will usually be structured in terms of showing that the elements ouc and our are equal for any arbitrary head $a$, so that the required result for osc and osr in these cases then follows from theorem 33. As in the proof of theorem 47, a number of the cases rely on the results that
\[
\text{NormOS2 (\{EpsSem\}) = \{EpsSem\},} \\
\text{NormOS2 (\{PhiSem\}) = \{PhiSem\},} \\
\text{NormOS2 (\{FinalActSem (a)\}) = \{FinalActSem (a)\}, and} \\
\text{NormOS2 (\{FinalAbActSem (a)\}) = \{FinalAbActSem (a)\}.} \\
\]
Base case.
The base case is that $\text{HeightOS (os1)} = 1 \land \text{HeightOS (os2)} = 1$, so that for any arbitrary element $a$ the possible constructions of $ou1$ and $ou2$ are given by theorem 5. For the case where $\# os1 = 1 \land \# os2 = 1$ the combinations of possible values are as follows, where ones that are symmetric with previously analysed combinations are implicitly ignored.

(i): $os1 = \{ \text{EpsSem} \} \land os2 = \{ \text{EpsSem} \}$
\[ \Rightarrow os = \text{NormOS2 (\{ \text{EpsSem} \})} = \{ \text{EpsSem} \} \]
\[ \Rightarrow osc = \text{NormOS2 (os3)} \]
\[ \text{and} \Rightarrow osr1 = os3 \land osr2 = os3 \]
\[ \Rightarrow osr = \text{NormOS2 (os3} \cup \text{os3)} = \text{osc} \]

(ii): $os1 = \{ \text{EpsSem} \} \land os2 = \{ \text{PhiSem} \}$
\[ \Rightarrow os = \text{NormOS2 (\{ \text{EpsSem, PhiSem} \})} = \text{NormOS2 (\{ \text{EpsSem} \})} \quad \text{from theorem 40} \]
\[ = \{ \text{EpsSem} \} \]
\[ \Rightarrow osc = \text{NormOS2 (os3)} \]
\[ \text{and} \Rightarrow osr1 = os3 \land osr2 = \{ \text{PhiSem} \} \]
\[ \Rightarrow osr = \text{NormOS2 (os3} \cup \{ \text{PhiSem} \}) = \text{NormOS2 (os3)} \quad \text{from theorem 40} \]
\[ = \text{osc} \].

(iii): $os1 = \{ \text{EpsSem} \} \land os2 = \{ \text{FinalActSem (a)} \}$
\[ \Rightarrow os = \text{NormOS2 (\{ \text{EpsSem, FinalActSem (a)} \})} \]
\[ \Rightarrow osc = \text{NormOS2 (os3} \cup \{ \text{Cont1ActSem (a, os3)} \}) \]
\[ \text{and} \Rightarrow osr1 = os3 \land osr2 = \{ \text{Cont1ActSem (a, os3)} \} \]
\[ \Rightarrow osr = \text{NormOS2 (os3} \cup \{ \text{Cont1ActSem (a, os3)} \}) = \text{osc} \].

(iv): $os1 = \{ \text{EpsSem} \} \land os2 = \{ \text{FinalAbActSem (a)} \}$
\[ \Rightarrow os = \text{NormOS2 (\{ \text{EpsSem, FinalAbActSem (a)} \})} \]
\[ \Rightarrow osc = \text{NormOS2 (os3} \cup \{ \text{FinalAbActSem (a)} \}) \]
\[ \text{and} \Rightarrow osr1 = os3 \land osr2 = \{ \text{FinalAbActSem (a)} \} \]
\[ \Rightarrow osr = \text{NormOS2 (os3} \cup \{ \text{FinalAbActSem (a)} \}) = \text{osc} \].

(v): $os1 = \{ \text{PhiSem} \} \land os2 = \{ \text{PhiSem} \}$
\[ \Rightarrow os = \text{NormOS2 (\{ \text{PhiSem} \})} = \{ \text{PhiSem} \} \]
\[ \Rightarrow osc = \text{NormOS2 (\{ \text{PhiSem} \})} \]
\[ \text{and} \Rightarrow osr1 = \{ \text{PhiSem} \} \land osr2 = \{ \text{PhiSem} \} \]
\[ \Rightarrow osr = \text{NormOS2 (\{ \text{PhiSem} \} \cup \{ \text{Cont1ActSem (a, os3)} \})} = \text{osc} \].

(vi): $os1 = \{ \text{PhiSem} \} \land os2 = \{ \text{FinalActSem (a)} \}$
\[ \Rightarrow os = \text{NormOS2 (\{ \text{PhiSem, FinalActSem (a)} \})} = \text{NormOS2 (\{ \text{FinalActSem (a)} \})} \quad \text{from theorem 40} \]
\[ = \{ \text{FinalActSem (a)} \} \]
\[ \Rightarrow osc = \text{NormOS2 (\{ \text{Cont1ActSem (a, os3)} \})} \]
\[ \text{and} \Rightarrow osr1 = \{ \text{PhiSem} \} \land osr2 = \{ \text{Cont1ActSem (a, os3)} \} \]
\[ \Rightarrow osr = \text{NormOS2 (\{ \text{PhiSem} \} \cup \{ \text{Cont1ActSem (a, os3)} \})} = \text{osc} \].

(vii): $os1 = \{ \text{PhiSem} \} \land os2 = \{ \text{FinalAbActSem (a)} \}$
\[ \Rightarrow os = \text{NormOS2 (\{ \text{PhiSem, FinalAbActSem (a)} \})} = \text{NormOS2 (\{ \text{FinalAbActSem (a)} \})} \quad \text{from theorem 40} \]
\[ = \{ \text{FinalAbActSem (a)} \} \]
\[ \Rightarrow osc = \text{NormOS2 (\{ \text{FinalAbActSem (a)} \})} \]
\[ \text{and} \Rightarrow osr1 = \{ \text{PhiSem} \} \land osr2 = \{ \text{FinalAbActSem (a)} \} \]
\[ \Rightarrow osr = \text{NormOS2 (\{ \text{PhiSem} \} \cup \{ \text{FinalAbActSem (a)} \})} = \text{osc} \].

(viii): $os1 = \{ \text{FinalActSem (a)} \} \land os2 = \{ \text{FinalActSem (a)} \}$
\[ \Rightarrow os = \text{NormOS2 (\{ \text{FinalActSem (a)} \})} \]
\[ = \{ \text{FinalActSem (a)} \} \]
\[ \Rightarrow osc = \text{NormOS2 (\{ \text{Cont1ActSem (a, os3)} \})} \]
\[ \text{and} \Rightarrow osr1 = \{ \text{Cont1ActSem (a, os3)} \} \land osr2 = \{ \text{Cont1ActSem (a, os3)} \} \]
\[ \Rightarrow osr = \text{NormOS2 (\{ \text{Cont1ActSem (a, os3)} \})} = \text{osc} \].
(ix): \[ \text{os} = \{ \text{FinalActSem (a)} \} \land \text{os} = \{ \text{FinalAbActSem (a)} \} \Rightarrow \text{os} = \text{NormOS2} (\{ \text{FinalActSem (a)}, \text{FinalAbActSem (a)} \}) = \{ \text{FinalActSem (a)} \} \]

\[
\Rightarrow \text{osc} = \text{NormOS2} (\{ \text{Cont1ActSem (a, os3)} \})
\]

and \[
\Rightarrow \text{osr} = \{ \text{Cont1ActSem (a, os3)} \} \land \text{osr} = \{ \text{FinalAbActSem (a)} \}
\]

\[
\Rightarrow \text{osr} = \text{NormOS2} (\{ \text{Cont1ActSem (a, os3)}, \text{FinalAbActSem (a)} \})
\]

There are then three possibilities for this, depending on the value of \(\text{os3}\), as follows.

(ix)(a): \[
\text{os3} = \{ \text{EpsSem} \} \Rightarrow \text{Cont1ActSem (a, os3)} = \text{FinalActSem (a)} \land \text{osr} = \text{NormOS2} (\{ \text{FinalActSem (a)} \}) = \text{osc}.
\]

(ix)(b): \[
\text{os3} = \{ \text{PhiSem} \} \Rightarrow \text{Cont1ActSem (a, os3)} = \text{FinalAbActSem (a)} \land \text{osr} = \text{NormOS2} (\{ \text{FinalAbActSem (a)} \}) = \text{osc}.
\]

(ix)(c): Otherwise, \[
\text{Cont1ActSem (a, os3)} = \text{ContActSem (a, os3)} \land \text{osr} = \text{NormOS2} (\{ \text{ContActSem (a, os3)} \}) = \text{osc}.
\]

(x): \[
\text{os} = \{ \text{FinalAbActSem (a)} \} \land \text{os} = \{ \text{FinalAbActSem (a)} \} \Rightarrow \text{os} = \text{NormOS2} (\{ \text{FinalAbActSem (a)} \}) = \{ \text{FinalAbActSem (a)} \} \Rightarrow \text{osc} = \text{NormOS2} (\{ \text{FinalAbActSem (a)} \}) = \text{osc}.
\]

Hence, the theorem holds for all the combinations where \(# \text{os1} = 1 \land # \text{os2} = 1\). For the case \(# \text{os1} > 1 \land # \text{os2} = 1\) the analysis considers the elements of \(\text{os1}\) with an arbitrary head \(a\), where the possibility of \(a = \phi\) can not occur. Many of the possible combinations duplicate cases (i) to (x) above, and so are not repeated here, and so the only additional cases are the following, which are numbered to continue the sequence above. The common feature of these is that they need to consider two elements of \(\text{os1}\), namely \(\text{ou1}\) as defined above and \(\text{EpsSem}\), although if \(a \notin h3\) then \(\text{SeqCompOU (EpsSem, os3)}\) does not affect \(\text{ouc}\), and so the argument is the same as for the appropriate one of the cases above.

If \(a \in h3\), though, then \(\text{SeqCompOU (EpsSem, os3)}\) does contain an element that contributes to \(\text{ouc}\) and \(\text{our}\), and if we denote this element by \(\text{our1e} : \text{OpSUnit}\) then we have \(\text{our1e} = \text{GetUnit (a, SeqCompOU (EpsSem, os3)) = ou3}\). Hence, in constructing \(\text{osc}\) the application of \(\text{NormOS2}\) invokes \(\text{MergeOU}\) to combine this element \(\text{our1e}\) with \(\text{ou}\), to give the element \(\text{ouc}\). In the construction of \(\text{osr}\) the two elements \(\text{our1}\) and \(\text{our1e}\) will remain separate, but will form a pair that will then be combined by \(\text{MergeOU}\) when it is invoked by \(\text{NormOS2}\) in the construction of \(\text{osr}\), and we will denote this pair by \(\text{our1p}\).

The analysis of these cases, and also of many in the inductive case, relies heavily on the properties of \(\text{MergeOU}\) that are expressed in theorems 19 to 21, and applications of these are not indicated explicitly in what follows. The analysis of some cases also relies on the fact that, for any object \(\text{ou'} : \text{OpSUnit}\) such that \(\text{GetHead (ou')} = a\),

\[\text{MergeOU (ou', FinalAbActSem (a)) = ou'}\]

The cases themselves are as follows.

(xi): \[
\text{ou1} = \text{FinalActSem (a)} \land \text{EpsSem} \in \text{os1} \land \text{ou2} = \text{FinalActSem (a)} \Rightarrow \text{ou} = \text{FinalActSem (a)}
\]

\[
a \in h3 \Rightarrow \text{ouc} = \text{NormOU2 (MergeOU (Cont1ActSem (a, os3), ou3))}
\]

and \[
\Rightarrow \text{our1} = \text{Cont1ActSem (a, os3)} \land \text{our1e} = \text{ou3} \land \text{our2} = \text{Cont1ActSem (a, os3)}
\]

\[
\Rightarrow \text{our1p} = \{ \text{Cont1ActSem (a, os3), ou3} \}
\]

\[
\Rightarrow \text{our} = \text{NormOU2 (MergeOU (MergeOU (Cont1ActSem (a, os3), ou3), Cont1ActSem (a, os3)))}
\]

\[
= \text{NormOU2 (MergeOU (Cont1ActSem (a, os3), ou3))}
\]

\[
= \text{ouc}.
\]

(xii): \[
\text{ou1} = \text{FinalActSem (a)} \land \text{EpsSem} \in \text{os1} \land \text{ou2} = \text{FinalAbActSem (a)} \Rightarrow \text{ou} = \text{FinalActSem (a)}
\]

\[
a \in h3 \Rightarrow \text{ouc} = \text{NormOU2 (MergeOU (Cont1ActSem (a, os3), ou3))}
\]

and \[
\Rightarrow \text{our1} = \text{Cont1ActSem (a, os3)} \land \text{our1e} = \text{ou3} \land \text{our2} = \text{FinalAbActSem (a)}
\]

\[
\Rightarrow \text{our1p} = \{ \text{Cont1ActSem (a, os3), ou3} \}
\]

\[
\Rightarrow \text{our} = \text{NormOU2 (MergeOU (MergeOU (Cont1ActSem (a, os3), ou3), FinalAbActSem (a)))}
\]

\[
= \text{NormOU2 (MergeOU (Cont1ActSem (a, os3), ou3))}
\]

\[
= \text{ouc}.
\]

(xiii): \[
\text{ou1} = \text{FinalAbActSem (a)} \land \text{EpsSem} \in \text{os1} \land \text{ou2} = \text{FinalActSem (a)} \Rightarrow \text{ou} = \text{FinalActSem (a)}
\]

\[
a \in h3 \Rightarrow \text{ouc} = \text{NormOU2 (MergeOU (FinalActSem (a), ou3))}
\]

and \[
\Rightarrow \text{our1} = \text{FinalAbActSem (a)} \land \text{our1e} = \text{ou3} \land \text{our2} = \text{FinalActSem (a)}
\]
For the combinations where \( \# \text{os}_1 = 1 \land \# \text{os}_2 > 1 \) the arguments are symmetric, and so are not repeated, while for the combinations where \( \# \text{os}_1 > 1 \land \# \text{os}_2 > 1 \) there are three more additional cases to analyse, plus a symmetric case that does not need to be analysed separately. These cases arise from the similar situation where \( \text{EpsSem} \in \text{os}_2 \land a \in h_3 \), so that there is also a contribution to osc and our from an element that we will denote by our2e: \text{OpSUnit} , and which is given by

\[
\text{our2e} = \text{GetUnit}(a, \text{SeqCompOU}(\text{EpsSem}, \text{os}_3)) = \text{ou}_3.
\]

Thus, in the construction of osc the application of \text{NormOS2} invokes \text{MergeOU} to combine this element our1e with our to give the element ouc, while in the construction of osr2 the two elements our2 and our2e form a pair that will be denoted our2p, and that are then combined by \text{MergeOU} when it is invoked by \text{NormOS2} in the construction of osr. Thus, the analysis of these cases is as follows.

(xvii): \( \text{ou}_1 = \text{FinalAbActSem}(a) \land \text{EpsSem} \in \text{os}_1 \land \text{ou}_2 = \text{FinalAbActSem}(a) \land \text{EpsSem} \in \text{os}_2 \)
\[
\Rightarrow \text{ou} = \text{FinalAbActSem}(a)
\]
\[
a \in h_3 \Rightarrow \text{ou} = \text{NormOU2}(\text{MergeOU}(\text{Cont1ActSem}(a, \text{os}_3), \text{ou}_3), \text{ou}_3)
\]
\[
= \text{NormOU2}(\text{MergeOU}(\text{Cont1ActSem}(a, \text{os}_3), \text{ou}_3), \text{ou}_3)
\]
\[
\Rightarrow \text{our1} = \text{Cont1ActSem}(a, \text{os}_3) \land \text{our1e} = \text{ou}_3 \land \text{our2} = \text{FinalAbActSem}(a) \land \text{ou2e} = \text{ou}_3
\]
\[
\Rightarrow \text{our1p} = \{ \text{Cont1ActSem}(a, \text{os}_3), \text{ou}_3 \} \land \text{our2p} = \{ \text{FinalAbActSem}(a, \text{os}_3), \text{ou}_3 \}
\]
\[
= \text{NormOU2}(\text{MergeOU}(\text{Cont1ActSem}(a, \text{os}_3), \text{ou}_3), \text{ou}_3)
\]
\[
= \text{NormOU2}(\text{MergeOU}(\text{Cont1ActSem}(a, \text{os}_3), \text{ou}_3), \text{ou}_3)
\]
\[
= \text{ouc}.
\]

Hence, the theorem holds for all of these different combinations, and so holds for the base case.

**Inductive Case.**

The inductive case is that, for some natural number \( n \), the maximum of \( \text{HeightOS}(\text{os}_1) \) and \( \text{HeightOS}(\text{os}_2) = n \), so that either \( \text{HeightOS}(\text{os}_1) > 1 \) or \( \text{HeightOS}(\text{os}_2) > 1 \) or both. The induction hypothesis is then that the theorem holds
for all os1 and os2 such that \( \text{HeightOS} (os1) < n \) and \( \text{HeightOS} (os2) < n \), and the induction step is to show from this that therefore it also holds for the inductive case.

For any arbitrary head element \( a \) this case gives rise to one new possible construction for each of \( ou1 \) and \( ou2 \), namely

\[
\exists \, os1x : \text{OpSem} \ | \ \text{IsActive} (os1x) \wedge \text{OSNorm0} (os1x) \wedge \text{OSNorm1} (os1x) \wedge \text{OSNorm2} (os1x)
\]

\[
\Rightarrow \, ou1 = \text{ContActSem} (a, os1x)
\]

\[
\exists \, os2x : \text{OpSem} \ | \ \text{IsActive} (os2x) \wedge \text{OSNorm0} (os2x) \wedge \text{OSNorm1} (os2x) \wedge \text{OSNorm2} (os2x)
\]

\[
\Rightarrow \, ou2 = \text{ContActSem} (a, os2x)
\]

There are then various combinations involving one or the other of these that need to be analysed, where the combinations are either with each other or with the different constructions considered in the base case. There is obviously no need to repeat the analysis of combinations that appear in the base case, and so the ones that need to be considered here are as follows, where elements \( ou1e \) and \( ou2e \), and the pairs \( ou1p \) and \( ou2p \), are as defined in the base case. Also as in the base case, in any combination that includes \( \text{EpsSem} \), where \( a \not\in h3 \) then \( \text{SeqCompOU} (\text{EpsSem}, os3) \) does not affect \( ouc \), and so the analysis is the same as for the equivalent combination that does not involve \( \text{EpsSem} \), and is not repeated here.

Firstly, there are six combinations corresponding to the sub-case where \( a \in h1 \land a \not\in h2 \), arising from the possibilities that \( \text{EpsSem} \) may be in either or both of \( os1 \) or \( os2 \), or that \( os2 \) may consist just of the element \( \Phi\text{Sem} \).

(i): \( ou1 = \text{ContActSem} (a, os1x) \wedge \text{EpsSem} \not\in os1 \wedge \text{EpsSem} \not\in os2 \wedge ou2 \) does not exist

\[
\Rightarrow \, ou = \text{ContActSem} (a, os1x)
\]

\[
\Rightarrow \, ouc = \text{NormOU2} (\text{Cont1ActSem} (a, \text{SeqCompOS} (os1x, os3)))
\]

and

\[
\Rightarrow \, ou1 = \text{Cont1ActSem} (a, \text{SeqCompOS} (os1x, os3)) \wedge ou2 \) does not exist

\[
\Rightarrow \, ou = \text{NormOU2} (\text{Cont1ActSem} (a, \text{SeqCompOS} (os1x, os3)))
\]

\[= ouc.\]

(ii): \( ou1 = \text{ContActSem} (a, os1x) \wedge \text{EpsSem} \in os1 \wedge \text{EpsSem} \not\in os2 \wedge ou2 \) does not exist

\[
\Rightarrow \, ou = \text{ContActSem} (a, os1x) \wedge ou1e = ou3
\]

\[a \in h3 \Rightarrow \, ouc = \text{NormOU2} (\text{MergeOU} (\text{Cont1ActSem} (a, \text{SeqCompOS} (os1x, os3)), ou3))\]

and

\[
\Rightarrow \, ou1 = \text{Cont1ActSem} (a, \text{SeqCompOS} (os1x, os3)) \wedge ou1e = ou3 \wedge ou2 \) does not exist

\[
\Rightarrow \, ou1p = \{ \text{Cont1ActSem} (a, \text{SeqCompOS} (os1x, os3)), ou3 \}
\]

\[
\Rightarrow \, ou = \text{NormOU2} (\text{MergeOU} (\text{Cont1ActSem} (a, \text{SeqCompOS} (os1x, os3)), ou3))
\]

\[= ouc.\]

(iii): \( ou1 = \text{ContActSem} (a, os1x) \wedge \text{EpsSem} \not\in os1 \wedge \text{EpsSem} \in os2 \wedge ou2 \) does not exist

\[
\Rightarrow \, ou = \text{ContActSem} (a, os1x) \wedge ou2e = ou3
\]

\[a \in h3 \Rightarrow \, ouc = \text{NormOU2} (\text{MergeOU} (\text{Cont1ActSem} (a, \text{SeqCompOS} (os1x, os3)), ou3))\]

and

\[
\Rightarrow \, ou1 = \text{Cont1ActSem} (a, \text{SeqCompOS} (os1x, os3)) \wedge ou2e = ou3 \wedge ou2 \) does not exist

\[
\Rightarrow \, ou = \text{NormOU2} (\text{MergeOU} (\text{Cont1ActSem} (a, \text{SeqCompOS} (os1x, os3)), ou3))
\]

\[= ouc.\]

(iv): \( ou1 = \text{ContActSem} (a, os1x) \wedge \text{EpsSem} \in os1 \wedge \text{EpsSem} \in os2 \wedge ou2 \) does not exist

\[
\Rightarrow \, ou = \text{ContActSem} (a, os1x) \wedge ou1e = ou3 \wedge ou2e = ou3
\]

\[a \in h3 \Rightarrow \, ouc = \text{NormOU2} (\text{MergeOU} (\text{Cont1ActSem} (a, \text{SeqCompOS} (os1x, os3)), ou3))\]

and

\[
\Rightarrow \, ou1 = \text{Cont1ActSem} (a, \text{SeqCompOS} (os1x, os3)) \wedge ou1e = ou3 \wedge ou2 \) does not exist

\[
\Rightarrow \, ou1p = \{ \text{Cont1ActSem} (a, \text{SeqCompOS} (os1x, os3)), ou3 \}
\]

\[
\Rightarrow \, ou = \text{NormOU2} (\text{MergeOU} (\text{Cont1ActSem} (a, \text{SeqCompOS} (os1x, os3)), ou3))
\]

\[= ouc.\]

(v): \( ou1 = \text{ContActSem} (a, os1x) \wedge \text{EpsSem} \not\in os1 \wedge os2 = \{ \Phi\text{Sem} \} \Rightarrow ou2 \) does not exist

\[
\Rightarrow \, ou = \text{ContActSem} (a, os1x)
\]

\[
\Rightarrow \, ouc = \text{NormOU2} (\text{Cont1ActSem} (a, \text{SeqCompOS} (os1x, os3)))
\]

and

\[
\Rightarrow \, ou1 = \text{Cont1ActSem} (a, \text{SeqCompOS} (os1x, os3)) \wedge ou2 \) does not exist

\[
\Rightarrow \, ou = \text{NormOU2} (\text{Cont1ActSem} (a, \text{SeqCompOS} (os1x, os3)))
\]

\[= ouc.\]

(vi): \( ou1 = \text{ContActSem} (a, os1x) \wedge \text{EpsSem} \in os1 \wedge os2 = \{ \Phi\text{Sem} \} \Rightarrow ou2 \) does not exist

\[
\Rightarrow \, ou = \text{ContActSem} (a, os1x) \wedge ou1e = ou3
\]

\[a \in h3 \Rightarrow \, ouc = \text{NormOU2} (\text{MergeOU} (\text{Cont1ActSem} (a, \text{SeqCompOS} (os1x, os3)), ou3))\]

and

\[
\Rightarrow \, ou1 = \text{Cont1ActSem} (a, \text{SeqCompOS} (os1x, os3)) \wedge ou1e = ou3 \wedge ou2 \) does not exist

\[
\Rightarrow \, ou1p = \{ \text{Cont1ActSem} (a, \text{SeqCompOS} (os1x, os3)), ou3 \}
\]

\[
\Rightarrow \, ou = \text{NormOU2} (\text{MergeOU} (\text{Cont1ActSem} (a, \text{SeqCompOS} (os1x, os3)), ou3))
\]

\[= ouc.\]
Secondly, there are six more combinations corresponding to the sub-case where \( a \notin h_1 \land a \in h_2 \), but since these are symmetric with cases (i) to (vi) above the detail of the analyses for them does not need to be repeated here.

Thirdly, there is the sub-case where \( a \in h_1 \land a \in h_2 \), which gives rise to three sets of combinations. The first of these sets is for \( \text{HeightOS} (os_1) > 1 \land \text{HeightOS} (os_2) = 1 \), where there are two different possible values for \( ou_2 \), namely \( \text{FinalActSem} (a) \) or \( \text{FinalAbActSem} (a) \), and the four possible combinations for each of these are produced by whether or not \( \text{EpsSem} \) is in either of \( os_1 \) or \( os_2 \). The second set is for the case \( \text{HeightOS} (os_1) = 1 \land \text{HeightOS} (os_2) > 1 \), but since this is symmetrical with the first set it does not need to be analysed here in detail. The third set is for the case where \( \text{HeightOS} (os_1) > 1 \land \text{HeightOS} (os_2) > 1 \), and this then consists of four cases, again depending on whether or not \( \text{EpsSem} \) is in either or both of \( os_1 \) or \( os_2 \).

For some of these combinations that need to be considered here, the analysis relies on the fact that

\[
 a \in h_3 \Rightarrow \text{IsActive} (os_3) \\
\Rightarrow \forall os' : \text{OpSem} \mid \text{OSNorm0} (os') \land \text{OSNorm1} (os') \land os' \neq \{ \text{PhiSem} \} \\
\text{IsActive} (\text{SeqCompOS} (os', os_3)) \\
\Rightarrow \text{Cont1ActSem} (a, \text{SeqCompOS} (os', os_3)) = \text{ContActSem} (a, \text{SeqCompOS} (os', os_3))
\]

from theorem 14

These cases are therefore as follows, where blank lines are used to separate the groups that each have the same values for \( ou_1 \) and \( ou_2 \).

(vii):

\[
\begin{align*}
(\text{vii}): & \quad \text{ou}_1 = \text{ContActSem} (a, os_1x) \land \text{ou}_2 = \text{FinalActSem} (a) \land \text{EpsSem} \notin os_1 \land \text{EpsSem} \notin os_2 \\
& \Rightarrow \text{ou} = \text{NormOU2} (\text{MergeOU} (\text{ContActSem} (a, os_1x), \text{FinalActSem} (a))) \\
& = \text{NormOU2} (\text{ContActSem} (a, os_1x \cup \{ \text{EpsSem} \})) \\
& = \text{ContActSem} (a, \text{NormOS2} (os_1x \cup \{ \text{EpsSem} \})) \\
& \Rightarrow \text{our}_1 = \text{Cont1ActSem} (a, \text{SeqCompOS} (\text{NormOS2} (os_1x \cup \{ \text{EpsSem} \}), os_3)) \\
& = \text{NormOU2} (\text{ContActSem} (a, \text{SeqCompOS} (\text{NormOS2} (os_1x \cup \{ \text{EpsSem} \}), os_3))) \\
& = \text{Cont1ActSem} (a, \text{NormOS2} (\text{SeqCompOS} (os_1x, os_3) \cup \text{SeqCompOS} (\{ \text{EpsSem} \}, os_3))) \\
& \text{from the induction hypothesis}
\end{align*}
\]

and

\[
\begin{align*}
\text{ou}_1 & = \text{ContActSem} (a, \text{NormOS2} (os_1x, os_3) \cup os_3)) \\
\Rightarrow \text{our}_1 & = \text{Cont1ActSem} (a, \text{SeqCompOS} (\text{NormOS2} (os_1x, os_3) \cup \text{SeqCompOS} (\{ \text{EpsSem} \}, os_3)), os_3) \\
& = \text{NormOU2} (\text{MergeOU} (\text{Cont1ActSem} (a, \text{SeqCompOS} (\text{NormOS2} (os_1x, os_3) \cup \text{SeqCompOS} (\{ \text{EpsSem} \}, os_3)), os_3))) \\
& = \text{NormOU2} (\text{MergeOU} (\text{Cont1ActSem} (a, \text{SeqCompOS} (\text{NormOS2} (os_1x, os_3) \cup \text{SeqCompOS} (\{ \text{EpsSem} \}, os_3)), os_3), os_3)) \\
& = \text{Cont1ActSem} (a, \text{NormOS2} (\text{SeqCompOS} (os_1x, os_3) \cup \text{SeqCompOS} (\{ \text{EpsSem} \}, os_3))) \\
& \text{from the induction hypothesis}
\end{align*}
\]

as in case (vii) above

\[
\begin{align*}
\text{viii}): & \quad \text{ou}_1 = \text{ContActSem} (a, os_1x) \land \text{ou}_2 = \text{FinalActSem} (a) \land \text{EpsSem} \in os_1 \land \text{EpsSem} \notin os_2 \\
& \Rightarrow \text{ou} = \text{ContActSem} (a, \text{NormOS2} (os_1x \cup \{ \text{EpsSem} \})) \\
& = \text{ContActSem} (a, \text{NormOS2} (os_1x \cup \{ \text{EpsSem} \})) \\
& \Rightarrow \text{our}_1 = \text{Cont1ActSem} (a, \text{SeqCompOS} (\text{NormOS2} (os_1x \cup \{ \text{EpsSem} \}), os_3)) \\
& = \text{NormOU2} (\text{MergeOU} (\text{Cont1ActSem} (a, \text{SeqCompOS} (\text{NormOS2} (os_1x \cup \{ \text{EpsSem} \}), os_3)), os_3)) \\
& = \text{NormOU2} (\text{MergeOU} (\text{Cont1ActSem} (a, \text{SeqCompOS} (\text{NormOS2} (os_1x \cup \{ \text{EpsSem} \}), os_3)), os_3), os_3) \\
& = \text{Cont1ActSem} (a, \text{NormOS2} (\text{SeqCompOS} (os_1x, os_3) \cup \text{SeqCompOS} (\{ \text{EpsSem} \}, os_3))) \\
& \text{from the induction hypothesis}
\end{align*}
\]

and

\[
\begin{align*}
\text{ou}_1 & = \text{ContActSem} (a, \text{SeqCompOS} (os_1x, os_3)) \\
\Rightarrow \text{our}_1 & = \text{Cont1ActSem} (a, \text{SeqCompOS} (os_1x, os_3)) \\
& = \text{Cont1ActSem} (a, \text{SeqCompOS} (os_1x, os_3) \cup \text{SeqCompOS} (\{ \text{EpsSem} \}, os_3))) \\
& = \text{NormOU2} (\text{MergeOU} (\text{Cont1ActSem} (a, \text{SeqCompOS} (os_1x, os_3) \cup \text{SeqCompOS} (\{ \text{EpsSem} \}, os_3))) \\
& = \text{NormOU2} (\text{MergeOU} (\text{Cont1ActSem} (a, \text{SeqCompOS} (os_1x, os_3) \cup \text{SeqCompOS} (\{ \text{EpsSem} \}, os_3)), os_3)) \\
& = \text{Cont1ActSem} (a, \text{SeqCompOS} (os_1x, os_3)) \\
& \text{from the induction hypothesis}
\end{align*}
\]
= NormOU2 (MergeOU (Cont1ActSem (a, NormOS2 (SeqCompOS (os1x, os3) \cup os3) ), ou3) )

from theorem 47

= ouc.

(ix): ou1 = ContActSem (a, os1x) \land ou2 = FinalActSem (a) \land EpsSem \notin os1 \land EpsSem \in os2
⇒ ou = ContActSem (a, NormOS2 (os1x \cup \{ EpsSem \} ) ) \land ou2e = ou3
a \in h3 ⇒
ouc = NormOU2 (MergeOU (Cont1ActSem (a, SeqCompOS (NormOS2 (os1x \cup \{ EpsSem \} ), os3) ),
ou3) )

= NormOU2 (MergeOU (Cont1ActSem (a, NormOS2 (SeqCompOS (os1x, os3) ), os3) ),
ou3) )

by the working in case (viii)

and ⇒ our1 = Cont1ActSem (a, SeqCompOS (os1x, os3) ) \land our2e = ou3 \land our2 = Cont1ActSem (a, os3)
⇒ our2p = \{ Cont1ActSem (a, os3), ou3 \}
⇒ our = NormOU2 (MergeOU (Cont1ActSem (a, SeqCompOS (os1x, os3) ),
MergeOU (Cont1ActSem (a, os3), ou3) )
= NormOU2 (MergeOU (MergeOU (Cont1ActSem (a, SeqCompOS (os1x, os3) ),
Cont1ActSem (a, os3) ), ou3) )
= NormOU2 (MergeOU (Cont1ActSem (a, NormOS2 (SeqCompOS (os1x, os3) ), os3) ),
ou3) )
from theorem 47

= ouc.

(x): ou1 = ContActSem (a, os1x) \land ou2 = FinalActSem (a)
⇒ ou = ContActSem (a, os1x \cup \{ EpsSem \} ) \land our1e = ou3 \land our2e = ou3
a \in h3 ⇒
ouc = NormOU2 (MergeOU (MergeOU (Cont1ActSem (a, SeqCompOS (os1x \cup \{ EpsSem \} ), os3) ),
ou3) )

= NormOU2 (MergeOU (Cont1ActSem (a, SeqCompOS (os1x, os3) ),
ou3) )
as in case (ix)

and ⇒ our1 = Cont1ActSem (a, SeqCompOS (os1x, os3) ) \land our1e = ou3 \land our2 = Cont1ActSem (a, os3)
⇒ our2p = \{ Cont1ActSem (a, os3), ou3 \}
⇒ our = NormOU2 (MergeOU (MergeOU (Cont1ActSem (a, SeqCompOS (os1x, os3) ),
Cont1ActSem (a, os3) ),
ou3) )
= NormOU2 (MergeOU (MergeOU (Cont1ActSem (a, NormOS2 (SeqCompOS (os1x, os3) ), os3) ),
ou3) )
= NormOU2 (MergeOU (Cont1ActSem (a, NormOS2 (SeqCompOS (os1x, os3) ), os3) ),
ou3) )
from theorem 47

= ouc.

(xi): ou1 = ContActSem (a, os1x) \land ou2 = FinalAbActSem (a)
⇒ ou = ContActSem (a, os1x \cup \{ EpsSem \} ) \land our1e = ou3 \land our2e = ou3
a \in h3 ⇒
ouc = NormOU2 (MergeOU (Cont1ActSem (a, SeqCompOS (os1x, os3) ),
ou3) )

= NormOU2 (ContActSem (a, os1x) )

from theorem 28

and ⇒ our1 = Cont1ActSem (a, SeqCompOS (os1x, os3) ) \land our2 = FinalAbActSem (a)
⇒ our = NormOU2 (MergeOU (Cont1ActSem (a, SeqCompOS (os1x, os3) ),
FinalAbActSem (a) )
= NormOU2 (Cont1ActSem (a, SeqCompOS (os1x, os3) ))
= ouc.

(xii): ou1 = ContActSem (a, os1x) \land ou2 = FinalAbActSem (a) \land EpsSem \notin os1 \land EpsSem \in os2
⇒ ou = NormOU2 (MergeOU (ContActSem (a, os1x), FinalAbActSem (a) ) )

= NormOU2 (ContActSem (a, os1x) )

from theorem 28

and ⇒ our1 = Cont1ActSem (a, SeqCompOS (os1x, os3) ) \land our2 = FinalAbActSem (a)
⇒ our = NormOU2 (MergeOU (Cont1ActSem (a, SeqCompOS (os1x, os3) ),
FinalAbActSem (a) )
= NormOU2 (Cont1ActSem (a, SeqCompOS (os1x, os3) ))
= ouc.

(xiii): ou1 = ContActSem (a, os1x) \land ou2 = FinalAbActSem (a) \land EpsSem \notin os1 \land EpsSem \in os2
⇒ ou = NormOU2 (MergeOU (ContActSem (a, os1x), FinalAbActSem (a) ) )

= NormOU2 (ContActSem (a, os1x) )

from theorem 28

and ⇒ our1 = Cont1ActSem (a, SeqCompOS (os1x, os3) ) \land our2 = FinalAbActSem (a)
⇒ our = NormOU2 (MergeOU (Cont1ActSem (a, SeqCompOS (os1x, os3) ),
FinalAbActSem (a) )
= NormOU2 (Cont1ActSem (a, SeqCompOS (os1x, os3) ))
= ouc.
An Operational Semantics for the Dataflow Algebra
A. J. Cowling

\[ \text{MergeOU} (\text{FinalAbActSem} (a), \text{ou3}) \]
\[ = \text{NormOU2} (\text{MergeOU} (\text{Cont1ActSem} (a, \text{SeqCompOS} (\text{os1x}, \text{os3})), \text{ou3})) \]
\[ = \text{ouc}. \]

(xiv):
\[ \text{ou1} = \text{ContActSem} (a, \text{os1x}) \land \text{ou2} = \text{FinalAbActSem} (a) \land \text{EpsSem} \in \text{os1} \land \text{EpsSem} \notin \text{os2} \]
\[ \Rightarrow \text{ou} = \text{ContActSem} (a, \text{os1x}) \land \text{ou1e} = \text{ou3} \land \text{ou2e} = \text{ou3} \]
\[ a \in h3 \Rightarrow \text{ouc} = \text{NormOU2} (\text{MergeOU} (\text{MergeOU} (\text{Cont1ActSem} (a, \text{SeqCompOS} (\text{os1x}, \text{os3})), \text{ou3}), \text{ou3}))) \]
\[ = \text{ouc}. \]

and
\[ \Rightarrow \text{our1} = \text{Cont1ActSem} (a, \text{SeqCompOS} (\text{os1x}, \text{os3})) \land \text{our1e} = \text{ou3} \land \text{our2} = \text{FinalAbActSem} (a) \]
\[ \Rightarrow \text{our1p} = \{ \text{Cont1ActSem} (a, \text{SeqCompOS} (\text{os1x}, \text{os3})), \text{ou3} \} \land \text{our2p} = \{ \text{FinalAbActSem} (a), \text{ou3} \} \]
\[ \Rightarrow \text{our} = \text{NormOU2} (\text{MergeOU} (\text{MergeOU} (\text{Cont1ActSem} (a, \text{SeqCompOS} (\text{os1x}, \text{os3})), \text{ou3}))), \text{ou3}), \]
\[ \text{MergeOU} (\text{FinalAbActSem} (a, \text{ou3}))) \]
\[ = \text{NormOU2} (\text{MergeOU} (\text{Cont1ActSem} (a, \text{SeqCompOS} (\text{os1x}, \text{os3}))), \text{ou3})) \]
\[ = \text{ouc}. \]

(xv):
\[ \text{ou1} = \text{ContActSem} (a, \text{os1x}) \land \text{ou2} = \text{ContActSem} (a, \text{os2x}) \land \text{EpsSem} \notin \text{os1} \land \text{EpsSem} \notin \text{os2} \]
\[ \Rightarrow \text{ou1} = \text{ContActSem} (a, \text{os1x} \cup \text{os2x})) \]
\[ = \text{ContActSem} (a, \text{NormOS2} (\text{os1x} \cup \text{os2x})) \]
\[ \Rightarrow \text{ouc} = \text{NormOU2} (\text{Cont1ActSem} (a, \text{SeqCompOS} (\text{NormOS2} (\text{os1x} \cup \text{os2x}), \text{os3}))) \]
\[ = \text{NormOU2} (\text{ContActSem} (a, \text{SeqCompOS} (\text{NormOS2} (\text{os1x} \cup \text{os2x}), \text{os3}))), \text{ou3})) \]
\[ = \text{ContActSem} (a, \text{NormOS2} (\text{SeqCompOS} (\text{NormOS2} (\text{os1x} \cup \text{os2x}), \text{os3}))) \]
\[ \text{from theorems 37 & 14} \]
\[ = \text{ContActSem} (a, \text{NormOS2} (\text{SeqCompOS} (\text{os1x} \cup \text{os2x}), \text{os3}))) \]
\[ \text{from the induction hypothesis} \]
\[ = \text{ouc}. \]

(xvi):
\[ \text{ou1} = \text{ContActSem} (a, \text{os1x}) \land \text{ou2} = \text{ContActSem} (a, \text{os2x}) \land \text{EpsSem} \notin \text{os1} \land \text{EpsSem} \notin \text{os2} \]
\[ \Rightarrow \text{ou} = \text{ContActSem} (a, \text{NormOS2} (\text{os1x} \cup \text{os2x}))) \]
\[ = \text{ContActSem} (a, \text{NormOS2} (\text{os1x} \cup \text{os2x}))) \]
\[ \Rightarrow \text{ouc} = \text{NormOU2} (\text{MergeOU} (\text{Cont1ActSem} (a, \text{SeqCompOS} (\text{NormOS2} (\text{os1x} \cup \text{os2x}), \text{os3}))), \text{ou3})) \]
\[ = \text{NormOU2} (\text{MergeOU} (\text{Cont1ActSem} (a, \text{SeqCompOS} (\text{NormOS2} (\text{os1x} \cup \text{os2x}), \text{os3}))), \text{ou3}))) \]
\[ \text{from theorem 47} \]
\[ = \text{ContActSem} (a, \text{NormOS2} (\text{SeqCompOS} (\text{NormOS2} (\text{os1x} \cup \text{os2x}), \text{os3}))), \text{ou3})) \]
\[ \text{from the induction hypothesis} \]
\[ = \text{ouc}. \]

(xvii):
\[ \text{ou1} = \text{ContActSem} (a, \text{os1x}) \land \text{ou2} = \text{ContActSem} (a, \text{os2x}) \land \text{EpsSem} \notin \text{os1} \land \text{EpsSem} \notin \text{os2} \]
\[ \Rightarrow \text{ou} = \text{ContActSem} (a, \text{NormOS2} (\text{os1x} \cup \text{os2x)))) \]
\[ = \text{ContActSem} (a, \text{NormOS2} (\text{os1x} \cup \text{os2x)))) \]
\[ = \text{ouc}. \]

and
\[ \Rightarrow \text{our1} = \text{Cont1ActSem} (a, \text{SeqCompOS} (\text{os1x}, \text{os3}))) \]
\[ \land \text{our2} = \text{Cont1ActSem} (a, \text{SeqCompOS} (\text{os2x}, \text{os3}))) \]
\[ \Rightarrow \text{our} = \text{NormOU2} (\text{MergeOU} (\text{Cont1ActSem} (a, \text{SeqCompOS} (\text{os1x}, \text{os3}))), \text{ou3})) \]
\[ = \text{NormOU2} (\text{MergeOU} (\text{Cont1ActSem} (a, \text{SeqCompOS} (\text{os1x}, \text{os3}))), \text{ou3})) \]
\[ = \text{NormOU2} (\text{MergeOU} (\text{Cont1ActSem} (a, \text{SeqCompOS} (\text{os1x}, \text{os3}))), \text{ou3})) \]
\[ \text{from theorem 47} \]
\[ = \text{NormOU2} (\text{MergeOU} (\text{Cont1ActSem} (a, \text{SeqCompOS} (\text{NormOS2} (\text{os1x} \cup \text{os2x})), \text{os3}))), \text{ou3})) \]
\[ \text{from the induction hypothesis} \]
\[ = \text{ouc}. \]
∧ our2 = Cont1ActSem (a, SeqCompOS (os2x, os3)) ∧ our2e = ou3
⇒ our2p = { Cont1ActSem (a, SeqCompOS (os2x, os3)), ou3 }
⇒ our = NormOU2 (MergeOU (Cont1ActSem (a, SeqCompOS (os1x, os3)),
MeruuOU (Cont1ActSem (a, SequuPSou (os2x, os3)), ou3))
= NormOU2 (MergeOU (MergeOU (Cont1ActSem (a, SequuPSou (os1x, os3)),
Cont1ActSem (a, SequuPSou (os2x, os3))), ou3))
= NormOU2 (MergeOU (Cont1ActSem (a, NormOS2 (SequuPSou (os1x, os3) ∪ SequuPSou (os2x, os3)), ou3)) from theorem 47
= NormOU2 (MergeOU (Cont1ActSem (a, NormOS2 (SequuPSou (os1x ∪ os2x), os3)), ou3)) from the induction hypothesis
⇒ our1 = Cont1ActSem (a, os1x) ∧ our2 = Cont1ActSem (a, os2x) ∧ EpsSem ∈ es1 ∧ EpsSem ∈ es2
⇒ ou = Cont1ActSem (a, NormOS2 (os1x ∪ os2x)) ∧ our1e = ou3 ∧ our2e = ou3
a ∈ h3 ⇒ ouc = NormOU2 (MergeOU (Cont1ActSem (a, SequuPSou (NormOS2 (os1x ∪ os2x)), os3)), ou3, ou3)
⇒ our1p = { Cont1ActSem (a, SequuPSou (os1x, os3)), ou3 }
⇒ our2p = { Cont1ActSem (a, SequuPSou (os2x, os3)), ou3 }
⇒ our = NormOU2 (MergeOU (Cont1ActSem (a, SequuPSou (NormOS2 (os1x ∪ os2x), os3)),
Cont1ActSem (a, SequuPSou (os2x, os3))), ou3))
= NormOU2 (MergeOU (Cont1ActSem (a, NormOS2 (SequuPSou (os1x, os3) ∪ SequuPSou (os2x, os3))), ou3)) from theorem 47
= NormOU2 (MergeOU (Cont1ActSem (a, NormOS2 (SequuPSou (os1x ∪ os2x), os3))), ou3)) from the induction hypothesis
= ouc.

Hence, the inductive case holds for all of these different combinations, so that if the theorem holds for all os1 and os2 such that HeightOS (os1) < n and HeightOS (os2) < n, it also holds for all os1 and os2 such that HeightOS (os1) = n and HeightOS (os2) = n, for any natural number n. Hence, by induction the theorem holds for all n, and so holds.

The other form of the distributive property for SeqCompOS, which involves its second parameter, is expressed as the following theorem.

**Theorem 49.**
\[ ∀ os1, os2 : OpSem | os1 ≠ ∅ ∧ OSNorm012 (os1) ∧ os2 ≠ ∅ ∧ OSNorm012 (os2) \]
\[ NormOS2 (SequuPSou (os1, NormOS2 (os2 ∪ os3))) = NormOS2 (SequuPSou (os1, os2) ∪ SequuPSou (os1, os3)). \]

**Proof.**
The proof here is by induction over the height of the forest os1. It is based on the observation that, from the definition of SequuPSou,
\[ ∀ ou1 : OpSUnit | ou1 ∈ os1 \] NormOS2 (SequuPSou (ou1, NormOS2 (os2 ∪ os3)))
\[ = NormOS2 (SequuPSou (ou1, os2) ∪ SequuPSou (ou1, os3))) \]
\[ = NormOS2 (SequuPSou (os1, NormOS2 (os2 ∪ os3))) \]
\[ = NormOS2 (SequuPSou (os1, os2) ∪ SequuPSou (os1, os3)). \]

Hence, the analysis just needs to consider the various possible cases for the construction of ou1, in order to establish the left-hand side of this implication, from which the theorem follows.

**Base case:** HeightOS (os1) = 1 ⇒ HeightOU (ou1) = 1.

From theorem 5 there are four possible forms for ou1 to be considered, namely: EpsSem, PhiSem, FinalActSem (a) and FinalAbActSem (a).
Case (i): \( \text{ou1} = \text{EpsSem} \Rightarrow \text{NormOS2} (\text{SeqCompOU (ou1, os2)} \cup \text{SeqCompOU (ou1, os3)} ) = \text{NormOS2} (\text{SeqCompOU (EpsSem, os2)} \cup \text{SeqCompOU (EpsSem, os3)} ) = \text{NormOS2} (\text{os2} \cup \text{os3}) \) from theorem 28.

Case (ii): \( \text{ou1} = \text{PhiSem} \Rightarrow \text{NormOS2} (\text{SeqCompOU (ou1, os2)} \cup \text{SeqCompOU (ou1, os3)} ) = \text{NormOS2} (\text{SeqCompOU (PhiSem, os2)} \cup \text{SeqCompOU (PhiSem, os3)} ) = \{ \text{PhiSem} \} \) from theorem 47.

Case (iii): \( \exists a : \text{PA} \bullet \text{ou1} = \text{FinalActSem (a)} \Rightarrow \text{NormOS2} (\text{SeqCompOU (ou1, os2)} \cup \text{SeqCompOU (ou1, os3)} ) = \text{NormOS2} (\text{SeqCompOU (FinalActSem (a), os2)} \cup \text{SeqCompOU (FinalActSem (a), os3)} ) = \{ \text{Cont1ActSem (a, os2)} \cup \text{Cont1ActSem (a, os3)} \}) = \{ \text{NormOU2 (MergeOU (Cont1ActSem (a, os2), Cont1ActSem (a, os3))} \} = \text{NormOS2} (\text{SeqCompOU (ContActSem (a, os1x), NormOS2 (os2 \cup os3))} ) \).

Case (iv): \( \exists a : \text{PA} \bullet \text{ou1} = \text{FinalAbActSem (a)} \Rightarrow \text{NormOS2} (\text{SeqCompOU (ou1, os2)} \cup \text{SeqCompOU (ou1, os3)} ) = \text{NormOS2} (\text{SeqCompOU (FinalAbActSem (a), os2)} \cup \text{SeqCompOU (FinalAbActSem (a), os3)} ) = \{ \text{FinalAbActSem (a)} \} = \text{NormOS2} (\text{SeqCompOU (FinalAbActSem (a), NormOS2 (os2 \cup os3))} ) \).

Hence, the theorem holds for all four of these possible constructions of \( \text{ou1} \), and so the base case holds.

**Inductive case:** \( \text{HeightOS (os1)} > 1 \Rightarrow \text{HeightOU (ou1)} > 1 \).

This gives rise to one new construction for \( \text{ou1} \), namely:

\( \exists \text{ou1x : OpSem | os1x} \neq \emptyset \land \text{os1x} \neq \{ \text{EpsSem} \} \land \text{os1x} \neq \{ \text{EpsSem} \} \bullet \text{ou1} = \text{ContActSem (a, os1x)} \).

Then, the induction hypothesis is that the theorem holds for all \( \text{os1x} \) such that \( \text{HeightOS (os1x)} < n \) for any natural number \( n > 1 \), and the induction step is to show that therefore it holds for all \( \text{ou1} \) with \( \text{HeightOU (ou1)} = n \). The analysis of this is as follows.

\[
\text{NormOS2} (\text{SeqCompOU (ou1, os2)} \cup \text{SeqCompOU (ou1, os3)} ) = \text{NormOS2} (\text{SeqCompOU (ContActSem (a, os1x), os2)} \cup \text{SeqCompOU (ContActSem (a, os1x), os3)} ) = \text{NormOS2} (\text{Cont1ActSem (a, SeqCompOS (os1x, os2))}) \cup \text{Cont1ActSem (a, SeqCompOS (os1x, os3))}) = \text{NormOU2 (MergeOU (Cont1ActSem (a, SeqCompOS (os1x, os2)), Cont1ActSem (a, SeqCompOS (os1x, os3))))} = \text{NormOS2} (\text{Cont1ActSem (a, NormOS2 (SeqCompOS (os1x, os2) \cup SeqCompOS (os1x, os3))))} \) from theorem 28.

\[
= \text{NormOS2} (\text{Cont1ActSem (a, NormOS2 (SeqCompOS (os1x, os2) \cup SeqCompOS (os1x, os3))))} \) from theorem 47.

\[
= \text{NormOS2} (\text{Cont1ActSem (a, NormOS2 (SeqCompOS (os1x, NormOS2 (os2 \cup os3))))} \) from the induction hypothesis.

\[
= \text{NormOS2} (\text{SeqCompOU (ContActSem (a, os1x), NormOS2 (os2 \cup os3))}) = \text{NormOS2} (\text{SeqCompOU (ou1, NormOS2 (os2 \cup os3))}).
\]

Hence the inductive step holds, and so by induction the theorem holds for all \( n \geq 1 \), and so holds.

The next intermediate result concerning the function \( \text{SeqCompOS} \) defines what is effectively the associative property for it, and is expressed as the following theorem.
Theorem 50.
\[ \forall \text{os1, os2, os3 : OpSem} | \text{os1} \neq \emptyset \land \text{OSNorm012 (os1)} \land \text{os2} \neq \emptyset \land \text{OSNorm012 (os2)} \land \text{os3} \neq \emptyset \land \text{OSNorm012 (os3)} \] 
\[ \text{NormOS2 (SeqCompOS (NormOS2 (SeqCompOS (os1, os2)) , os3))} \]
\[ = \text{NormOS2 (SeqCompOS (os1, NormOS2 (SeqCompOS (os2, os3)))}) \].

Proof.
The proof is by induction over the height of the forest os1. Throughout it, we let the following variables of type OpSem be defined as:

\[ \text{os12} = \text{NormOS2 (SeqCompOS (os1, os2))} \], \text{os1} = \text{NormOS2 (SeqCompOS (os12, os3))} \]
\[ \text{os23} = \text{NormOS2 (SeqCompOS (os2, os3))} \], and \text{osr} = \text{NormOS2 (SeqCompOS (os1, os23))}

so that the result of the theorem can be expressed as osl = osr. We also let the following variables of type P Act be defined as:

\[ h1 = \text{Heads (os1)}, h2 = \text{Heads (os2)}, \text{and } h3 = \text{Heads (os3)} \]

and as appropriate (meaning, when these objects actually exist, as in the proofs of theorem 41 and 48), then for any action a we let:

\[ \text{ou1} : \text{OpSUnit be such that ou1} \in \text{os1} \land \text{GetHead (ou1)} = a, \]
\[ \text{ou2} : \text{OpSUnit be such that ou2} \in \text{os2} \land \text{GetHead (ou2)} = a, \]
\[ \text{ou3} : \text{OpSUnit be such that ou3} \in \text{os3} \land \text{GetHead (ou3)} = a, \]
\[ \text{ou12} : \text{OpSUnit be such that ou12} \in \text{os12} \land \text{GetHead (ou12)} = a, \]
\[ \text{ou : OpSUnit be such that ou} \in \text{osl} \land \text{GetHead (ou)} = a, \]
\[ \text{ou23} : \text{OpSUnit be such that ou23} \in \text{os23} \land \text{GetHead (ou23)} = a, \]
\[ \text{and } \text{our : OpSUnit be such that our} \in \text{osr} \land \text{GetHead (our)} = a. \]

As in the proof of theorem 48, since each of os1 and os2 are in second normal form the argument can in principle be structured in terms of analysing the elements of them and of the result sets that have a particular head a. Here, though, we are mainly only interested in the case where a is in h1, since the only other case where such an element could appear in the result would be if either os1 includes EpsSem as an element and a is in h2, or both os1 and os2 include EpsSem as an element and a is in h3. Thus, for each of the possible constructions of ou1 we have to consider separately the two cases where os1 does or does not include EpsSem, and similarly for each of the possible constructions of ou2 we have to consider separately the two cases where os2 does or does not include EpsSem.

Unlike the proof of theorem 48, though, the only singleton set that we need to analyse separately is \{ \text{PhiSem} \}. Although an element EpsSem can occur as a singleton set, it is simpler to analyse it as just one of the possible cases for ou1 or ou2 as appropriate. This means that then the combinations of cases that arise where os1 or os2 include EpsSem can be analysed using the following general argument.

**Combination case:** this applies when there are two different cases for the values of ou1 or ou2 (as appropriate) contributing to ou1 and ou2. These two cases will give rise to two values for each of ou12, ou23, ou1 and ou2 (where in some cases both values in one of these pairs will be the same, but the argument does not need to rely on this). The resultant pairs of values will be denoted by ou[1] and ou[2], and by our[1] and our[2]. Then, ou and our will be constructed as

\[ \text{ou} = \text{NormOU2 (MergeOU (ou[1], ou[2])} \] \land \text{our} = \text{NormOU2 (MergeOU (our[1], our[2])}} \]

and hence if the analyses of the individual cases have already shown that ou[1] = our[1] \land ou[2] = our[2], then it follows immediately that ou = our.

Furthermore, for some of the possible constructions of ou1 we have to consider not just the case where ou2 with head a exists, but the set of all possible elements of os2, irrespective of their heads, since they will all be descendents of the object constructed from ou1. Thus, for the case where any action b is allowed to range over all the possible elements of h2, this pair notation is extended to let arbitrary elements of os2 be denoted by ou2[b], where \( b \in h2 \land \text{GetHead (ou2b)} = b \). Then, for each of the values ou12, ou23, ou1 and ou2 (as appropriate) it will sometimes be convenient to have a notation for the corresponding set produced by letting b range over all the elements of h2. If ou’ stands for any of ou2, ou12, ou23, ou1 or our, then within this proof such a set will be denoted either as

\[ \text{ou'bs, where ou'bs} = \{ \forall b : \text{Act} | b \in h2 \land \text{ou'[b]} \} , \]

or simply as \{ ou'[b] \},

and this latter notation will be extended to cover constructions including b, such as \{ \text{Cont1ActSem ( [b], os3) } \}. For example, using this notation, ou2bs or { ou2[b] } would both be alternative ways of writing the object os2.

Given these various notations, then the base and inductive cases for the induction are as follows.
Base case: HeightOS (os1) = 1.
From theorem 5 there are four possible forms for ou1: EpsSem, PhiSem, FinalActSem (a) and FinalAbActSem (a). Of these, PhiSem can only occur if # os1 = 1, because os1 is in second normal form. The case where ou1 = EpsSem can occur either as a singleton set or in combination with the other two cases, but the analyses of the latter are covered by the combination case above, and so do not need to be considered separately. There are then three cases that are independent of os2, as follows.

(i): os1 = { PhiSem } \implies os12 = { PhiSem } \land os1 = { PhiSem } \\
\land os23 = NormOS2 (SeqCompOS (os2, os3)) \land osr = { PhiSem } \\
\implies osl = osr.

(ii): ou1 = EpsSem \implies ou12 = ou2 \land ou1 = NormOU2 (SeqCompOU (ou2, os3)) \\
\land os23 = NormOU2 (SeqCompOU (ou2, os3)) \land our = NormOU2 (SeqCompOU (ou2, os3)) \\
\implies oul = our.

(iii): ou1 = FinalAbActSem (a) \implies ou12 = FinalAbActSem (a) \land ou1 = FinalAbActSem (a) \\
\land ou23[b] = NormOU2 (SeqCompOU (ou2[b], os3)) \land our = FinalAbActSem (a) \\
\implies oul = our.

If ou1 = FinalActSem (a) then the analysis does depend on os2, where there are five possible forms for ou2, plus the possible combinations of these with EpsSem, but the latter are covered by the combination case above and do not need to be analysed separately. Hence, the following cases need to be considered.

(iv): ou1 = FinalActSem (a) \land os2 = { PhiSem } \\
\implies ou12 = FinalAbActSem (a) \land ou1 = FinalAbActSem (a) \\
\land ou23 = { PhiSem } \land our = FinalAbActSem (a) \\
\implies oul = our.

(v): ou1 = FinalActSem (a) \land ou2 = EpsSem \\
\implies ou12 = FinalActSem (a) \land ou1 = Cont1ActSem (a, os3) \\
\land ou23 = os3 \land our = Cont1ActSem (a, os3) \\
\implies oul = our.

(vi): ou1 = FinalActSem (a) \land ou2[b] = FinalActSem (b) \\
\implies ou12 = Cont1ActSem (a, { FinalActSem (b) }) \\
\land ou1 = Cont1ActSem (a, { FinalActSem (b), os3 }) \\
\land ou23[b] = Cont1ActSem (b, os3) \land our = Cont1ActSem (a, { FinalActSem (b), os3 }) \\
\implies oul = our.

(vii): ou1 = FinalActSem (a) \land ou2[b] = FinalAbActSem (b) \\
\implies ou12 = Cont1ActSem (a, { FinalAbActSem (b) }) \\
\land ou1 = Cont1ActSem (a, { FinalAbActSem (b) }) \\
\land ou23[b] = Cont1ActSem (b, os3) \land our = Cont1ActSem (a, { FinalAbActSem (b) }) \\
\implies oul = our.

(viii): ou1 = FinalActSem (a) \\
\land \exists ou2x : OpSem | os2x \neq \emptyset \land os2x \neq \{ EpsSem \} \land os2x \neq \{ EpsSem \} \\
\land ou2[b] = ContActSem (b, os2x) \\
\implies ou12 = Cont1ActSem (a, { ContActSem (b, os2x) }) \\
= Cont1ActSem (a, { ContActSem (b, os2x) }) \\
\land ou1 = Cont1ActSem (a, { ContActSem (b, NormOS2 (SeqCompOS (os2x, os3))) }) \\
\land ou23[b] = Cont1ActSem (b, NormOS2 (SeqCompOS (os2x, os3))) \\
\land our = Cont1ActSem (a, { ContActSem (b, NormOS2 (SeqCompOS (os2x, os3))) }) \\
\implies oul = our.

Inductive case: HeightOS (os1) > 1.
This gives rise to one new construction for ou1, namely \\
\exists os1x : OpSem | isActive (os1x) \bullet ou1 = ContActSem (a, os1x).

Then, the induction hypothesis is that the theorem holds for all os1x such that HeightOS (os1x) < n for any natural number n > 1, and the induction step is to show that therefore it holds for all ou1 with HeightOU (ou1) = n. The analysis of this is as follows.

ou1 = ContActSem (a, os1x) \\
\implies ou12 = NormOU2 (Cont1ActSem (a, SeqCompOS (os1x, os2))) \\
= NormOU2 (Cont1ActSem (a, SeqCompOS (os1x, os2))) \\
= Cont1ActSem (a, NormOS2 (SeqCompOS (os1x, os2))) \\
\land ou1 = NormOU2 (Cont1ActSem (a, SeqCompOS (NormOS2 (SeqCompOS (os1x, os2)), os3))) 

67
= NormOU2 (ContActSem (a, SeqCompOS (NormOS2 (SeqCompOS (os1x, os2) ), os3) ) )
  from theorem 36
= ContActSem (a, NormOS2 (SeqCompOS (NormOS2 (SeqCompOS (os1x, os2) ), os3) ) )
  from theorem 35
\wedge os23 = NormOS2 (SeqCompOS (os2, os3) )
= NormOU2 (Cont1ActSem (a, SeqCompOS (os1x, os23) ) )
  from theorem 36
= ContActSem (a, NormOS2 (SeqCompOS (os1x, os3) ) )
  from theorem 35
= ContActSem (a, NormOS2 (SeqCompOS (NormOS2 (SeqCompOS (os1x, os2) ), os3) ) )
  from the induction hypothesis
\Rightarrow oul = our.

Hence the inductive case holds, and so by induction the theorem holds for all values of n, and so holds.

The final property to be established for the function SeqCompOS is that the object \{ EpsSem \} forms both a left and a right identity for it. For the left identity this property is expressed as the following theorem.

**Theorem 51.**
\[
\forall os : OpSem \ | \ os \neq \emptyset \wedge OSNorm01 (os) \bullet SeqCompOS ( \{ EpsSem \}, os) = os.
\]

**Proof.**
The proof follows directly from the definition of SeqCompOU.

The corresponding property for the right identity is expressed as the following theorem, but establishing it requires rather more work.

**Theorem 52.**
\[
\forall os : OpSem \ | \ os \neq \emptyset \wedge OSNorm01 (os) \bullet SeqCompOS (os, \{ EpsSem \}) = os.
\]

**Proof.**
The proof is by induction over the height of the forest os, and it relies on the observation that, from the definition of SeqCompOS,
\[
[ \forall ou : OpSUnit \ | \ ou \in os \bullet SeqCompOU (ou, \{ EpsSem \}) = \{ ou \}] \Rightarrow SeqCompOS (os, \{ EpsSem \}) = os.
\]

**Base case.**
The base case is that HeightOS (os) = 1 \Rightarrow HeightOU (ou) = 1, so that from theorem 5 there are four possible constructions of ou to be analysed, as follows.

(i) \quad ou = EpsSem \Rightarrow SeqCompOU (ou, \{ EpsSem \}) = \{ EpsSem \} = \{ ou \}.
(ii) \quad ou = PhiSem \Rightarrow SeqCompOU (ou, \{ EpsSem \}) = \{ PhiSem \} = \{ ou \}.
(iii) \quad \exists a : PA \bullet ou = FinalActSem (a) \Rightarrow SeqCompOU (ou, \{ EpsSem \}) = \{ Cont1ActSem (a, \{ EpsSem \}) \} = \{ FinalActSem (a) \} = \{ ou \}.
(iv) \quad \exists a : PA \bullet ou = FinalAbActSem (a) \Rightarrow SeqCompOU (ou, \{ EpsSem \}) = \{ FinalAbActSem (a) \} = \{ ou \}.

Hence the theorem holds for all four of these possible constructions, and so holds for the base case.

**Inductive Case.**
The induction hypothesis is that, for some natural number n > 1, the theorem holds for all os such that HeightOS (os) < n, and the induction step is to show that therefore it holds for all os such that HeightOS (os) = n. In addition to the possible constructions for ou that have already been analysed as part of the base case, this case adds one further construction, for which the analysis is as follows.

(v) \quad \exists os1 : OpSem \ | \ IsActive (os1) \bullet ou = ContActSem (a, os1) \Rightarrow SeqCompOU (ou, \{ EpsSem \}) = \{ Cont1ActSem (a, SeqCompOS (os1, \{ EpsSem \})) \} = \{ ContActSem (a, SeqCompOS (os1, \{ EpsSem \})) \}
  since IsActive (SeqCompOS (os1, \{ EpsSem \})) from theorem 36
Hence the theorem holds for the inductive case too, and so by induction it holds for all \( n \), and so holds.

9. Semantic Functions

Given these auxiliary functions and their properties, we can now define the semantic function that produces the equivalent of derivation sequences for DFA constructions, but with all of the unnecessary unlabelled transitions removed (i.e., elided). Thus, while the fundamental nature of the operational semantics for a system is that they are defined by the transition relation for that system, there is a sense in which this function could also be said to define the operational semantics for the DFA, in that its significance is analogous to the function \( \text{Sem} \) that defines the denotational semantics. For this reason this function is called \( \text{OpS} \), and it has signature \( \text{SeqConst} \rightarrow \text{OpSem} \), and is defined as follows.

\[
\begin{align*}
\text{OpS} \ (a) & \equiv \{ \text{FinalActSem} \ (a) \}, \text{where} \ a \in \text{PA} \\
\text{OpS} \ (\varepsilon) & \equiv \{ \text{EpsSem} \} \\
\text{OpS} \ (\phi) & \equiv \{ \text{PhiSem} \} \\
\text{OpS} \ (s_1 \ | \ s_2) & \equiv \text{NormOS2} \ (\text{OpS} \ (s_1) \cup \text{OpS} \ (s_2)) \\
\text{OpS} \ (s_1 \ ; \ s_2) & \equiv \text{NormOS2} \ (\text{SeqCompOS} \ (\text{OpS} \ (s_1), \text{OpS} \ (s_2)))
\end{align*}
\]

The first property that then needs to be established for this function is that it does actually produce structures that contain no unnecessary unlabelled transitions, and this is expressed as the following theorem.

Theorem 53.
\[
\forall \ s : \text{SeqConst} \quad \text{OSNorm0} \ (\text{OpS} \ (s)) \land \text{OSNorm1} \ (\text{OpS} \ (s)) \land \text{OSNorm2} \ (\text{OpS} \ (s)).
\]

Proof.
The proof is by structural induction over \( s \), with the function \( \text{SCC} \) as the metric for the induction.

Base case.
The base case is that \( \text{SCC} \ (s) = 1 \), meaning that \( s \) must be an action, and this gives rise to the following three sub-cases, which all have the same structure.

Sub-case (i): \( s = a \) where \( a \in \text{PA} \)

\[
\begin{align*}
\Rightarrow \ & \text{OpS} \ (s) = \{ \text{FinalActSem} \ (a) \} \\
\Rightarrow & \quad \text{OSNorm0} \ (\text{OpS} \ (s)) \quad \text{from theorem 4} \\
\text{and} \ & \quad \text{OSNorm1} \ (\text{OpS} \ (s)) \quad \text{from theorem 8} \\
\text{and} \ & \quad \text{OSNorm2a} \ (\text{OpS} \ (s)) \quad \text{by calculation} \\
\text{and} \ & \quad \text{OSNorm2b} \ (\text{OpS} \ (s)) \quad \text{by calculation} \\
\Rightarrow & \quad \text{OSNorm2} \ (\text{OpS} \ (s)) \quad \text{by calculation}.
\end{align*}
\]

Sub-case (ii): \( s = \varepsilon \Rightarrow \text{OpS} \ (s) = \{ \text{EpsSem} \}

\[
\begin{align*}
\Rightarrow & \quad \text{OSNorm0} \ (\text{OpS} \ (s)) \quad \text{from theorem 4} \\
\text{and} \ & \quad \text{OSNorm1} \ (\text{OpS} \ (s)) \quad \text{from theorem 8} \\
\text{and} \ & \quad \text{OSNorm2a} \ (\text{OpS} \ (s)) \quad \text{by calculation} \\
\text{and} \ & \quad \text{OSNorm2b} \ (\text{OpS} \ (s)) \quad \text{by calculation} \\
\Rightarrow & \quad \text{OSNorm2} \ (\text{OpS} \ (s)) \quad \text{by calculation}.
\end{align*}
\]

Sub-case (iii): \( s = \phi \Rightarrow \text{OpS} \ (s) = \{ \text{PhiSem} \}

\[
\begin{align*}
\Rightarrow & \quad \text{OSNorm0} \ (\text{OpS} \ (s)) \quad \text{from theorem 4} \\
\text{and} \ & \quad \text{OSNorm1} \ (\text{OpS} \ (s)) \quad \text{from theorem 8} \\
\text{and} \ & \quad \text{OSNorm2a} \ (\text{OpS} \ (s)) \quad \text{by calculation} \\
\text{and} \ & \quad \text{OSNorm2b} \ (\text{OpS} \ (s)) \quad \text{by calculation} \\
\Rightarrow & \quad \text{OSNorm2} \ (\text{OpS} \ (s)) \quad \text{by calculation}.
\end{align*}
\]

Hence, the theorem holds for the base case.
Inductive case.
The induction hypothesis is that, for any $n > 1$, the theorem holds for all $s : \text{SeqConst}$ such that $\text{SCC}(s) < n$, and so the induction step is to show from this that the theorem must also hold for any $s : \text{SeqConst}$ such that $\text{SCC}(s) = n$. There are then two sub-cases to be analysed, depending on the construction of $s$: one for the case where it is constructed by alternation, and the other for the case where it is constructed by sequencing.

Alternation sub-case: $s = s_1 | s_2$, where $\text{SCC}(s_1) < n$ and $\text{SCC}(s_2) < n$

If we let $os_1, os_2, os_3 : \text{OpSem}$ be defined as $os_1 = \text{OpS}(s_1), os_2 = \text{Ops}(s_2)$ and $os_3 = os_1 \cup os_2$, then we have $\text{OpS}(s) = \text{NormOS2}(os_3)$

\[ \Rightarrow \text{OSNorm0}(os_1) \land \text{OSNorm1}(os_1) \land \text{OSNorm2}(os_1) \land \text{OSNorm0}(os_2) \land \text{OSNorm1}(os_2) \land \text{OSNorm2}(os_2) \] from the induction hypothesis

\[ \Rightarrow \forall ou_1, ou_2 : \text{OpSUnit} | ou_1 \in os_1 \land ou_2 \in os_2 \bullet \text{OUNorm0}(ou_1) \land \text{OUNorm1}(ou_1) \land \text{OUNorm2}(ou_1) \land \text{OUNorm0}(ou_2) \land \text{OUNorm1}(ou_2) \land \text{OUNorm2}(ou_2) \]

\[ \Rightarrow \forall ou_3 : \text{OpSUnit} | ou_3 \in os_3 \bullet \text{OUNorm0}(ou_3) \land \text{OUNorm1}(ou_3) \land \text{OUNorm2}(ou_3) \]

\[ \Rightarrow \text{OSNorm0}(os_3) \land \text{OSNorm1}(os_3) \land \text{OSNorm2}(os_3) \] from theorem 26

and

\[ \Rightarrow \text{OSNorm2}(\text{NormOS2}(os_3)) \] from theorem 27.

Hence, the theorem holds for this alternation sub-case of the inductive case.

Sequencing sub-case: $s = s_1 ; s_2$, where $\text{SCC}(s_1) < n$ and $\text{SCC}(s_2) < n$

If we let $os_1, os_2, os_3 : \text{OpSem}$ be defined as $os_1 = \text{OpS}(s_1), os_2 = \text{Ops}(s_2)$ and $os_3 = \text{SeqCompOS}(os_1, os_2)$, then we have $\text{OpS}(s) = \text{NormOS2}(os_3)$.

Again, from the induction hypothesis, since $\text{SCC}(s_1) < n$ and $\text{SCC}(s_2) < n$, we have

\[ \Rightarrow \text{OSNorm0}(os_1) \land \text{OSNorm1}(os_1) \land \text{OSNorm2}(os_1) \land \text{OSNorm0}(os_2) \land \text{OSNorm1}(os_2) \land \text{OSNorm2}(os_2) \] from the induction hypothesis

\[ \Rightarrow \text{OSNorm0}(os_3) \land \text{OSNorm1}(os_3) \land \text{OSNorm2}(os_3) \] from theorem 34

and

\[ \Rightarrow \text{OSNorm2}(\text{NormOS2}(os_3)) \] from theorem 27.

Hence, the theorem holds for this sequencing sub-case of the inductive case, and so the inductive case follows from the combination of these two sub-cases. Therefore, by induction the theorem holds for all values of $n$, and so holds.

Given this intermediate result, then the next property of the semantic function that needs to be established is that it produces structures that correctly model the notion of possibilities for the next action performed in executing a DFA sequence, as this notion is defined by the function $\text{SeqHeads}$ that is used in the definition of the transition relation. This property is expressed as the following theorem.

**Theorem 54.**

\[ \forall s : \text{SeqConst} \bullet \text{SeqHeads}(s) = \text{Heads}(\text{OpS}(s)). \]

**Proof.**

This has a similar structure to the proof of theorem 53, in that it is by structural induction over $s$, with the function $\text{SCC}$ as metric for the induction.

**Base case.**

The base case is that $\text{SCC}(s) = 1$, meaning that $s$ must be an action, and this gives rise to the following three sub-cases, which all have the same structure.

Sub-case (i): $s = a$ where $a \in \text{PA}$

\[ \Rightarrow \text{SeqHeads}(s) = \{ a \} \]

and

\[ \Rightarrow \text{OpS}(s) = \{ \text{FinalActSem}(a) \} \]

\[ \Rightarrow \text{Heads}(\text{OpS}(s)) = \{ a \} \]

\[ \Rightarrow \text{Heads}(\text{OpS}(s)) = \text{SeqHeads}(s). \]

Sub-case (ii): $s = \varepsilon \Rightarrow \text{SeqHeads}(s) = \{ \varepsilon \}$

and

\[ \Rightarrow \text{OpS}(s) = \{ \text{EpsSem} \} \]
⇒ Heads (OpS (s)) = { ϕ }
⇒ Heads (OpS (s)) = SeqHeads (s).

Sub-case (iii): s = ϕ ⇒ SeqHeads (s) = { ϕ }
and ⇒ OpS (s) = { PhiSem }
⇒ Heads (OpS (s)) = { ϕ }
⇒ Heads (OpS (s)) = SeqHeads (s).

Hence, the theorem holds for the base case.

Inductive case.

As in the proof of theorem 53, the induction hypothesis is that, for any n > 1, the theorem holds for all s : SeqConst such that SCC (s) < n, and so the induction step is to show from this that the theorem must also hold for any s : SeqConst such that SCC (s) = n. Again, therefore, there are two main sub-cases, depending on whether s is constructed by alternation or sequencing.

Alternation sub-case: s = s1 | s2, where SCC (s1) < n and SCC (s2) < n
Let h1, h2 : P PA be defined as h1 = SeqHeads (s1) and h2 = SeqHeads (s2), and let os1, os2, os3 : OpSem be defined as os1 = OpS (s1), os2 = OpS (s2) and os3 = os1 ∪ os2, so that OpS (s) = NormOS2 (os3).

Then, to cover all of the cases in the definition of SeqHeads there are four further possible sub-cases to consider, depending on whether or not h1 = { ϕ } and whether or not h2 = { ϕ }. The arguments are similar in each sub-case, and depend on the fact that, from theorem 34, any object constructed by OpS satisfies the conditions of theorems 24 and 30, and hence the only case where Heads when applied to such an object delivers the result { ϕ } will be if it is the object constructed as the singleton set { PhiSem }. Hence, the four sub-cases are as follows.

Sub-case (i): h1 = { ϕ } ∧ h2 = { ϕ } ⇒ SeqHeads (s) = { ϕ }
and ⇒ Heads (os1) = { ϕ } ∧ Heads (os2) = { ϕ }
⇒ os1 = { PhiSem } ∧ os2 = { PhiSem }
⇒ os3 = { PhiSem }
⇒ OpS (s) = { PhiSem }
⇒ Heads (OpS (s)) = { ϕ }
⇒ Heads (OpS (s)) = SeqHeads (s).

Sub-case (ii): h1 = { ϕ } ∧ h2 ≠ { ϕ } ⇒ SeqHeads (s) = h2
and ⇒ Heads (os1) = { ϕ } ∧ Heads (os2) = h2
⇒ os1 = { PhiSem } ∧ os2 ≠ { PhiSem }
⇒ os3 = { PhiSem } ∪ os2
⇒ OpS (s) = NormOS2 ( { PhiSem } ∪ os2 )
⇒ OpS (s) = os2
⇒ Heads (OpS (s)) = h2
⇒ Heads (OpS (s)) = SeqHeads (s).

Sub-case (iii): h1 ≠ { ϕ } ∧ h2 = { ϕ } ⇒ SeqHeads (s) = h1
and ⇒ Heads (os1) = h1 ∧ Heads (os2) = { ϕ }
⇒ os1 ≠ { PhiSem } ∧ os2 = { PhiSem }
⇒ os3 = os1 ∪ { PhiSem }
⇒ OpS (s) = NormOS2 ( os1 ∪ { PhiSem } )
⇒ OpS (s) = os1
⇒ Heads (OpS (s)) = h1
⇒ Heads (OpS (s)) = SeqHeads (s).

Sub-case (iv): h1 ≠ { ϕ } ∧ h2 ≠ { ϕ } ⇒ SeqHeads (s) = h1 ∪ h2
and ⇒ Heads (os1) = h1 ∧ Heads (os2) = h2
⇒ Heads (os3) = h1 ∪ h2
and ⇒ os1 ≠ { PhiSem } ∧ os2 ≠ { PhiSem }
⇒ PhiSem ∉ os1 ∧ PhiSem ∉ os2
⇒ PhiSem ∉ os3
⇒ { ϕ } ∉ Heads (os3)
and ⇒ OpS (s) = NormOS2 (os3)
⇒ Heads (OpS (s)) = Heads (NormOS2 (os3))

which, since \(\{\phi\} \notin \text{Heads (os3)}\), means that irrespective of the value of \(#\ os3\), from theorem 29

⇒ Heads (OpS (s)) = Heads (os3)
⇒ Heads (OpS (s)) = h1 ∪ h2
⇒ Heads (OpS (s)) = SeqHeads (s).

Hence, the theorem holds for all four of these sub-cases, and so holds for this alternation sub-case of the inductive case.

Sequencing sub-case: \(s = s1 \cdot s2\), where SCC (s1) < n and SCC (s2) < n

Again, let \(h1, h2 : PA\) be defined as \(h1 = \text{SeqHeads (s1)}\) and \(h2 = \text{SeqHeads (s2)}\), and let \(os1, os2, os3 : OpSem\) be defined as \(os1 = \text{OpS (s1)}\), \(os2 = \text{Ops (s2)}\) and \(os3 = \text{SeqCompOS (os1, os2)}\), so that \(\text{OpS (s)} = \text{NormOS2 (os3)}\).

Then, as in the alternation sub-case, there are four further possible sub-cases to consider. Two of these are the special cases of \(h1 = \{\varepsilon\}\) and \(h1 = \{\phi\}\), for which the arguments depend on the fact that (from theorems 34 and 30) the only objects of type \(\text{OpSem}\) that are in both first and second normal form and for which \(\text{Heads}\) will deliver either of the results \(\{\varepsilon\}\) or \(\{\phi\}\) are the objects constructed as the singleton sets \(\{\text{EpsSem}\}\) or \(\{\text{PhiSem}\}\). The other two sub-cases depend on whether or not \(\varepsilon \in h1\), and so the analyses of the four sub-cases are as follows.

Sub-case (i): \(h1 = \{\varepsilon\} \Rightarrow \text{SeqHeads (s)} = h2\)

and

⇒ \(os1 = \{\text{EpsSem}\}\)
⇒ \(os3 = os2\)
⇒ \(\text{OpS (s)} = \text{NormOS2 (os2)}\)
⇒ \(\text{OpS (s)} = os2\) from theorem 28
⇒ \(\text{Heads (OpS (s))} = h2\) from the induction hypothesis
⇒ \(\text{Heads (OpS (s))} = \text{SeqHeads (s)}\).

Sub-case (ii): \(h1 = \{\phi\} \Rightarrow \text{os1 = \{\text{PhiSem}\}}\)

⇒ \(os3 = os1\)
⇒ \(\text{OpS (s)} = \text{NormOS2 (os1)}\)
⇒ \(\text{Heads (OpS (s))} = h1\) from the induction hypothesis
⇒ \(\text{Heads (OpS (s))} = \text{SeqHeads (s)}\).

Sub-case (iii): \(h1 \neq \{\varepsilon\} \land \varepsilon \in h1 \Rightarrow \text{SeqHeads (s)} = (h1 - \{\varepsilon\}) \cup (h2 - \{\phi\})\)

and

⇒ \(os1 = \{\text{EpsSem}\} \cup \{\forall ou1 : \text{OpSUnit} | ou1 \in os1 \land ou1 \neq \text{EpsSem}\}\)
where \(\text{OSNorm2 (os1)} \land \# os1 > 1 \Rightarrow \text{ou1 = PhiSem}\)
⇒ \(os3 = \{os2\} \cup \{\forall ou1 : \text{OpSUnit} | ou1 \in os1 \land ou1 \neq \text{EpsSem} \land \text{SeqCompOU (ou1, os2)}\}\)
where \(\text{ou1.DoesAct}\)
⇒ \(\text{Heads (OpS (s))} = \text{Heads (NormOS2 (os3))}\)
⇒ \(\text{Heads (OpS (s))} = \text{Heads (os3)} - \{\phi\}\) from theorem 29, as \# os3 > 1
⇒ \(\text{Heads (OpS (s))} = (\text{Heads (os2)} \cup \text{Heads (os1 - \{EpsSem\}})) - \{\phi\}\)
⇒ \(\text{Heads (OpS (s))} = (\text{Heads (os1)} - \{\varepsilon\}) \cup (\text{Heads (os2)} - \{\phi\})\) since \(\phi \notin \text{Heads (os1)}\)
⇒ \(\text{Heads (OpS (s))} = (h1 - \{\varepsilon\}) \cup (h2 - \{\phi\})\) from the induction hypothesis
⇒ \(\text{Heads (OpS (s))} = \text{SeqHeads (s)}\).

Sub-case (iv): \(\varepsilon \notin h1 \Rightarrow \text{SeqHeads (s)} = h1\)

and

⇒ \(\forall ou1 : \text{OpSUnit} | ou1 \in os1 \land ou1 \neq \text{EpsSem} \land ou1 \neq \text{PhiSem}\) since \(\text{OSNorm2 (os1)}\)
⇒ \(\forall ou1 : \text{OpSUnit} | ou1 \in os1 \land ou1.DoesAct \land ou1.TheAct \in h1\)
⇒ \(os3 = \{\forall ou1 : \text{OpSUnit} | ou1 \in os1 \land \text{Cont1ActSem (ou1.TheAct, SeqCompOS (ou1.Rest, os2))}\}\)
⇒ \(\text{Heads (os3)} = \{\forall ou1 : \text{OpSUnit} | ou1 \in os1 \land \text{GetHead (Cont1ActSem (ou1.TheAct, SeqCompOS (ou1.Rest, os2))}\}\)
⇒ \(\text{Heads (os3)} = \{\forall ou1 : \text{OpSUnit} | ou1 \in os1 \land \text{ou1.TheAct}\}
⇒ \(\text{Heads (os3)} = h1\)
⇒ \([\text{Heads (OpS (s))} = \text{Heads (os3)} - \{\phi\}] \lor [\text{Heads (OpS (s))} = \text{Heads (os3)}]\) from theorem 24
⇒ \(\text{Heads (OpS (s))} = \text{Heads (os3)}\) from theorem 29
⇒ \(\text{Heads (OpS (s))} = \text{Heads (os3)}\) since \(\phi \notin \text{Heads (os3)}\)
⇒ \(\text{Heads (OpS (s))} = h1\)
⇒ \(\text{Heads (OpS (s))} = \text{SeqHeads (s)}\).
Hence, the theorem holds for all these sub-cases of the sequencing sub-case, and hence the inductive case follows from the combination of the alternation and sequencing sub-cases. Therefore, by induction the theorem holds for all values of n, and so holds.

The remaining property that we want to show for the semantic function $\text{OpS}$ is the one that defines its relationship with derivation sequences, and this is expressed as the following theorem.

**Theorem 55.**

\[ \forall s : \text{SeqConst} \quad \text{OpS} (s) = \text{NormOS2} (\text{NormOS1} (\text{DerSeq} (s)) ). \]

**Proof.**

This again has a similar structure to the proofs of theorems 53 and 54, in that it is by structural induction over s, with the function SCC as metric for the induction. This induction is also effectively over the height of the derivation sequences, although the two are not exactly equivalent metrics, but the approximate equivalence will emerge from the different cases in the proof, which correspond to the clauses in the definition of the transition relation, as these are reflected in the definition of DerSeq.

**Base case.**

The base case is that SCC ($s$) = 1, meaning that $s$ must be an action, and this gives rise to the following three sub-cases, which all have the same structure.

Sub-case (i): $s = a$ where $a \in \text{PA}$

\[ \Rightarrow \text{OpS} (s) = \{ \text{FinalActSem} (a) \} \]

and

\[ \Rightarrow \text{DerSeq} (s) = \{ \text{FinalActSem} (a) \} \]

from clause (i) of the transition relation

\[ \Rightarrow \text{NormOS1} (\text{DerSeq} (s)) = \{ \text{FinalActSem} (a) \} \]

by calculation

\[ \Rightarrow \text{NormOS2} (\text{NormOS1} (\text{DerSeq} (s))) = \{ \text{FinalActSem} (a) \} \]

by calculation

\[ = \text{OpS} (s). \]

Sub-case (ii): $s = \varepsilon$

\[ \Rightarrow \text{OpS} (s) = \{ \text{EpsSem} \} \]

and

\[ \Rightarrow \text{DerSeq} (s) = \{ \text{EpsSem} \} \]

from clause (ii) of the transition relation

\[ \Rightarrow \text{NormOS1} (\text{DerSeq} (s)) = \{ \text{EpsSem} \} \]

by calculation

\[ \Rightarrow \text{NormOS2} (\text{NormOS1} (\text{DerSeq} (s))) = \{ \text{EpsSem} \} \]

by calculation

\[ = \text{OpS} (s). \]

Sub-case (iii): $s = \phi$

\[ \Rightarrow \text{OpS} (s) = \{ \text{PhiSem} \} \]

and

\[ \Rightarrow \text{DerSeq} (s) = \{ \text{PhiSem} \} \]

from clause (iii) of the transition relation

\[ \Rightarrow \text{NormOS1} (\text{DerSeq} (s)) = \{ \text{PhiSem} \} \]

by calculation

\[ \Rightarrow \text{NormOS2} (\text{NormOS1} (\text{DerSeq} (s))) = \{ \text{PhiSem} \} \]

by calculation

\[ = \text{OpS} (s). \]

Hence, the theorem holds for all three of these sub-cases, and so holds for the base case.

**Inductive cases.**

As in the proofs of theorems 53 and 54, the induction hypothesis is that the theorem holds for all $s : \text{SeqConst}$ such that SCC ($s$) < n, for any arbitrary $n > 1$, and so the induction step is to show from this that the theorem must also hold for any $s : \text{SeqConst}$ such that SCC ($s$) = n. Again, therefore, there are two main sub-cases, depending on whether $s$ is constructed by alternation or sequencing.

**Alternation case.**

This case is defined by $s = s1 | s2$, where SCC ($s1$) < n and SCC ($s2$) < n.

In a similar fashion to the corresponding case in the proof of theorem 54, there are four possible sub-cases to consider, but here they arise from the guards on the transitions that give rise to the derivation sequences. Thus, these four cases depend on whether or not $\text{SeqHeads} (s1) = \{ \phi \}$ and whether or not $\text{SeqHeads} (s2) = \{ \phi \}$, and are as follows.

Sub-case (i):

\[ \text{SeqHeads} (s1) = \{ \phi \} \land \text{SeqHeads} (s2) = \{ \phi \} \]

\[ \Rightarrow \text{OpS} (s) = \text{NormOS2} (\text{OpS} (s1) \lor \text{OpS} (s2)) \]
An Operational Semantics for the Dataflow Algebra  A. J. Cowling

\[
= \text{NormOS2} \left( \text{NormOS1} \left( \text{DerSeq} \left( s_1 \right) \right) \cup \text{NormOS2} \left( \text{NormOS1} \left( \text{DerSeq} \left( s_2 \right) \right) \right) \right)
\]

from the induction hypothesis

\[
= \text{NormOS2} \left( \text{NormOS1} \left( \text{DerSeq} \left( s_1 \right) \right) \cup \text{NormOS1} \left( \text{DerSeq} \left( s_2 \right) \right) \right)
\]

from theorem 43

and

\[
\Rightarrow \text{DerSeq} (s) = \{ \text{EmptyTrans} \left( \text{DerSeq} \left( s_1 \right) \right), \text{EmptyTrans} \left( \text{DerSeq} \left( s_2 \right) \right) \}
\]

\[
\Rightarrow \text{NormOS1} \left( \text{DerSeq} \left( s \right) \right) = \text{NormOS1} \left( \text{DerSeq} \left( s_1 \right) \cup \text{DerSeq} \left( s_2 \right) \right)
\]

from theorem 39

\[
\Rightarrow \text{NormOS2} \left( \text{NormOS1} \left( \text{DerSeq} \left( s \right) \right) \right)
\]

\[
= \text{NormOS2} \left( \text{NormOS1} \left( \text{DerSeq} \left( s_1 \right) \right) \cup \text{NormOS1} \left( \text{DerSeq} \left( s_2 \right) \right) \right)
\]

= \text{OpS} (s).

Sub-case (ii):

\[
\text{SeqHeads} \left( s_1 \right) = \{ \phi \} \land \text{SeqHeads} \left( s_2 \right) \neq \{ \phi \} \Rightarrow s_1 = \phi \land \text{PhiSem} \notin \text{OpS} \left( s_2 \right)
\]

\[
\Rightarrow \text{OpS} (s) = \text{NormOS2} \left( \text{OpS} \left( s_1 \right) \cup \text{OpS} \left( s_2 \right) \right)
\]

\[
= \text{NormOS2} \left( \text{NormOS1} \left( \text{DerSeq} \left( s_1 \right) \right) \cup \text{NormOS1} \left( \text{DerSeq} \left( s_2 \right) \right) \right)
\]

from the induction hypothesis

\[
= \text{NormOS2} \left( \text{NormOS1} \left( \text{DerSeq} \left( s_2 \right) \right) \right)
\]

= \text{OpS} (s).

Sub-case (iii):

\[
\text{SeqHeads} \left( s_1 \right) \neq \{ \phi \} \land \text{SeqHeads} \left( s_2 \right) = \{ \phi \}
\]

This is symmetrical with sub-case (ii), and the argument does not need to be repeated in detail.

Sub-case (iv):

\[
\text{SeqHeads} \left( s_1 \right) \neq \{ \phi \} \land \text{SeqHeads} \left( s_2 \right) \neq \{ \phi \}
\]

\[
\Rightarrow \text{OpS} (s) = \text{NormOS2} \left( \text{OpS} \left( s_1 \right) \cup \text{OpS} \left( s_2 \right) \right)
\]

and

\[
\Rightarrow \text{DerSeq} (s) = \{ \text{EmptyTrans} \left( \text{DerSeq} \left( s_1 \right) \right), \text{EmptyTrans} \left( \text{DerSeq} \left( s_2 \right) \right) \}
\]

and so the rest of the argument is identical with that of sub-case (i), and does not need to be repeated here in detail.

Hence, the theorem holds for all four of these sub-cases, and so holds for this alternation sub-case of the inductive case.

**Sequencing case.**

This case is defined by \( s = s_1 ; s_2 \), where SCC (\( s_1 \)) < \( n \) and SCC (\( s_2 \)) < \( n \).

This has three sub-cases, for the possible constructions of \( s_1 \), where the sequencing sub-case requires an inner induction, over SCC (\( s_1 \)). Hence we have what is effectively a double induction, where the inner induction is over a natural number \( n_i \) such that SCC (\( s_1 \)) ≤ \( n_i \), but the structure is slightly more complicated than the other double inductions that have been used in previous proofs. The inner base case is for where \( s_1 \) is an action, so that \( n_i = 1 \), but the two inductive cases are not both defined over the same metric. Thus, there is an inner alternation case, for where \( s_1 \) is constructed as \( s_1a | s_1b \), and the induction for this assumes both that \( n_i \) reduces, since SCC (\( s_1a \)) < SCC (\( s_1 \)) and SCC (\( s_1b \)) < SCC (\( s_1 \)), and also that \( n \) reduces, since the induction hypothesis is applied separately to each of \( s_1a \), \( s_1b \), and \( s_2 \), so that it is actually only the outer induction hypothesis over \( n \) that is used. By contrast, in the inner sequencing case, where \( s_1 \) is constructed as \( s_1a ; s_1b \), the induction again assumes that \( n_i \) reduces, since again SCC (\( s_1a \)) < SCC (\( s_1 \)), but this time that \( n \) stays constant, since (\( s_1a ; s_1b \)); \( s_2 \) is replaced by \( s_1a ; (s_1b ; s_2) \).

**Inner base case.**

For the case where \( s_1 \) is constructed as an action \( a \) there are three sub-cases, as follows.

Sub-case (i): \( s_1 = a \) where \( a \in PA \)

\[
\Rightarrow \text{OpS} (s) = \text{NormOS2} \left( \text{SeqCompOS} \left( \text{OpS} \left( s_1 \right), \text{OpS} \left( s_2 \right) \right) \right)
\]

\[
= \text{NormOS2} \left( \text{SeqCompOS} \left( \{ \text{FinalActSem} (a) \}, \text{OpS} \left( s_2 \right) \right) \right)
\]

= \text{NormOS2} \left( \text{SeqCompOS} \left( \{ \text{FinalActSem} (a) \}, \text{NormOS2} \left( \text{NormOS1} \left( \text{DerSeq} \left( s_2 \right) \right) \right) \right) \right)
\]

from the outer induction hypothesis

\[
= \text{NormOS2} \left( \{ \text{Cont1ActSem} (a, \text{NormOS2} \left( \text{NormOS1} \left( \text{DerSeq} \left( s_2 \right) \right) \right) \} \right)
\]

and

\[
\Rightarrow \text{DerSeq} (s) = \{ \text{ContActSem} (a, \text{DerSeq} \left( s_2 \right)) \}
\]

from clause (iv) of the transition relation

\[
\Rightarrow \text{NormOS1} \left( \text{DerSeq} \left( s \right) \right) = \text{NormOS1} \left( \{ \text{ContActSem} (a, \text{DerSeq} \left( s_2 \right)) \} \right)
\]

\[
\Rightarrow \text{NormOS2} \left( \text{NormOS1} \left( \text{DerSeq} \left( s \right) \right) \right) = \text{NormOS2} \left( \text{NormOS1} \left( \{ \text{ContActSem} (a, \text{DerSeq} \left( s_2 \right)) \} \right) \right)
\]

74
Then to complete the analysis we have three further sub-cases, depending on the construction of \( s_2 \), as follows.

Sub-case (i)(a): \( s_2 = \varepsilon \) \( \Rightarrow \) DerSeq \( (s_2) = \{ \text{EpsSem} \} \)
\( \Rightarrow \) NormOS2 \( (\text{NormOS1 (DerSeq (s_2))} ) = \{ \text{EpsSem} \} \)
and \( \Rightarrow \) NormOS1 \( (\{ \text{ContActSem (a, DerSeq (s_2))} \} = \{ \text{FinalActSem (a)} \} \)
\( \Rightarrow \) NormOS2 \( (\text{NormOS1 (DerSeq (s_2))} ) = \text{NormOS2 (\{ FinalActSem (a) \})} \)
\( = \text{OpS (s)}. \)

Sub-case (i)(b): \( s_2 = \phi \) \( \Rightarrow \) DerSeq \( (s_2) = \{ \text{PhiSem} \} \)
\( \Rightarrow \) NormOS2 \( (\text{NormOS1 (DerSeq (s_2))} ) = \{ \text{PhiSem} \} \)
and \( \Rightarrow \) NormOS1 \( (\{ \text{ContActSem (a, DerSeq (s_2))} \} = \{ \text{FinalAbActSem (a)} \} \)
\( \Rightarrow \) NormOS2 \( (\text{NormOS1 (DerSeq (s_2))} ) = \text{NormOS2 (\{ FinalAbActSem (a) \})} \)
\( = \text{OpS (s)}. \)

Sub-case (i)(c): Otherwise, \text{Cont1ActSem (a, DerSeq (s_2))} \( = \text{ContActSem (a, DerSeq (s_2))} \)
\( \Rightarrow \) OpS \( (s) = \text{NormOS2 (\{ ContActSem (a, NormOS2 (\text{NormOS1 (DerSeq (s_2))} )) \})} \)
\( = \{ \text{ContActSem (a, NormOS2 (\text{NormOS1 (DerSeq (s_2))} ))} \} \) from theorem 28
and \( \Rightarrow \) NormOS2 \( (\text{NormOS1 (DerSeq (s_2))} ) = \text{NormOS2 (\{ ContActSem (a, NormOS1 (DerSeq (s_2)) \})} \)
\( = \{ \text{ContActSem (a, NormOS2 (NormOS1 (DerSeq (s_2)) ))} \} \)
\( = \text{OpS (s)}. \)

Sub-case (ii): \( s_1 = \varepsilon \)
\( \Rightarrow \) OpS \( (s) = \text{NormOS2 (SeqCompOS (OpS (s_1), OpS (s_2))} \)
\( = \text{NormOS2 (SeqCompOS (\{ EpsSem \}, OpS (s_2))} \)
\( = \text{NormOS2 (OpS (s_2))} \) from theorem 51
\( = \text{NormOS2 (NormOS1 (DerSeq (s_2))} \) from the outer induction hypothesis
\( = \text{NormOS2 (NormOS1 (DerSeq (s_2))} \) from theorem 28
and \( \Rightarrow \) DerSeq \( (s) = \{ \text{EmptyTrans (DerSeq (s_2))} \) from clause (v) of the transition relation
\( \Rightarrow \) NormOS1 \( (\text{DerSeq (s)}) \) \( = \text{NormOS1 (\text{DerSeq (s_2))} \)
\( \Rightarrow \) NormOS2 \( (\text{NormOS1 (DerSeq (s))} ) = \text{NormOS2 (NormOS1 (DerSeq (s_2))} \)
\( = \text{OpS (s)}. \)

Sub-case (iii): \( s_1 = \phi \)
\( \Rightarrow \) OpS \( (s) = \text{NormOS2 (SeqCompOS (OpS (s_1), OpS (s_2))} \)
\( = \text{NormOS2 (SeqCompOS (\{ PhiSem \}, OpS (s_2))} \)
\( = \text{NormOS2 (\{ PhiSem \})} \)
\( = \{ \text{PhiSem} \} \) from clause (iii) of the transition relation
and \( \Rightarrow \) DerSeq \( (s) = \{ \text{PhiSem} \) from clause (iii) of the transition relation
\( \Rightarrow \) NormOS1 \( (\text{DerSeq (s)}) \) \( = \{ \text{PhiSem} \)
\( \Rightarrow \) NormOS2 \( (\text{NormOS1 (DerSeq (s))} ) = \{ \text{PhiSem} \) from theorem 48
\( = \text{OpS (s)}. \)

Inner alternation case.

This case is defined by \( s_1 = s_{1a} \mid s_{1b} \), which gives:
\( \text{OpS (s)} = \text{NormOS2 (SeqCompOS (OpS (s_1), OpS (s_2))} \)
\( = \text{NormOS2 (SeqCompOS (\{ EpsSem \} , OpS (s_2))} \)
\( = \text{NormOS2 (\{ EpsSem \})} \) from theorem 48
\( = \text{NormOS2 (\{ PhiSem \})} \)
\( = \{ \text{PhiSem} \} \) from theorem 43
This case also gives
\[
\text{DerSeq} (s) = \{ \text{EmptyTrans} (\text{DerSeq} \left( (s_1a ; s_2) \mid (s_1b ; s_2) \right)) \}
\]
from clause (viii) of the transition relation, which then gives four further sub-cases, depending on whether or not \(\text{SeqHeads} (s1a ; s2) = \{ \phi \} \) and whether or not \(\text{SeqHeads} (s1b ; s2) = \{ \phi \} \), as follows, where we will take the cases in the opposite order from the outer alternation case described above.

Sub-case (i): \(\text{SeqHeads} (s1a ; s2) \neq \{ \phi \} \land \text{SeqHeads} (s1b ; s2) \neq \{ \phi \} \)
\[
\Rightarrow \text{DerSeq} (s) = \emptyset
\]
\[
\Rightarrow \text{NormOS1} (\text{DerSeq} (s)) = \text{NormOS1} (\text{DerSeq} (s1a ; s2) \cup \text{DerSeq} (s1b ; s2))
\]
from theorem 39
\[
\Rightarrow \text{NormOS2} (\text{NormOS1} (\text{DerSeq} (s)))
\]
\[
= \text{NormOS2} (\text{NormOS1} (\text{NormOS2} (\text{NormOS1} (\text{DerSeq} (s1a ; s2)))) \cup \text{NormOS2} (\text{NormOS2} (\text{NormOS1} (\text{DerSeq} (s1b ; s2)))))
\]
from theorem 43
\[
= \text{NormOS2} (\text{OpS} (s1a ; s2) \cup \text{OpS} (s1b ; s2))
\]
from the outer induction hypothesis
\[
= \text{OpS} (s1a ; s2) \mid (s1b ; s2)
\]
sub-case (i)
\[
\Rightarrow \text{NormOS2} (\text{NormOS1} (\text{DerSeq} (s))) = \text{NormOS2} (\text{NormOS1} (\text{DerSeq} (s1a ; s2)) \cup \text{NormOS2} (\text{NormOS2} (\text{NormOS1} (\text{DerSeq} (s1b ; s2)))))
\]
from the outer induction hypothesis
\[
= \text{OpS} (s1a ; s2) \mid (s1b ; s2)
\]
= \text{OpS} (s).

Sub-case (ii): \(\text{SeqHeads} (s1a ; s2) \neq \{ \phi \} \land \text{SeqHeads} (s1b ; s2) = \{ \phi \} \)
\[
\Rightarrow \text{DerSeq} (s) = \emptyset
\]
\[
\Rightarrow \text{NormOS1} (\text{DerSeq} (s)) = \text{NormOS1} (\text{DerSeq} (s1a ; s2))
\]
\[
\Rightarrow \text{NormOS2} (\text{NormOS1} (\text{DerSeq} (s))) = \text{NormOS2} (\text{NormOS1} (\text{DerSeq} (s1a ; s2)))
\]
But, \(\text{SeqHeads} (s1b ; s2) = \{ \phi \} \Rightarrow (s1b ; s2) = \phi \Rightarrow (s1a ; s2) | (s1b ; s2) = (s1a ; s2) | \phi = (s1a ; s2) \)
\[
\Rightarrow \text{OpS} ((s1a ; s2) | (s1b ; s2)) = \text{OpS} (s1a ; s2).
\]
Hence, \(\text{NormOS2} (\text{NormOS1} (\text{DerSeq} (s))) = \text{OpS} (s).\)

Sub-case (iii): \(\text{SeqHeads} (s1a ; s2) = \{ \phi \} \land \text{SeqHeads} (s1b ; s2) \neq \{ \phi \} \)
This is symmetric with sub-case (ii), and the argument does not need to be repeated here in detail.

Sub-case (iv): \(\text{SeqHeads} (s1a ; s2) = \{ \phi \} \land \text{SeqHeads} (s1b ; s2) = \{ \phi \} \)
As in the outer alternation case above, this gives the same expression for \(\text{DerSeq} (s)\) as in sub-case (i), and so the analysis is identical, and does not need to be repeated here.

Hence, the theorem holds for all four of these sub-cases, and so holds for this inner alternation case too.

**Inner sequencing case.**

This case is defined by \(s1 = s1a ; s1b\), which gives:
\[
\text{OpS} (s) = \text{NormOS2} (\text{SeqCompOS} (\text{OpS} (s1), \text{OpS} (s2)))
\]
\[
= \text{NormOS2} (\text{SeqCompOS} (\text{NormOS2} (\text{SeqCompOS} (\text{OpS} (s1a), \text{OpS} (s1b))))), \text{OpS} (s2)))
\]
\[
= \text{NormOS2} (\text{SeqCompOS} (\text{NormOS2} (\text{SeqCompOS} (\text{NormOS2} (\text{NormOS1} (\text{DerSeq} (s1a)))),
\text{NormOS2} (\text{NormOS1} (\text{DerSeq} (s1b))))), \text{NormOS2} (\text{NormOS1} (\text{DerSeq} (s2))))
\]
from the outer induction hypothesis
\[
= \text{NormOS2} (\text{SeqCompOS} (\text{NormOS2} (\text{NormOS1} (\text{DerSeq} (s1a))))) \text{NormOS2} (\text{SeqCompOS} (\text{NormOS2} (\text{NormOS1} (\text{DerSeq} (s1b)))),
\text{NormOS2} (\text{NormOS1} (\text{DerSeq} (s2))))
\]
from theorem 50
\[
= \text{NormOS2} (\text{SeqCompOS} (\text{OpS} (s1a), \text{NormOS2} (\text{SeqCompOS} (\text{OpS} (s1b), \text{OpS} (s2))))
\]
from the outer induction hypothesis
\[
= \text{NormOS2} (\text{SeqCompOS} (\text{OpS} (s1a), \text{OpS} (s1b ; s2)))
\]
\[
= \text{OpS} (s1a ; (s1b ; s2))
\]
and \(\Rightarrow \text{DerSeq} (s) = \emptyset \) from clause (vii) of the transition relation
\[
\Rightarrow \text{NormOS1} (\text{DerSeq} (s)) = \text{NormOS1} (\text{DerSeq} (s1a ; (s1b ; s2))))
\]
\[
= \text{OpS} (s1a ; (s1b ; s2))
\]
from the inner induction hypothesis
\[
= \text{OpS} (s).
\]
Hence, the theorem holds for this inner inductive case, and so by the inner induction it holds for all sequences $s_1$ constructed as $s_1 = s_1a \cdot s_1b$, for all values of $n$. Hence, by the outer induction it also holds for the other possible constructions of $s_1$, and hence holds for all sequences $s$ constructed as $s = s_1 \cdot s_2$, for all values of $n$, so that the outer inductive case follows from the combination of the alternation and sequencing sub-cases. Therefore, by induction the theorem holds for all values of $n$, and so holds.

10. Consistency of the Semantics

Having defined the semantic functions and their properties, the next stage is to use them to prove the consistency of the semantics with the axioms of the DFA (viz axioms (i) to (ix) from section 6 of [4]), a property that is more usually known as the soundness of the axioms with respect to the semantics. This consistency is expressed as the set of theorems in this section, with one for each of the axioms. The theorems and their proofs are presented in the same order as the axioms appear in [4], and to make the presentation of these results clearer, the same convention as in [4] is adopted for the theorems in this section, namely that elements of $\text{SeqConst}$ are shown in bold face. This also ensures that occurrences of the symbol $|$ that denote the DFA operation of alternation can be distinguished from those that form part of a qualifier.

Theorem 56.
$\forall s_1, s_2, s_3 \in \text{SeqConst} \quad \text{OpS} \left( s_1 ; (s_2 ; s_3) \right) = \text{OpS} \left( (s_1 ; s_2) ; s_3 \right)$.

Proof.
Let $os_1, os_2, os_3 : \text{OpSem}$ be defined as $os_1 = \text{OpS} \left( s_1 \right), os_2 = \text{OpS} \left( s_2 \right)$ and $os_3 = \text{OpS} \left( s_3 \right)$. Then,
$\text{OpS} \left( s_2 ; s_3 \right) = \text{NormOS2} \left( \text{SeqCompOS} \left( os_2, os_3 \right) \right)$
$\text{OpS} \left( s_1 ; (s_2 ; s_3) \right) = \text{NormOS2} \left( \text{SeqCompOS} \left( os_1, \text{NormOS2} \left( \text{SeqCompOS} \left( os_2, os_3 \right) \right) \right) \right)$
And
$\text{OpS} \left( (s_1 ; s_2) ; s_3 \right) = \text{NormOS2} \left( \text{SeqCompOS} \left( \text{NormOS2} \left( \text{SeqCompOS} \left( os_1, \text{NormOS2} \left( \text{SeqCompOS} \left( os_2, os_3 \right) \right) \right) \right) \right) \right)$

= $\text{OpS} \left( s_1 ; (s_2 ; s_3) \right)$ from theorem 49

Theorem 57.
$\forall s \in \text{SeqConst} \quad \text{OpS} \left( s ; \epsilon \right) = \text{OpS} \left( s \right) = \text{OpS} \left( \epsilon ; s \right)$.

Proof.
The proof relies in part on the observation that $\text{OSNorm2} \left( \text{OpS} \left( s \right) \right)$ holds, from theorem 53. Then
$\text{OpS} \left( s ; \epsilon \right) = \text{NormOS2} \left( \text{SeqCompOS} \left( \text{OpS} \left( s \right), \left\{ \text{EpsSem} \right\} \right) \right)$

= $\text{NormOS2} \left( \text{OpS} \left( s \right) \right)$ from theorem 52

= $\text{OpS} \left( s \right)$ from theorem 28

And
$\text{OpS} \left( \epsilon ; s \right) = \text{NormOS2} \left( \text{SeqCompOS} \left( \left\{ \text{EpsSem} \right\}, \text{OpS} \left( s \right) \right) \right)$

= $\text{NormOS2} \left( \text{OpS} \left( s \right) \right)$ from theorem 51

= $\text{OpS} \left( s \right)$ from theorem 28

Theorem 58.
$\forall s_1, s_2, s_3 \in \text{SeqConst} \quad \text{OpS} \left( s_1 \mid (s_2 \mid s_3) \right) = \text{OpS} \left( (s_1 \mid s_2) \mid s_3 \right)$.

Proof.
Let $os_1, os_2, os_3 : \text{OpSem}$ be defined as $os_1 = \text{OpS} \left( s_1 \right), os_2 = \text{OpS} \left( s_2 \right)$ and $os_3 = \text{OpS} \left( s_3 \right)$. Then,
$\text{OpS} \left( s_2 \mid s_3 \right) = \text{NormOS2} \left( os_2 \cup os_3 \right)$
$\text{OpS} \left( s_1 \mid (s_2 \mid s_3) \right) = \text{NormOS2} \left( os_1 \cup \text{NormOS2} \left( os_2 \cup os_3 \right) \right)$

= $\text{NormOS2} \left( os_1 \cup os_2 \cup os_3 \right)$ from theorem 42

And
$\text{OpS} \left( s_1 \mid s_2 \right) = \text{NormOS2} \left( os_1 \cup os_2 \right)$
$\text{OpS} \left( (s_1 \mid s_2) \mid s_3 \right) = \text{NormOS2} \left( \text{NormOS2} \left( os_1 \cup os_2 \right) \cup os_3 \right)$

= $\text{OpS} \left( s_1 \mid (s_2 \mid s_3) \right)$ from theorem 41

Theorem 59.
$\forall s_1, s_2 \in \text{SeqConst} \quad \text{OpS} \left( s_1 \mid s_2 \right) = \text{OpS} \left( s_2 \mid s_1 \right)$.
Proof.
\[
\text{OpS}(s_1 | s_2) = \text{NormOS2}(\text{OpS}(s_1) \cup \text{OpS}(s_2)) \\
= \text{NormOS2}(\text{OpS}(s_2) \cup \text{OpS}(s_1)) \\
= \text{OpS}(s_2 | s_1)
\]

Theorem 60.
\[\forall s \in \text{SeqConst} \bullet \text{OpS}(s | s) = \text{OpS}(s)\]

Proof.
The proof relies in part on the observation that \(\text{OSNorm2}(\text{OpS}(s))\) holds, from theorem 53. Then
\[
\text{OpS}(s | s) = \text{NormOS2}(\text{OpS}(s) \cup \text{OpS}(s)) \\
= \text{NormOS2}(\text{OpS}(s)) \\
= \text{OpS}(s)
\]
from theorem 28

Theorem 61.
\[\forall s \in \text{SeqConst} \bullet \text{OpS}(s | \phi) = \text{OpS}(\phi | s) = \text{OpS}(s)\]

Proof.
Again, the proof relies in part on the observation that \(\text{OSNorm2}(\text{OpS}(s))\) holds, from theorem 53. Then
\[
\text{OpS}(s | \phi) = \text{NormOS2}((\text{PhiSem}) \cup \text{OpS}(s)) \\
= \text{NormOS2}(\text{OpS}(s)) \\
= \text{OpS}(s)
\]
from theorem 28

Theorem 62.
\[\forall s_1, s_2, s_3 \in \text{SeqConst} \bullet \text{OpS}(s_1 ; (s_2 | s_3)) = \text{OpS}((s_1 ; s_2) | (s_1 ; s_3))\]

Proof.
Let \(os_1, os_2, os_3 : \text{OpSem}\) be defined as \(os_1 = \text{OpS}(s_1), os_2 = \text{OpS}(s_2)\) and \(os_3 = \text{OpS}(s_3)\). Then,
\[
\text{OpS}(s_2 | s_3) = \text{NormOS2}(os_2 \cup os_3) \\
\text{OpS}(s_1 ; (s_2 | s_3)) = \text{NormOS2}((\text{SeqCompOS}(os_1, \text{NormOS2}(os_2 \cup os_3)))\) \\
And \[
\text{OpS}(s_1 ; s_2) = \text{NormOS2}(\text{SeqCompOS}(os_1, os_2)) \\
\text{OpS}(s_1 ; s_3) = \text{NormOS2}(\text{SeqCompOS}(os_1, os_3)) \\
\text{OpS}((s_1 ; s_2) | (s_1 ; s_3)) \\
= \text{NormOS2}((\text{SeqCompOS}(os_1, os_2)) \cup \text{NormOS2}((\text{SeqCompOS}(os_1, os_3))) \\
= \text{NormOS2}((\text{SeqCompOS}(os_1, os_2) \cup \text{SeqCompOS}(os_1, os_3))) \\
= \text{NormOS2}(\text{SeqCompOS}(os_1, \text{NormOS2}(os_2 \cup os_3))) \\
= \text{OpS}(s_1 ; (s_2 | s_3))
\]
from theorem 43

Theorem 63.
\[\forall s_1, s_2, s_3 \in \text{SeqConst} \bullet \text{OpS}((s_1 | s_2) ; s_3) = \text{OpS}((s_1 ; s_3) | (s_2 ; s_3))\]

Proof.
Let \(os_1, os_2, os_3 : \text{OpSem}\) be defined as \(os_1 = \text{OpS}(s_1), os_2 = \text{OpS}(s_2)\) and \(os_3 = \text{OpS}(s_3)\). Then,
\[
\text{OpS}(s_1 | s_2) = \text{NormOS2}(os_1 \cup os_2) \\
\text{OpS}((s_1 | s_2) ; s_3) = \text{NormOS2}((\text{SeqCompOS}(os_1, \text{NormOS2}(os_1 \cup os_2), os_3)) \\
And \[
\text{OpS}(s_1 ; s_3) = \text{NormOS2}(\text{SeqCompOS}(os_1, os_3)) \\
\text{OpS}(s_2 ; s_3) = \text{NormOS2}(\text{SeqCompOS}(os_2, os_3)) \\
\text{OpS}((s_1 ; s_3) | (s_2 ; s_3)) \\
= \text{NormOS2}((\text{SeqCompOS}(os_1, os_3)) \cup \text{NormOS2}((\text{SeqCompOS}(os_2, os_3))) \\
= \text{NormOS2}((\text{SeqCompOS}(os_1, os_3) \cup \text{SeqCompOS}(os_2, os_3))) \\
= \text{NormOS2}(\text{SeqCompOS}(os_1, \text{SeqCompOS}(os_2, os_3))) \\
= \text{OpS}((s_1 | s_2) ; s_3)
\]
from theorem 43

And
Theorem 64.
\[ \forall s \in \text{SeqConst} \land \text{OpS}(\phi; s) = \text{OpS}(\phi). \]

Proof.
The proof relies in part on the observation that \( \text{OSNorm2}(\{\text{PhiSem}\}) \) holds, since \( \text{OUNorm2}(\text{PhiSem}) \) holds. Then
\[
\begin{align*}
\text{OpS}(\phi; s) &= \text{NormOS2}(\text{SeqCompOS}(\{\text{PhiSem}\}, \text{OpS}(s))) \\
&= \text{NormOS2}(\{\text{PhiSem}\}) \\
&= \{\text{PhiSem}\} \\
&= \text{OpS}(\phi)
\end{align*}
\]
from theorem 28

Having thus established the soundness of the individual axioms, the results of theorems 56 to 64 can be collected together as follows.

Theorem 65.
\[ \forall sa, sb \in \text{SeqConst} \mid sa = sb \text{ as a consequence of a single application of any of axioms (i) to (ix) of section 4}, \]
\[ \land \text{OpS}(sa) = \text{OpS}(sb). \]

Proof.
The proof is direct from the application of whichever of theorems 56 to 64 corresponds to the relevant axiom.

Theorem 66.
\[ \forall sa, sb \in \text{SeqConst} \mid sa = sb \land \text{OpS}(sa) = \text{OpS}(sb). \]

Proof.
The proof is direct from the property that equality of the sequence constructions results from the transitive closure of applications of axioms (i) to (ix) of section 6 of [4]. Equality of the semantics then results from the corresponding transitive closure of applications of theorem 65.

This therefore establishes the soundness of the set of axioms with respect to the operational semantics, although (as with the denotational semantics defined in [4]), the fact that the axioms had been defined first and then the semantics constructed to reflect them means that in practice it is probably more appropriate to regard this result as establishing the consistency of the semantics with the axioms.

11. Completeness of the Algebra

As in [4], the next step after showing the consistency of the semantics with the axioms is to show that the algebra is complete with respect to the semantics, so that for any valid construction in the semantics there is an equivalent construction in the DFA. To define this, we take the same constructive approach as in [4], of constructing an inverse semantic function, which for any non-empty forest structure produces an equivalent construction in the DFA. Here the forest structure might normally be expected to be in second normal form, but in practice the definition of the function only needs to assume that it is in zeroth and first strict normal forms. Also, as in [4], it is necessary to exclude the case of the empty forest, since this would have to be regarded as the semantics of an empty DFA specification, and this (which is not the same thing as the silent action) is not a valid element of \( \text{SeqConst} \).

The function has to be defined recursively over a forest, and since such a recursion depends at each stage on selecting an arbitrary tree from the forest it would essentially be non-deterministic if it were defined as producing elements of \( \text{SeqConst} \). These different possible elements of \( \text{SeqConst} \) are, however, equal under the axioms of the algebra, and so this problem could be avoided in two ways. One way would be to define the function formally as producing elements of \( \text{Seq} \) instead of \( \text{SeqConst} \), in which case it would be deterministic, but in principle it would not formally be producing a result of the appropriate type for subsequent applications of \( \text{OpS} \), although in practice we could (as in [4]) rely on the coercion of selection to map elements of \( \text{Seq} \) into appropriate elements of \( \text{SeqConst} \). The other way would be to ignore the inherent non-determinism, which is effectively what is done in [4], where no mention is made of the fact that the inverse semantic function \( \text{InvSem} \) (which has signature \( \text{SeqSem} \rightarrow \text{SeqConst} \)) is defined in terms of a function \( \text{InvSemA} \), whose definition is inherently non-deterministic in the same way. Given that handling the feature of non-determinism correctly is an important aspect of the operational semantics, the latter approach seems inherently unsatisfactory, and so here the inverse semantic function will be defined in terms of producing elements of \( \text{Seq} \).
As for many of the operations that have been introduced, to define this recursion requires a pair of functions, one that operates over elements of $\text{OpSem}$ and the other that operates over elements of $\text{OpSUnit}$. These two functions are therefore called $\text{OSToSeq}$ and $\text{OUToSeq}$, and they have signatures $\text{OpSem} \rightarrow \text{Seq}$ and $\text{OpSUnit} \rightarrow \text{Seq}$ respectively. They are defined as follows, where $\text{OSToSeq}$ requires the precondition that its parameter is not the empty forest. Here though, instead of the bold face used in the previous section for constructions in $\text{SeqConst}$, square brackets $[]$ are used to enclose constructions in $\text{Seq}$, so that inside them the symbol $|$ denotes the alternation operator, rather than part of a qualifier.

\[
\begin{align*}
\text{os} \neq \emptyset \Rightarrow \text{OSToSeq}(\text{os}) & \equiv \text{if #os} = 1 \\
& \text{then } \exists \text{ou} : \text{OpSUnit} \mid \text{os} = \{ \text{ou} \} \bullet \text{OUToSeq}(\text{ou}) \\
& \text{else } \exists \text{ou} : \text{OpSUnit} \mid \text{ou} \in \text{os} \bullet [ \text{OUToSeq}(\text{ou}) ] \text{OSToSeq}(\text{os} - \{ \text{ou} \}) ] \\
& \text{fi}
\end{align*}
\]

\[
\begin{align*}
\text{OUToSeq}(\text{ou}) & \equiv \text{if ou}.\text{DoesAct} \\
& \text{then if ou}.NextState = \text{continues} \\
& \text{then } [ \text{ou}.\text{TheAct} ; \text{OSToSeq}(\text{ou}.\text{Rest}) ] \\
& \text{elsif ou}.NextState = \text{normend} \\
& \text{then } [ \text{ou}.\text{TheAct} ] \\
& \text{else } [ \text{ou}.\text{TheAct} ; \phi ] \\
& \text{fi}
\end{align*}
\]

\[
\begin{align*}
\text{elsif ou}.NextState = \text{continues} \\
& \text{then } [ \text{OUToSeq}(\text{ou}.\text{Rest}) ] \\
& \text{elsif ou}.NextState = \text{normend} \\
& \text{then } [ \epsilon ] \\
& \text{else } [ \phi ] \\
& \text{fi}
\end{align*}
\]

In formulating this definition of $\text{OSToSeq}$ we have avoided using the derived DFA operator $U$, in order to highlight the non-determinism, but it should be apparent that it could be used to rewrite this definition in the form

\[
\begin{align*}
\text{os} \neq \emptyset \Rightarrow \text{OSToSeq}(\text{os}) & \equiv \bigcup_{\text{ou} \in \text{os}} \text{OUToSeq}(\text{ou})
\end{align*}
\]

and this form will be used in the proofs of the required properties for this function.

There are then two key results that are required for this function. The one that establishes the formal property that the algebra is complete with respect to the semantics is the following theorem, which implicitly relies on the selection coercion to allow $\text{OpS}$ to be applied to objects constructed by $\text{OSToSeq}$. As in the definitions above, the convention is used here and in the following theorems in this section that expressions in $\text{Seq}$ (meaning ones that involve DFA operators) are denoted inside square brackets, to make the interpretation of the symbol $|$ unambiguous.

**Theorem 67.**

\[
\forall \text{os} : \text{OpSem} \mid \text{os} \neq \emptyset \land \text{OSNorm012}(\text{os}) \bullet \text{OpS}(\text{OSToSeq}(\text{os})) = \text{os}.
\]

**Proof.**

The proof here is by a double induction, with the outer induction being over the height of the forest $\text{os}$ and the inner one over its cardinality. For both of these, the base and inductive cases involve analysing an arbitrary element $\text{ou} : \text{OpSUnit}$ of $\text{os}$, to show that $\text{OpS}(\text{OUToSeq}(\text{ou})) = \{ \text{ou} \}$, and the role of the inner induction is to show that the result of the theorem then follows from this for any set of such elements.

**Outer Base Case.**

The outer base case is that $\text{HeightOS}(\text{os}) = 1$, meaning that (from theorem 5) there are just four possible constructions of $\text{ou}$ that need to be analysed, as follows.

**Inner Base Case.**

The inner base case is that $\# \text{os} = 1$, so that the analysis of the four possible constructions for the singleton set is as follows.

\[
\begin{align*}
\text{(i) } \text{os} = \{ \text{ou} \} \land \text{ou} = \text{EpsSem} \\
\Rightarrow \text{OpS}(\text{OSToSeq}(\text{os})) & = \text{OpS}(\text{OUToSeq}(\text{EpsSem})) \\
& = \text{OpS}( [ \epsilon ] ) = ( \text{EpsSem} ) = \text{os}.
\end{align*}
\]
(ii) \(os = \{ ou \} \land ou = \text{PhiSem}\) \\
\Rightarrow OpS (OSToSeq (os)) = OpS (OUToSeq (PhiSem)) \\
= OpS (\{ \phi \}) = \{ \text{PhiSem} \} = os.

(iii) \(os = \{ ou \} \land \exists a : \text{PA} \cdot ou = \text{FinalActSem} (a)\) \\
\Rightarrow OpS (OSToSeq (os)) = OpS (OUToSeq (FinalActSem (a))) \\
= OpS (\{ a \}) = \{ \text{FinalActSem} (a) \} = os.

(iv) \(os = \{ ou \} \land \exists a : \text{PA} \cdot ou = \text{FinalAbActSem} (a)\) \\
\Rightarrow OpS (OSToSeq (os)) = OpS (OUToSeq (FinalAbActSem (a))) \\
= OpS (\{ a \}) = \{ \text{FinalAbActSem} (a) \} \\
= \{ \text{FinalAbActSem} (a) \} \text{ from theorems } 10 \text{ & } 28 \\
= os.

Hence the theorem holds for all possible constructions of os, and so holds for the inner base case.

**Inner Inductive Case.**

The inner inductive case is that \(# os > 1\), and so the induction hypothesis is that, for any arbitrary natural number \(ni > 1\), the theorem holds for any os such that \(# os < ni\), and the induction step is to show that therefore it also holds for any os such that \(# os = ni\). For this case, we let os’ : OpSem be any object such that \(# os’ = ni - 1\), and such that os can be written as \(os = \{ ou \} \cup os’\). Then we have:

\[ OpS (OSToSeq (os)) = OpS (OSToSeq (\{ ou \} \cup os’)) = OpS (\{ ou \} \cup os’) \]

\[ = \text{NormOS2} (\{ ou \} \cup os’) \text{ from the inner induction hypothesis & base case} \]

\[ = \text{NormOS2} (os) \text{ from theorem } 28, \text{ since OSNorm2 (os)}. \]

Hence the theorem holds for this inner inductive case, and so by induction over \(ni\) it holds for all \(os\) meeting the conditions of the outer base case, which therefore holds.

**Outer Inductive Case.**

The outer inductive case is that \(\text{HeightOS} (os) > 1\), and so the induction hypothesis is that, for any arbitrary natural number \(no > 1\), the theorem holds for any os such that \(\text{HeightOS} (os) < no\), and the induction step is to show that therefore it also holds for any os such that \(\text{HeightOS} (os) = no\). This case introduces one additional construction for the element ou, in addition to those which have already been analysed as part of the outer base case.

**Inner Base Case.**

Again, the inner base case is that \(# os = 1\), and the analysis of the additional possible construction for the singleton set is as follows.

\[ os = \{ ou \} \land \exists a : \text{PA}, os1 : \text{OpSem} \mid \text{IsActive} (os1) \cdot ou = \text{ContActSem} (a, os1) \]

\[ \Rightarrow OpS (OSToSeq (os)) = OpS (OUToSeq (ContActSem (a, os1))) = OpS (\{ a \}) = \{ \text{Cont1ActSem} (a, os1) \}) \]

\[ = \text{NormOS2} (\{ Cont1ActSem (a, os1) \} \cup \text{OpS (OSToSeq (os1))}) \]

\[ = \text{NormOS2} (\{ Cont1ActSem (a, os1)\}) \text{ from the outer induction hypothesis} \]

\[ = \text{NormOS2} (os) \text{ since IsActive (os1)} \]

\[ = os \text{ from theorem } 28, \text{ since OSNorm2 (os)}. \]

Hence the theorem also holds for the inner base case of this outer inductive case.

**Inner Inductive Case.**

The inner inductive case here is the same as for the outer base case, and so the induction hypothesis and the induction step carry across unchanged, and do not need to be repeated here. Also, since the analysis does not depend in any formal way on the heights of the forest os’ or the element ou, it does not need to be repeated here, but what does need to be noted is that, where this analysis uses the inner base case and induction hypothesis, then here these are actually applying to different sets of objects than for the outer base case, as the constraints on the heights of the objects are different.
Similarly, in concluding that the theorem holds for this inner inductive case, we are here concluding from this by induction that therefore it holds for all \( n \), and so holds for all \( o_s \) meeting the conditions of the outer inductive case, which therefore holds. From this in turn we can therefore conclude by induction that the theorem holds for all \( n_0 \), and so holds for all forests of any height, and therefore holds.

As well as this result, a second useful property for the inverse semantic function is that, for any DFA sequence, if we apply the semantic function and then apply its inverse, we get back to a sequence that is equal to the original. Before this can be established, though, there are various intermediate properties that need to be shown, of which the first is expressed as the following theorem.

**Theorem 68.**
\[
\forall o_s1, o_s2 : \text{OpSem} \mid o_s1 \neq \emptyset \land \text{OSNorm01} (o_s1) \land o_s2 \neq \emptyset \land \text{OSNorm01} (o_s2) \Rightarrow
\text{OSToSeq} (o_s1 \cup o_s2) = [\text{OSToSeq} (o_s1) \mid \text{OSToSeq} (o_s2)].
\]

**Proof.**
\[
\text{OSToSeq} (o_s1) \cup \text{OSToSeq} (o_s2) = [\bigcup_{o_u \in o_s1} \text{OUToSeq} (o_u) \mid \bigcup_{o_u \in o_s2} \text{OUToSeq} (o_u)] = \text{OSToSeq} (o_s1 \cup o_s2).
\]

The next intermediate result defines a similar distributive property, but for \( \text{OUToSeq} \) over the operation \( \text{MergeOU} \), and it is expressed as the following theorem.

**Theorem 69.**
\[
\forall o_u1, o_u2 : \text{OpSUnit} \mid \text{OUNorm01} (o_u1) \land \text{OUNorm01} (o_u2) \land o_u1 \neq o_u2 \land \text{GetHead} (o_u1) = \text{GetHead} (o_u2) \Rightarrow
\text{OUToSeq} (\text{MergeOU} (o_u1, o_u2)) = [\text{OUToSeq} (o_u1) \mid \text{OUToSeq} (o_u2)].
\]

**Proof.**
The proof is by induction over the maximum height of the two objects \( o_u1 \) and \( o_u2 \).

**Base case.**
The base case is that \( \text{HeightOU} (o_u1) = 1 \land \text{HeightOU} (o_u2) = 1 \), meaning that the only possible constructions for \( o_u1 \) and \( o_u2 \) that satisfy the conditions of the theorem are \( \text{FinalActSem} (a) \) and \( \text{FinalAbActSem} (a) \). Since \( \text{MergeOU} \) is commutative, from theorem 19, then it is sufficient to analyse this as the following single case:

\[
o_u1 = \text{FinalActSem} (a) \Rightarrow \text{OUToSeq} (\text{MergeOU} (o_u1, o_u2)) = [a]
\]

and

\[
\Rightarrow [\text{OUToSeq} (o_u1) \mid \text{OUToSeq} (o_u2)] = [(a ; (a ; \phi)) = [a] = \text{OUToSeq} (\text{MergeOU} (o_u1, o_u2)).
\]

**Inductive case.**
The induction hypothesis is that the theorem holds for all \( o_u1 \) and \( o_u2 \) such that, for any arbitrary natural number \( n > 1 \), \( \text{HeightOU} (o_u1) < n \land \text{HeightOU} (o_u2) < n \). The induction step is then to show from this that the theorem must also hold for any \( o_u1 \) and \( o_u2 \) such that \( \text{max} (\text{HeightOU} (o_u1), \text{HeightOU} (o_u2)) = n \).

Because of the commutative property of \( \text{MergeOU} \), we can assume without loss of generality that \( \text{HeightOU} (o_u1) = n \), so that its construction must be

\[
\exists o_s1 : \text{OpSem} \mid \text{IsActive} (o_s1) \Rightarrow o_u1 = \text{ContActSem} (a, o_s1)
\]

and there are then three possible cases to be analysed for the construction of \( o_u2 \), as follows.

(i) \[
\exists o_s2 : \text{OpSem} \mid \text{IsActive} (o_s2) \Rightarrow o_u2 = \text{ContActSem} (a, o_s2)
\]

⇒ \( \text{MergeOU} (o_u1, o_u2) = \text{ContActSem} (a, o_s1 \cup o_s2) \)

⇒ \( \text{OUToSeq} (\text{MergeOU} (o_u1, o_u2)) = \text{OUToSeq} (\text{ContActSem} (a, o_s1 \cup o_s2)) \)

= [a ; OSToSeq (o_s1 \cup o_s2)]

and

⇒ \( [\text{OUToSeq} (o_u1) \mid \text{OUToSeq} (o_u2)] = [(a ; \text{OSToSeq} (o_s1)) \mid (a ; \text{OSToSeq} (o_s2))] = [a ; (\text{OSToSeq} (o_s1) \mid \text{OSToSeq} (o_s2))] \) from theorem 68

= \( \text{OUToSeq} (\text{MergeOU} (o_u1, o_u2)). \)

(ii) \[
o_u2 = \text{FinalActSem} (a)
\]

⇒ \( \text{MergeOU} (o_u1, o_u2) = \text{ContActSem} (a, o_s1 \cup \{ \text{EpsSem} \}) \)

82
⇒ OToSeq (MergeOU (ou1, ou2)) = OToSeq (ContActSem (a, os1 ∪ { EpsSem } ))
= [a ; OSToSeq (os1 ∪ { EpsSem } ) ]
and
⇒ [ OToSeq (ou1) | OToSeq (ou2) ] = [ (a ; OToSeq (os1)) | a ] = [ a ; ( OToSeq (os1) | ε ) ]
= [ a ; ( OToSeq (os1) | OToSeq ( { EpsSem } ) ) ]
= [ a ; OSToSeq (os1 ∪ { EpsSem } ) ]
from theorem 68
= OToSeq (MergeOU (ou1, ou2)).

(iii) ou2 = FinalAbActSem (a)
⇒ MergeOU (ou1, ou2) = ContActSem (a, os1)
⇒ OToSeq (MergeOU (ou1, ou2)) = OToSeq (ContActSem (a, os1))
= [a ; OSToSeq (os1) ]
and
⇒ [ OToSeq (ou1) | OToSeq (ou2) ] = [ (a ; OToSeq (os1)) | (a ; φ ) ]
= [ a ; ( OToSeq (os1) | φ ) ] = [ a ; OSToSeq (os1) ]
= OToSeq (MergeOU (ou1, ou2)).

Hence, the theorem holds for all the possible constructions of ou1 and ou2 in this inductive case, and so by induction it holds for all values of n, and so holds.

As noted above, the definitions of the functions OSToSeq and OToSeq only need to assume that their parameters are in zeroth and first normal forms. The significance of second normal form is that it reflects in the forest structures the fact that φ is the identity for alternation, so that in second normal form there are no objects corresponding to constructions in the DFA of the form s | φ. Consequently, while converting the parameter to a call of OSToSeq or OToSeq to second normal form may alter the actual object produced as a result if it is interpreted as an element of SeqConst, it should not alter the result when it is interpreted as an element of Seq, since the results in the two cases should be equal under the axioms of the DFA. This property is expressed formally as the following theorem.

**Theorem 70.**
∀ os : OpSem | OSNorm01 (os) • OSToSeq (NormOS2 (os)) = OSToSeq (os).

**Proof.**

The proof is by a double induction, where the outer induction is over HeightOS (os), and the inner induction is over the metric # os − # Heads (os). For each of the base and inductive cases the argument involves analysing an arbitrary element ou : OpSUnit of os, such that GetHead (ou) = a, where a : Act.

**Outer Base Case.**
The outer base case is that HeightOS (os) = 1, so that there are only a limited range of constructions for the elements ou of os, although the details of these do not need to be used directly in the analysis. For them, the base and inductive cases for the inner induction are as follows.

**Inner Base Case.**
The inner base case is that # os − # Heads (os) = 0 ⇒ NormOS2 (os) = NormOS2b (os), and then the effect of the limited constructions of ou arising from the outer base case is that in applying NormOS2b (os), the applications of NormOU2 will not alter the element ou. Hence the only possible difference between NormOS2b (os) and os could be as a consequence of NormOS2b (os) omitting an element PhiSem from os, which gives two possible cases, as follows.

(i) PhiSem ∉ os ⇒ NormOS2b (os) = os ⇒ NormOS2 (os) = os
⇒ OSToSeq (NormOS2 (os)) = OSToSeq (os).
(ii) PhiSem ∈ os ⇒ NormOS2b (os) = os − { PhiSem }
⇒ NormOS2 (os) = os − { PhiSem }
⇒ NormOS2 (os) ∪ { PhiSem } = os
⇒ OSToSeq (os) = OSToSeq (NormOS2 (os) ∪ { PhiSem } )
= [ OSToSeq (NormOS2 (os)) | OSToSeq ( { PhiSem } ) ]
= [ OSToSeq (NormOS2 (os)) | φ ]
= OSToSeq (NormOS2 (os)).

Thus, the theorem holds for the inner base case.

**Inner Inductive Case.**
The inner inductive case is given by # os − # Heads (os) > 0, so that the induction hypothesis is that, for any natural number ni > 0, the theorem holds for any os such that # os − # Heads (os) < ni, and the induction step is to show that therefore it holds for # os − # Heads (os) = ni.
In this case there must exist elements \( a : \text{Act} \) and \( \text{ou}_1, \text{ou}_2 : \text{OpSUnit} \) such that \( \text{ou}_1 \in \text{os} \land \text{ou}_2 \in \text{os} \land \text{ou}_1 \neq \text{ou}_2 \land \) \( \text{GetHead} (\text{ou}_1) = \text{GetHead} (\text{ou}_2) = a \). Let us define the forest \( \text{os}' : \text{OpSem} \) as \( \text{os}' = \text{os} - \{ \text{ou}_1 \} - \{ \text{ou}_2 \} \), so that we have \( \text{os} = \text{os}' \cup \{ \text{ou}_1, \text{ou}_2 \} \).

Then, the operation of \( \text{NormOS2} \) is to invoke \( \text{MergeOU} \) to combine the elements \( \text{ou}_1 \) and \( \text{ou}_2 \) into a single element, so as to produce a new forest \( \text{os}'' : \text{OpSem} \) given by \( \text{os}'' = \text{os}' \cup \{ \text{MergeOU} (\text{ou}_1, \text{ou}_2) \} \), where

\[
\# \text{os}'' - \# \text{Heads} (\text{os}'' ) = n_i - 1,
\]

and \( \text{NormOS2} (\text{os}) = \text{NormOS2} (\text{os}'' ) \) 

\[ \Rightarrow \text{OSToSeq} \left( \text{NormOS2} (\text{os}) \right) = \text{OSToSeq} \left( \text{NormOS2} (\text{os}'' ) \right) \]

from the induction hypothesis

\[ = \text{OSToSeq} (\text{os}'' ) \]

\[ = \text{OSToSeq} (\text{os}' \cup \{ \text{MergeOU} (\text{ou}_1, \text{ou}_2) \}) \]

from theorem 68

\[ = [ \text{OSToSeq} (\text{os}' ) \mid \text{OSToSeq} (\{ \text{MergeOU} (\text{ou}_1, \text{ou}_2) \}) ] \]

from theorem 69

\[ = [ \text{OSToSeq} (\text{os}' ) \mid \text{OUToSeq} (\text{ou}_1) \mid \text{OUToSeq} (\text{ou}_2) ] \]

from theorem 68

\[ = \text{OSToSeq} (\text{os}' \cup \{ \text{ou}_1 \} ) \mid \text{OUToSeq} (\text{ou}_2) \]

from theorem 68

\[ = \text{OSToSeq} (\text{os}' \cup \{ \text{ou}_1, \text{ou}_2 \} ) \]

from theorem 68

\[ = \text{OSToSeq} (\text{os}). \]

Hence, the theorem holds for the inner inductive case, and so by induction it holds for all values of \( n_i \), and so holds for the outer base case. It should also be noted that the analysis of this inner inductive case does not depend explicitly on the height of \( \text{os} \), and so it can be reused for the inner induction of the outer inductive case without needing to be repeated. On the other hand, its use in the induction for the outer base case does depend on this property, because the inner base case depends on it. Consequently, in the outer inductive case the induction over \( n_i \) will be starting from a different set of base cases, and so in principle it will be a different induction.

**Outer Inductive Case.**

The outer inductive case is that \( \text{HeightOS} (\text{os}) > 1 \), so that the induction hypothesis is that, for any natural number \( n_o > 0 \), the theorem holds for any \( \text{os} \) such that \( \text{HeightOS} (\text{os}) < n_o \), and the induction step is to show that therefore it holds for \( \text{HeightOS} (\text{os}) = n_o \). The effect of this is that there is an additional construction for \( \text{ou} \) that needs to be analysed in the inner base case, and here (unlike the outer base case) it does need to be analysed explicitly. On the other hand, as noted above, the analysis for the inner inductive case can simply be reused here.

**Inner Base Case.**

Again, the inner base case is that \( \# \text{os} - \# \text{Heads} (\text{os}) = 0 \Rightarrow \text{NormOS2} (\text{os}) = \text{NormOS2b} (\text{os}) \), but here the effect of applying \( \text{NormOU2} \) as part of the application of \( \text{NormOS2b} (\text{os}) \) does have to be analysed. On the other hand, it is not necessary to repeat the analysis of the effects of \( \text{NormOS2b} (\text{os}) \) omitting an element \( \PhiSem \) from \( \text{os} \), since the previous analysis of this did not depend on the height of the forest \( \text{os} \).

Thus, we have for the construction of \( \text{ou} \) that

\[ \exists \text{os}_1 : \text{OpSem} \mid \text{IsActive} (\text{os}_1) \rightarrow \text{ou} = \text{ContActSem} (a, \text{os}_1) \]

\[ \Rightarrow \text{NormOU2} (\text{ou}) = \text{Cont1ActSem} (a, \text{NormOS2} (\text{os}_1)) \]

which then gives two possible cases, depending on the construction of \( \text{os}_1 \).

(i) \( \text{os}_1 = \{ \text{EpsSem}, \PhiSem \} \)

\[ \Rightarrow \text{NormOS2} (\text{os}_1) = \{ \text{EpsSem} \} \Rightarrow \text{NormOU2} (\text{ou}) = \text{FinalActSem} (a) \]

\[ \Rightarrow \text{OUToSeq} (\text{NormOU2} (\text{ou})) = \text{OUToSeq} (\text{FinalActSem} (a)) = [a] \]

and

\[ \Rightarrow \text{OUToSeq} (\text{ou}) = \text{OUToSeq} (\text{ContActSem} (a, \{ \text{EpsSem}, \PhiSem \})) = \text{OUToSeq} (\text{ContActSem} (a, \{ \text{EpsSem}, \PhiSem \})) = [a; \text{OUToSeq} (\{ \text{EpsSem}, \PhiSem \})] = [a; \{ \}$1$], \}

\[ = [a; \text{OUToSeq}(\text{NormOU2}(\text{ou}))]. \]

(ii) otherwise, \( \text{NormOU2} (\text{ou}) = \text{ContActSem} (a, \text{NormOS2} (\text{os}_1)) \)

\[ \Rightarrow \text{OUToSeq} (\text{NormOU2} (\text{ou})) = \text{OUToSeq} (\text{ContActSem} (a, \text{NormOS2} (\text{os}_1))) \]

\[ = [a; \text{OUToSeq} (\text{NormOS2}(\text{os}_1))], \]

\[ = [a; \text{OUToSeq}(\text{os}_1)], \]

\[ = \text{OUToSeq} (\text{ContActSem} (a, \text{os}_1)) \]

\[ = \text{OUToSeq} (\text{ou}). \]

Hence, for any possible construction of an arbitrary element \( \text{ou} \) we have that

\[ \text{OUToSeq} (\text{NormOU2} (\text{ou})) = \text{OUToSeq} (\text{ou}) \]

and hence (by the argument concerning the omission of \( \PhiSem \) that was analysed for the inner base case of the outer base case) that

\[ \text{OSToSeq} (\text{NormOS2} (\text{os})) = \text{OSToSeq} (\text{os}) \]

so that the theorem holds for this inner base case too.
Inner Inductive Case.

Again, the inner inductive case is given by \( \# os - \# \text{Heads}(os) > 0 \), the induction hypothesis is that, for any natural number \( ni > 0 \), the theorem holds for any \( os \) such that \( \# os - \# \text{Heads}(os) < ni \), and the induction step is to show that therefore it holds for \( \# os - \# \text{Heads}(os) = ni \).

As already noted, the analysis from the inner inductive case of the outer base case applies unchanged, and does not need to be repeated here. Hence, by induction the theorem holds for all values of \( ni \), and so holds for the outer inductive case (which applies to forests of any height), so that this holds too.

Therefore, the theorem holds too for the outer inductive case, and so by induction it holds for all values of \( no \), and so holds.

The final intermediate result that is needed for the functions \( \text{OSToSeq} \) and \( \text{OUToSeq} \) is the equivalent of theorem 56 for the DFA operation of sequencing, and this is expressed as the following theorem.

**Theorem 71.**

\[
\forall os1, os2 : \text{OpSem} \mid \text{os1} \neq \emptyset \land \text{OSNorm01}(os1) \land \text{os1} \neq \emptyset \land \text{OSNorm01}(os2) \Rightarrow \\
\text{OSToSeq}(\text{SeqCompOS}(os1, os2)) = [\text{OSToSeq}(os1); \text{OSToSeq}(os2)].
\]

**Proof.**

The proof is by induction over the height of the forest \( os1 \), and again for both the base and inductive cases it involves analysing an arbitrary element \( ou1 : \text{OpSUnit} \) of \( os1 \), such that \( \text{GetHead}(ou1) = a \), where \( a : \text{Act} \), in order to show that \( \text{OSToSeq}(\text{SeqCompOU}(ou1, os2)) = [\text{OUToSeq}(ou1); \text{OSToSeq}(os2)] \)

from which the result of the theorem follows directly.

**Base case.**

The base case is that \( \text{HeightOS}(os1) = 1 \Rightarrow \text{HeightOU}(ou1) = 1 \), so that from the conditions of the theorem and from theorem 5 there are four possible constructions for \( ou1 \), as follows.

(i) \( ou1 = \text{EpsSem} \Rightarrow \text{SeqCompOU}(ou1, os2) = os2 \)
\[
\Rightarrow \text{OSToSeq}(\text{SeqCompOU}(ou1, os2)) = \text{OSToSeq}(os2) = [\varepsilon; \text{OSToSeq}(os2)] = [\text{OUToSeq}(\text{EpsSem}); \text{OSToSeq}(os2)].
\]

(ii) \( ou1 = \text{PhiSem} \Rightarrow \text{SeqCompOU}(ou1, os2) = \{ \text{PhiSem} \} \)
\[
\Rightarrow \text{OSToSeq}(\text{SeqCompOU}(ou1, os2)) = \text{OSToSeq}(\{ \text{PhiSem} \}) = [\phi] = [\phi; \text{OSToSeq}(os2)] = [\text{OUToSeq}(\text{PhiSem}); \text{OSToSeq}(os2)] = [\text{OUToSeq}(ou1); \text{OSToSeq}(os2)].
\]

(iii)(a) \( ou1 = \text{FinalActSem}(a) \Rightarrow \text{SeqCompOU}(ou1, os2) = \{ \text{Cont1ActSem}(a, os2) \} \)
\[
\text{and there are then three possible sub-cases to be analysed, depending on the construction of os2, as follows.}
\]

(iii)(a)(i) \( os2 = \{ \text{EpsSem} \} \Rightarrow \text{Cont1ActSem}(a, os2) = \text{FinalActSem}(a) \)
\[
\Rightarrow \text{OSToSeq}(\text{SeqCompOU}(ou1, os2)) = \text{OSToSeq}(\{ \text{FinalActSem}(a) \}) = [a] = [a; \varepsilon] = [\text{OUToSeq}(\text{FinalActSem}(a)); \text{OSToSeq}(\{ \text{EpsSem} \})] = [\text{OUToSeq}(ou1); \text{OSToSeq}(os2)].
\]

(iii)(b) otherwise \( os2 = \{ \phi \} \Rightarrow \text{Cont1ActSem}(a, os2) = \text{FinalAbActSem}(a) \)
\[
\Rightarrow \text{OSToSeq}(\text{SeqCompOU}(ou1, os2)) = \text{OSToSeq}(\{ \text{FinalAbActSem}(a) \}) = [a; \phi] = [\text{OUToSeq}(\text{FinalActSem}(a)); \text{OSToSeq}(\{ \text{FinalAbActSem}(a) \}) = [\text{OUToSeq}(ou1); \text{OSToSeq}(os2)].
\]

(iv) \( ou1 = \text{FinalAbActSem}(a) \Rightarrow \text{SeqCompOU}(ou1, os2) = \{ \text{FinalAbActSem}(a) \} \)
\[
\Rightarrow \text{OSToSeq}(\text{SeqCompOU}(ou1, os2)) = \text{OSToSeq}(\{ \text{FinalAbActSem}(a) \}) = [a; \phi] = [\text{OUToSeq}(\text{FinalAbActSem}(a)); \text{OSToSeq}(os2) = [\text{OUToSeq}(ou1); \text{OSToSeq}(os2)].
\]

Hence, the theorem holds for all four of these cases, and so holds for the base case.
Inductive case.
The inductive case is that $\text{HeightOS (os1)} > 1 \Rightarrow \text{HeightOU (ou1)} > 1$, so that the induction hypothesis is that, for any natural number $n > 0$, the theorem holds for any $\text{os1}$ such that $\text{HeightOS (os1)} < n$, and the induction step is to show that therefore it holds for $\text{HeightOS (os1)} = n$. The effect of this is that there is an additional construction for $\text{ou}$ that needs to be analysed, as follows.

\[
\exists \text{os1x : OpSem | IsActive (os1x)} \bullet \text{ou1 = ContActSem (a, os1x)}
\Rightarrow \{ \text{SeqCompOU (ou1, os2)} = \{ \text{Cont1ActSem (a, SeqCompOS (os1x, os2))} \} \\
= \{ \text{ContActSem (a, SeqCompOS (os1x, os2))} \} \text{ from theorems 36 and 35} \Rightarrow \text{OSToSeq (SeqCompOU (ou1, os2)) = OSToSeq (\{ ContActSem (a, SeqCompOS (os1x, os2)) \)} \\
= [a ; \text{OSToSeq (SeqCompOS (os1x, os2))}] = \{ [a ; \text{OSToSeq (os1x)} ; \text{OSToSeq (os2)}] \} \text{ from the induction hypothesis} \\
= [\text{OUToSeq (ContActSem (a, os1x)) ; OSToSeq (os2)}] = [\text{OUToSeq (ou1)} ; \text{OSToSeq (os2)}] \\
\]  

Hence the theorem holds for the inductive case, and so by induction it holds for all values of $n$, and so holds.

Given these intermediate results, then the property that $\text{OSToSeq}$ is indeed the inverse of $\text{OpS}$ is established formally as the following theorem.

**Theorem 72.**
\[
\forall s : \text{SeqConst} \bullet \text{OSToSeq (OpS (s)) = s.}
\]

**Proof.**
This is by structural induction over $s$, with the function SCC as metric for the induction.

**Base case.**
The base case is that $\text{SCC (s)} = 1$, meaning that $s$ must be an action, and this gives rise to the following three sub-cases, which all have the same structure.

Sub-case (i): $s = a$ where $a \in PA$
\[
\Rightarrow \text{OpS (s)} = \{ \text{FinalActSem (a)} \} \\
\Rightarrow \text{OSToSeq (OpS (s)) = OSToSeq (\{ FinalActSem (a) \})} \\
= [a] = s.
\]

Sub-case (ii): $s = \epsilon \Rightarrow \text{OpS (s)} = \{ \text{EpsSem} \} \\
\Rightarrow \text{OSToSeq (OpS (s)) = OSToSeq (\{ EpsSem \})} \\
= [\epsilon] = s.
\]

Sub-case (iii): $s = \phi \Rightarrow \text{OpS (s)} = \{ \text{PhiSem} \} \\
\Rightarrow \text{OSToSeq (OpS (s)) = OSToSeq (\{ PhiSem \})} \\
= [\phi] = s.
\]

Hence, the theorem holds for the base case.

**Inductive case.**
The induction hypothesis is that, for any $n > 1$, the theorem holds for all $s : \text{SeqConst}$ such that $\text{SCC (s)} < n$, and so the induction step is to show from this that the theorem must also hold for any $s : \text{SeqConst}$ such that $\text{SCC (s)} = n$. There are then two sub-cases to be analysed, depending on the construction of $s$: one for the case where it is constructed by alternation, and the other for the case where it is constructed by sequencing.

Alternation sub-case: $s = s_1 | s_2$, where $\text{SCC (s1)} < n$ and $\text{SCC (s2)} < n$, for which we have
\[
\text{OpS (s) = NormOS2 (OpS (s1) \cup \text{Ops (s2)})} \\
\Rightarrow \text{OSToSeq (OpS (s)) = OSToSeq (NormOS2 (OpS (s1) \cup \text{Ops (s2)})})} \\
= \text{OSToSeq (OpS (s1) \cup \text{Ops (s2)})} \text{ from theorem 70} \\
= [\text{OSToSeq (Ops (s1)) \cup \text{OSToSeq (Ops (s2))}] \text{ from theorem 68} \\
= [s_1 | s_2] \text{ from the induction hypothesis} \\
= s.
\]
Hence, the theorem holds for this alternation sub-case of the inductive case.

Sequencing sub-case: \( s = s_1 ; s_2 \), where SCC \((s_1) < n \) and SCC \((s_2) < n \), for which we have

\[
\text{OpS} (s) = \text{NormOS2} (\text{SeqCompOS} (\text{OpS} (s_1), \text{Ops} (s_2))) .
\]

\[
\Rightarrow \text{OSToSeq} (\text{OpS} (s)) = \text{OSToSeq} (\text{NormOS2} (\text{SeqCompOS} (\text{OpS} (s_1), \text{Ops} (s_2))))
\]

\[
= \text{OSToSeq} (\text{SeqCompOS} (\text{OpS} (s_1), \text{Ops} (s_2))) \quad \text{from theorem 70}
\]

\[
= [ \text{OSToSeq} (\text{OpS} (s_1)) ; \text{OSToSeq} (\text{Ops} (s_2))] \quad \text{from theorem 71}
\]

\[
= [s_1 ; s_2]
\]

\[
= s.
\]

Hence, the theorem holds for this sequencing sub-case of the inductive case, and so the inductive case follows from the combination of these two sub-cases. Therefore, by induction the theorem holds for all values of \( n \), and so holds.

Between them, therefore, the results of theorems 67 and 72 show that \text{OpS} and \text{OSToSeq} are inverses of each other in both directions, so that whether one starts from elements of \text{OpSem} (as in theorem 67) or from elements of \text{SeqConst} (as in theorem 72), successive applications of each will always produce a result that is equal to the starting value.

### 12. Summary and Conclusions

In summary, therefore, this report has shown how an operational semantics can be constructed for the event layer of the DFA, and that this semantics is completely consistent with the algebraic properties of the DFA event layer, as these were defined by the axioms in section 6 of [4]. Specifically, the report has defined the possible states of the computation that results from executing an event layer specification, and hence has defined (in clauses (i) to (xii) of section 2) the possible transitions between these states that can occur as such a computation progresses. This therefore constitutes a state machine model for such computations, and from this a structure (namely \text{OpSem}) has been derived (in section 3) to represent the derivation sequences of this model: that is, the set of possible execution paths for this state machine. Also, a function has been defined (namely \text{DerSeq}) that, for any event layer specification, will produce the corresponding structure directly from this state machine model.

Then the normal forms for these derivation sequences have been defined, along with the normalisation functions that produce them, in order to model the effects of some of the axioms of the DFA event layer. Specifically, strict first normal form models the property that the silent action is the identity for the operation of sequencing, and strict second normal form models the properties that the forbidden action is the identity for the operation of alternation, and that any event layer expression of the form \((s_1 ; s_2a) | (s_1 ; s_2b)\) can be rewritten as \(s_1 ; (s_2a | s_2b)\). Demonstrating the properties of these normal forms and the associated functions has formed a significant part of this report (sections 4, 5 and 6), but this has then made it possible to define (in section 7) various auxiliary functions that model important features of the event layer, and particularly the function \text{SeqCompOS} that models the operation of sequencing. Demonstrating that these various auxiliary functions possess properties that enable them to reflect accurately the corresponding properties of the event layer has then also formed a significant part of this report, not least because of the complexity of the proofs of some of the theorems involved, but it has provided the basis for the other key results in this work.

One of these key results is the definition (in section 9) of the semantic function (namely \text{OpS}), which for any element of \text{SeqConst} produces the structure constituting its semantics, and the next is the property (theorem 55) that this structure is the normalised form of the equivalent derivation sequence. From these it has then been shown (in section 10) that the axioms of the DFA event layer are sound with respect to the semantics, although (as with the denotational semantics defined in [4]) the fact that the axioms were defined first means that it is probably more appropriate to regard this property as showing instead that the semantics are consistent with the axioms. Similarly, the final key result is that the algebra for the DFA event layer is also complete with respect to the semantics, in that for every possible structure in the semantics there is a construction in the DFA for which this structure is the semantics. As with the denotational semantics defined in [4], this is shown (in section 11) by constructing an inverse semantic function (namely \text{OSToSeq}), and then proving the property that this function and \text{OpS} are actually inverses of each other. Again, though, it is more appropriate to regard this as establishing a property of the semantics, rather than a property of the algebra.

One consequence of this relationship between the properties of the algebra and the properties of the semantics is that, while the results themselves are clear, their significance is perhaps less clear. When the denotational semantics was defined in [4], it was reasonable to claim that the proofs of completeness of the algebra forming the event layer, and of the soundness of the axioms for this layer, served to validate the approach that had been taken to defining the DFA itself. Since it had been so validated, though, it might now be equally reasonable to question the extent to which defining this operational semantics has provided further validation, since it could be argued that one ought to be able to create an...
The first of these points is that one of the features which might be expected to complicate the construction of an operational semantics for a language is non-determinism, and this (in the form of the alternation operator) is a key feature of the DFA event layer. Thus, the fact that an operational semantics has been defined successfully is important for demonstrating that, while the DFA is non-deterministic, this non-determinism is well-behaved, in the sense of giving rise to a well-defined branching structure within the set of possible execution paths for a DFA specification, as represented by the tree and forest structures of OpSem and OpSUnit. Indeed, the analysis of this branching structure and of the constraints on it, as the latter are modelled by the various normal forms, is probably a more significant outcome of this work than any contribution that it has made to increasing confidence in the validity of the DFA as a modelling formalism.

This leads to the second point, which is that while it is easy to say that one ought to be able to construct some formalism, the only way to find out what problems might need to be overcome in so doing is actually to construct it. In this case, this has principally involved analysing the effects of the non-determinism inherent in the alternation operator, and the way in which these effects do give rise to this branching structure for the execution paths. This structure is certainly not explicit in the denotational semantics, although it is implicit in the use of the function Prefixes to define the invariant for the domain SeqSem. Consequently, the operational semantics that has been defined here, which has to model this branching structure explicitly in terms of trees and forests, actually contains much more explicit information about the meaning of a construction in the DFA than the denotational semantics does, and so it is the identification and representation of this additional information that should be seen as the main contribution of this report.

Of course, now that both forms of semantics have been defined, it is apparent that this branching structure is implicit in the denotational semantics, in the form of different strings of actions having a common prefix. Hence, since inverse semantic functions have been defined for both forms of semantics, one would expect it to be possible also to define direct mappings between the two forms of semantics, so that the mapping from the denotational to the operational semantics would involve identifying occurrences of this branching structure and making them explicit. For the purposes of this report, though, this has to be left as future work.

The other consequence of this work that has to be left for the future is the question that it raises as to how the semantics of the event layer might be reflected back into the DFA itself. It appears that the two forms of the semantics give rise to two different notions of a canonical form for structures in SeqConst, in that if a1, a2 and a3 are elements of PA, then the expression (a1 ; a2) | (a1 ; a3) is in a form that reflects the structure of the denotational semantics, while the equivalent expression a1 ; (a2 | a3) is in a form that reflects the structure of the operational semantics. Of course, such forms are only partly canonical – a1 ; (a2 | a3) could equally well be written as a1 ; (a3 | a2) – but even so there would seem to be value in being able to identify elements of SeqConst that are in one or the other of these forms that reflect the structure of one of the sets of semantics. In particular, there are constructions in the DFA, such as parallel composition, whose behaviour can be at least partly understood in terms of the concepts underlying the operational semantics, and so there may be advantages in being able to frame definitions of these in terms of the concepts underlying the operational semantics, either instead of or as well as defining them in terms of the basic structural recursions, as was done in [4]. Furthermore, if in exploring such ideas it does then prove valuable to identify completely canonical forms in SeqConst, it appears that this should be achievable by imposing some (probably arbitrary) order relation on the elements of Act, but, as already noted, exploring this needs to be left as future work.

In summary therefore, this report has defined an operational semantics for the event layer of the DFA, and it can be concluded that this is significant for three reasons. Firstly it defines a richer semantic structure than the denotational semantics, and so in some ways it captures more of the important features of the behaviour of the objects in this layer. Secondly, although less importantly, it further reinforces the validation of the event layer model that was provided by the definition of the denotational semantics in [4]. Finally, it provides the theoretical foundation on which to explore further the notion of canonical constructions within the event layer, and specifically within the domain SeqConst.

13. References

Appendix: List of Theorems

The following list briefly summarises the results of each of the theorems presented above. Following the usual practice for normal forms in databases, the abbreviations 0NF, 1NF and 2NF are used here for zeroth, first and second normal forms respectively.

Theorem 1.
EpsSem and PhiSem are consistent with the invariants.

Theorem 2.
FinalActSem and FinalAbActSem are consistent with the invariants.

Theorem 3.
ContActSem and EmptyTrans are consistent with the invariants.

Theorem 4.
EpsSem, PhiSem, FinalActSem and FinalAbActSem are all in 0NF.

Theorem 5.
Objects in 0NF with HeightOU = 1 must be constructed as EpsSem, PhiSem, or by FinalActSem or FinalAbActSem

Theorem 6.
ContActSem (a, os) and EmptyTrans (os) maintain 0NF.

Theorem 7.
DerSeq (s) is in 0NF.

Theorem 8.
EpsSem, PhiSem, FinalActSem and FinalAbActSem are all in strict 1NF.

Theorem 9.
Conditions under which ContActSem is in strict 1NF.

Theorem 10.
EpsSem, PhiSem, FinalActSem and FinalAbActSem are all in strict 2NF.

Theorem 11.
Conditions under which ContActSem is in strict 2NF.

Theorem 12.
The only objects in 1NF for which ou.DoesAct = false are EpsSem and PhiSem

Theorem 13.
Cont1ActSem (a, os) maintains 1NF for os.

Theorem 14.
If os is active, Cont1ActSem and ContActSem are equal.

Theorem 15.
Objects in 1NF must be constructed as EpsSem, PhiSem, or by FinalActSem, FinalAbActSem or ContActSem

Theorem 16.
NormOS1 (and hence NormOU1) maintain 0NF.

Theorem 17.
NormOS1 (and hence NormOU1) produce objects in strict 1NF from parameters in 0NF.

Theorem 18.
NormOS1 and NormOU1 do not change objects that are already in 1NF.

Theorem 19.
MergeOU is commutative.

Theorem 20.
MergeOU is associative.

Theorem 21.
MergeOU is idempotent.

Theorem 22.
Any objects in 1NF with head ε or φ must be EpsSem or PhiSem respectively.

Theorem 23.
Any objects in 1NF constructed by Cont1ActSem must have head a.
Theorem 24. Relationship between Heads (NormOS2b (os)) and Heads (os).

Theorem 25. Consequences of # Heads (os) < # os or # Heads (os) = # os for existence of objects with the same head.

Theorem 26. NormOS2 (and hence NormOU2) maintains 1NF.

Theorem 27. NormOS2 produces objects in strict 2NF from parameters in 1NF.

Theorem 28. NormOS2 and NormOU2 do not change objects that are already in 2NF.

Theorem 29. Relationship between Heads (NormOS2 (os)) and Heads (os).

Theorem 30. For objects in 2NF, # Heads (os) = # os

Theorem 31. Relationships between PhiSem and os, and between φ and Heads (os), for objects in 2NF.

Theorem 32. Relationships between EndsAfter, AbEndsAfter and RestAfter.

Theorem 33. OSeqN2 is equivalent to equality for objects in 2NF.

Theorem 34. SeqCompOS maintains 1NF.

Theorem 35. NormOU2 (ContActSem) = ContActSem (NormOS2)

Theorem 36. SeqCompOS maintains activity of its first parameter.

Theorem 37. SeqCompOS maintains activity of its second parameter, unless the first parameter is { PhiSem }

Theorem 38. Heads distributes over set union.

Theorem 39. NormOS1 distributes over set union.

Theorem 40. PhiSem and EpsSem can be introduced into sets that are in 2NF.

Theorem 41. NormOS2 can be introduced into the first parameter of “NormOS2 applied to set union”.

Theorem 42. NormOS2 can be introduced into the second parameter of “NormOS2 applied to set union”.

Theorem 43. NormOS2 can be introduced into both parameters of “NormOS2 applied to set union”.

Theorem 44. NormOU2 can be introduced into the first parameter of “NormOU2 applied to MergeOU”.

Theorem 45. NormOU2 can be introduced into the second parameter of “NormOU2 applied to MergeOU”.

Theorem 46. NormOU2 can be introduced into both parameters of “NormOU2 applied to MergeOU”.

Theorem 47. Evaluate NormOU2 (MergeOU (Cont1ActSem, Cont1ActSem))

Theorem 48. NormOS2 (SeqCompOS) distributes over set union for its first parameter.

Theorem 49. NormOS2 (SeqCompOS) distributes over set union for its second parameter.

Theorem 50. NormOS2 (SeqCompOS) is associative.

Theorem 51. { EpsSem } is left identity for SeqCompOS.

Theorem 52. { EpsSem } is right identity for SeqCompOS.

Theorem 53. OpS is in 2NF

Theorem 54. Equality of SeqHeads and Heads (OpS)
Theorem 55. Equality of \(\text{OpS} \) with result of normalising \(\text{DerSeq} \) to 2NF

Theorem 56. Consistency of \(\text{OpS} \) with axiom (i) of the DFA (sequencing is associative).

Theorem 57. Consistency of \(\text{OpS} \) with axiom (ii) of the DFA (\(\varepsilon\) is identity for sequencing).

Theorem 58. Consistency of \(\text{OpS} \) with axiom (iii) of the DFA (alternation is associative).

Theorem 59. Consistency of \(\text{OpS} \) with axiom (iv) of the DFA (alternation is commutative).

Theorem 60. Consistency of \(\text{OpS} \) with axiom (v) of the DFA (alternation is idempotent).

Theorem 61. Consistency of \(\text{OpS} \) with axiom (vi) of the DFA (\(\phi\) is identity for alternation)

Theorem 62. Consistency of \(\text{OpS} \) with axiom (vii) of the DFA (sequencing distributes over alternation for its right parameter)

Theorem 63. Consistency of \(\text{OpS} \) with axiom (viii) of the DFA (sequencing distributes over alternation for its left parameter).

Theorem 64. Consistency of \(\text{OpS} \) with axiom (ix) of the DFA (\(\phi\) is a left zero for sequencing).

Theorem 65. Consistency of \(\text{OpS} \) with any single axiom of the DFA.

Theorem 66. \(\text{OpS} \) ensures consistency of equality in \(\text{SeqConst} \) and \(\text{OpSem} \).

Theorem 67. \(\text{OpS} \) (\(\text{OSToSeq} \)) is an identity.

Theorem 68. \(\text{OSToSeq} \) of a union is an alternation (a form of distributive property).

Theorem 69. \(\text{OSToSeq} \) applied to \(\text{MergeOU} \) is an alternation (also a form of distributive property).

Theorem 70. \(\text{OSToSeq} \) produces the same result for a forest in 1NF as for one in 2NF.

Theorem 71. \(\text{OSToSeq} \) applied to \(\text{SeqCompOS} \) is a sequence (a form of distributive property).

Theorem 72. \(\text{OSToSeq} \) (\(\text{OpS} \)) is an identity in \(\text{Seq} \).