California Institute of Technology
Computer Science Division

Concurrent System Design Using Flow

Submitted by
Cheng Hu

Master's Research Thesis
May 2007

Abstract

We define a formal model for concurrent systems named Flow. The goal of this formal model is to be both practical in allowing users to build the types of concurrent systems they want to build in real life and practical in allowing users to quickly prove correctness properties about the systems they build.

We define the model formally as the asynchronous product of extended state automata. Extended state automata and their asynchronous products are used to define other modeling languages as well, such as the state space exploration verification tool SPIN [11, 12]. Thus an offshoot of our formal definition is that we see it would not be difficult to use automated verification tools to verify Flow model properties.

Using the formal definition, we show a set of theorems that users will be able to reuse to prove correctness properties about Flow models. One category of theorems deals with showing and constructing Flow models for which all executions are guaranteed to be finite. Another category of theorems deals with showing or constructing Flow models for which all executions from a start state are guaranteed to have the same end state. This is useful because it allows users to know how all concurrent executions from a start state terminate by looking at just one execution. Another category of theorems deals with dividing complex Flow models into smaller sub-models, and showing the properties of the full model by showing the properties of the sub-models, allowing for a divide and conquer strategy to be used with Flow model proofs. The last category deals with using Hoare triples [10] to prove properties about all executions from a set of possible start states as characterized by some pre-condition by proving properties about a set of representative executions from those start states. In the best case, we can use a combination of these techniques to show that all executions of a Flow model with start states that satisfy some pre-condition have end states that satisfy some post-condition by considering just one feasible execution of the model.

Acknowledgements

This thesis would not have been possible without the support of my thesis advisor, Professor Steven Low. He gave me time and freedom to develop my own ideas as well as the problem which motivated me to develop Flow.

I am also grateful to Doctor Gerard Holzmann, the creator of the SPIN verification tool. It was his course on formal verification, his papers on the formal definition of SPIN, and my experience working under him at JPL that gave me the foundation I needed to define Flow formally.

I am also grateful to Professor Jason Hickey and Professor Alain Martin of the Caltech Computer Science Department. Professor Hickey encouraged me to formally define Flow, and were it not for his encouragement I might have shied away from this rewarding challenge. Professor Martin's course on concurrent computation first introduced me to the canon of formalisms used to prove concurrent programs correct, to which I now hope I am making a useful contribution with this thesis.

Table Of Contents

1. Introduction And Related Work

We define a formal model for concurrent systems named Flow. The goal of this model is to be both practical in allowing users to describe concurrent systems they want to build in real life and practical in allowing users to prove useful correctness properties about the concurrent systems they build.

We begin by defining the model formally as the asynchronous product of extended state automata. Extended state automata and their asynchronous products are used to define other modeling languages as well, such as the state space exploration verification tool SPIN [11, 12]. Thus an offshoot of our formal definition is that we see it would not be difficult to use automated verification tools to verify Flow model properties.

Using the formal definition, we show a set of theorems that users will be able to reuse to prove correctness properties about Flow models. One category of theorems deals with showing and constructing Flow models for which all executions are guaranteed to be finite. Another category of theorems deals with showing or constructing Flow models for which all executions from a start state are guaranteed to have the same end state. This is useful because it allows users to know how all concurrent executions from a start state terminate by looking at just one execution. Another category of theorems deals with dividing complex Flow models into smaller sub-models, and showing the properties of the full model by showing the properties of the sub-models, allowing for a divide and conquer strategy to be used with Flow model proofs. The last category deals with using Hoare triples [10] to prove properties about all executions from a set of possible start states as characterized by some pre-condition by proving properties about a set of representative executions from those start states. In the best case, we can use a combination of these techniques to show that all executions of a Flow model with start states that satisfy some pre-condition have end states that satisfy some post-condition by considering just one feasible execution of the model.

One can compare our model to the formal asynchronous network model used to study distributed computing [1, 5, 9 16], which it has a superficial similarity to. Both involves processing units that coordinate via communication channels. The most important differences are the underlying assumptions that govern how the two models execute. In the asynchronous network model, each processing unit can initiate action independently. In the Flow model, each processing unit can only execute in response to one or more messages sent to it on its communication channels. This enables us to prove some properties about Flow models that asynchronous network models do not have. Another important difference is that in asynchronous networks, the actions that transform the state of processors and the actions that send messages and receive messages are each an atomic unit of execution. In Flow models, the entire action which involves receiving messages, changing the state of the processor and sending out messages is one atomic unit of execution. This also allows us to prove certain properties about Flow model executions that asynchronous network models do not have.

One can also compare our model to an event web [2, 3], which is a practical implementation with many features added that is useful as enterprise and crisis management systems. Flow models are simpler to reason with because they do not have features such as timeouts and prioritized events, which event webs need to function as enterprise systems. In contrast, Flow is meant to be used as a modeling tool for

concurrent systems that facilitates correctness proofs. Indeed, one can use a Flow model to model an event web and take advantage of many of the theorems proved here to reason about the behavior of an event web.

Lastly, one can compare a Flow model to the flow network pattern [15, 18, 20] after which it is named. The flow network pattern, and a family of related patterns such as the pipeline pattern, are informal recipes which are used often to organize concurrent computation in real programs. Indeed, Flow models can be seen as a formalization of these patterns which allows us to prove why these patterns are useful for simplifying concurrent computation. Having a formal definition also allows us to precisely state what one can and cannot do with these patterns, and generalize on the properties that make these patterns effective to build models that are not traditionally allowed by these patterns, such as networks of processors that include cycles.

2. Flow Models

2.1 Formal Definition Of Flow Models

We begin by defining a data object. Intuitively, a data object is simply anything that has a well defined set of possible states and can be distinguished from other data objects. For example, an integer variable in a traditional programming language such as C can be considered a data object. We reuse modified versions of many of the definitions of extended state automata used to defined the automated verification tool SPIN. [11, 12]

Definition 2.1.1. A *data object* O is a tuple $\{id, V\}$ with an integer *id* and *domain* V a set of *values*. Two data objects with the same id are considered to be the same object and thus must have the same domain.

Definition 2.1.2. Specifying a *state* for a data object O means specifying a value from its domain. Two data objects have the same state if they are specified to have the same value.

We define some special types of data objects used in Flow models. Intuitively, an in-channel is a communications queue of events that components of Flow models use to communicate with each-other. An out-channel is a stub that is used to specify which in-channels a Flow model component can add events to.

Definition 2.1.3. A Flow model *out-channel* is a data object with an empty domain.

Definition 2.1.4. A Flow model *in-channel* is a data object that represents a queue containing a finite number of *events*. An *event* is a value from some domain of values. A state of an in-channel specifies the order of events in the in-channel and the value of each event. Any such specification also constitutes a state for the in-channel. We use C to denote an in-channel. We use $e_1 e_2 ... e_n$ to denote the state of an in-channel where e_1 is the event value at the *head* of the queue and e_n is the event value at the *tail*.

Actions manipulate the states of a set of data objects. An action consists of a guard and an effect. Intuitively, a guard checks the states of objects in the set to see whether the action should execute. If the guard evaluates to true, then the effect evaluates to states for objects in the set.

Definition 2.1.5. A *guard* on a finite set of data objects $D = \{O_1, O_2, ..., O_n\}$ is a total function $g : V_1 \times V_2 \times ... \times V_n \rightarrow \{true, false\}$ where V_i is the domain of O_i.

Definition 2.1.6. An *effect* on a finite set of data objects $D = \{O_1, O_2, ..., O_n\}$ is a partial function $f : V_1 \times V_2 \times ... \times V_n \rightarrow V_1 \times V_2 \times ... \times V_n$ where V_i is the domain of O_i.

Definition 2.1.7. An *action* on a finite set of data objects $D = \{O_1, O_2, ..., O_n\}$ consists of two parts: a guard and an effect on the set, with the assumption that if the guard evaluates to true on a set of values for the data objects, then the effect is defined on the set of values. We use a to denote an action.

An extended state automaton is a collection of data objects and the actions that manipulate them. Extended state automatons are used to define event producer consumers in Flow models, which is the construct by which data objects and actions are organized in Flow models. We do not define a starting state or finishing states for extended state automatons. This is because we do not use these constructs in our definition of Flow models.

Definition 2.1.8. An *extended state automaton* is a tuple $\{S, D, L, T\}$ with S a set of *states*, D a set of *data objects*, L a set of *actions* on objects in D, and $T \subseteq S \times L \times S$, called the *transition relation*.

Definition 2.1.9. Specifying a *state* for an extended state automaton means specifying a state for each of the data objects in D and a state from S.

We now describe what we mean by a Flow model, which is simply a collection of event producer consumers (EPCs) and connections between their in-channels and out-channels.

Intuitively, an EPC, which we will map to an extended state automaton, consists of the data objects that represent its internal state, the in-channels that it can read from and out-channels it can write to. The actions of an EPC are predicated upon the internal state and in-channels that belong to the EPC and manipulate the internal state of the EPC and write events to the out-channels of the EPC.

Intuitively, a connection between the out-channel of one EPC and the in-channel of another means that all events written to the out-channel of the first EPC is added to the in-channel of the second EPC.

Definition 2.1.10. A *Flow model* is a set of event producer consumers as defined in 2.1.11 and a set of connections as defined in 2.1.15 satisfying the constraint that no data object, in-channel, or out-channel *belongs* to more than one event producer consumer as defined in 2.1.11. We use F to denote a Flow model.

Definition 2.1.11. An *event producer consumer (EPC)* is a tuple $\{D, OC, IC, L\}$ with D a finite set of data objects not including in-channels and out-channels, OC a set of out-

channels, IC a set of in-channels, and L a set of Flow actions as defined in 2.1.16. All the data objects of D, out-channels of OC, in-channels of IC, and actions of L are said to *belong* to the EPC. We use M to denote an EPC.

Definition 2.1.12. Specifying an *internal state* for an EPC means specifying a state for each of the data objects in D.

Definition 2.1.13. Specifying a *local state* for an EPC means specifying a state for each of the data objects in D and each of the in-channels in IC.

Definition 2.1.14. Specifying a *state* for a Flow model means specifying a local state for each of the EPCs in the model.

Definition 2.1.15. A Flow model *connection* is an ordered pair (O_1, O_2) where O_1 is an out-channel belonging to an EPC in the model and O_2 is an in-channel belonging to an EPC in the model. We say that out-channel O_1 and in-channel O_2 are *connected* in this case.

Definition 2.1.16. A Flow action a of an EPC M consists of a guard and an effect. The guard is a total function $g : V_1 \times V_2 \times ... \times V_n \rightarrow \{true, false\}$ where $V_1, V_2, ..., V_n$ consists of the domains of each data object in D and the domains of a fixed number of in-channels $C_1, C_2, ..., C_m$ in IC. The guard must satisfy the following conditions:

1. Given a value in the domain of the guard of a, which consists of a value for each data objects in D and a value for each in-channel $C_1, C_2, ..., C_m$, if the value of any of the in-channels $C_1, C_2, ..., C_m$ is such that the in-channel is empty, then the guard must evaluate to false; and
2. given two values v_1 and v_2 in the domain of the guard of a, if none of the in-channels $C_1, C_2, ..., C_m$ are empty in v_1, and none of them are empty in v_2, and the event at the head of C_i in v_1 has the same value as the event at the head of C_i in v_2 for each in-channel $C_1, C_2, ..., C_m$, and the value of any data object in D is the same in v_1 and v_2, then the guard evaluates to true on v_1 if and only if it evaluates to true on v_2.

The effect is a partial function $f : V_1 \times V_2 \times ... \times V_n \rightarrow V_1 \times V_2 \times ... \times V_n x$ {sequence of outputs on out-channels of OC}. Note that the domain of f and g are the same, and consists of the domains of all the data objects in D and the domains of $C_1, C_2, ..., C_m$. It is assumed that if the guard evaluates to true on a value in its domain, then the effect is defined on that value. The sequence of outputs on out-channels of OC is a totally ordered finite sequence $(O_1, e_1), (O_2, e_2), ..., (O_p, e_p)$ of ordered pairs where O_i is any out-channel in OC and e_i is the value of an event, and is taken to mean that each event is written to its associated out-channel in the order they appear in the sequence. The effect must satisfy the following conditions:

1. Given a value v in the domain of the guard of a and thus in the domain of the effect of a, if the guard of a is true when evaluated on v, and v' is the result of evaluating the effect of a on v, then for any in-channel in IC that is one of the in-channels $C_1, C_2, ..., C_n$, if its value in v is $e_1, e_2, ..., e_p$ where it must be the case $p \geq 1$ since the guard of a evaluates to true, then its value in v' must be $e_2, ..., e_p$; and
2. given two values v_1 and v_2 in the domain of the guard of a, if the guard of a is true when evaluated on v_1 and v_2, and v_1' and v_2' are the results of evaluating the effect of a on v_1 and v_2 respectively, and the event at the head of C_i in v_1 has the same value as the event at the head of C_i in v_2 for each in-channel $C_1, C_2, ..., C_n$, and the value of all data objects in D are the same in v_1 and v_2, then:
    a. The value of any data object in D is the same in v_1' and v_2'; and
    b. the sequence of outputs on out-channels of OC in v_1' and v_2' are the same.

Definition 2.1.17. Specifying a *control state* for an action a belonging to an EPC M means specifying the internal state of M and the giving the values of the events at the heads of the in-channels belonging to M that it is predicated on.

We now describe what is a valid execution of the asynchronous product of extended state automaton in general. Intuitively, the asynchronous product of two extended state automata is another extended automaton consisting of all the data objects and actions of the two component automata. We can then reduce the extended automata that is the asynchronous product to a pure state automaton the execution of which is precisely defined. We will then treat a Flow model as the asynchronous product of extended state automaton in order to define its execution.

Definition 2.1.18. The *asynchronous product* of the extended automata $M_1 = \{S_1, D_1, L_1, T_1\}$ and $M_2 = \{S_2, D_2, L_2, T_2\}$ is another extended automaton $M = \{S, D, L, T\}$ such that $S = S_1 \times S_2$, $D = D_1 \cup D_2$, $L = L_1 \cup L_2$, $T \subseteq S \times L \times S$, and
$\forall((s_1, s_2), a, (s_1', s_2')) \in T : (a \in L_1 \wedge (s_1, a, s_1') \in T_1) \vee (a \in L_2 \wedge (s_2, a, s_2') \in T_2)$.

Definition 2.1.19. A *pure state automaton* is a tuple $\{S, T\}$ with S a set of *states*, and $T \subseteq S \times L \times S$ a set of labeled transitions where L is a set of labels.

Definition 2.1.20. Specifying a *state* for a pure state automaton means specifying a state from S.

Algorithm 2.1.21. To reduce an extended state automaton $\{S', D', L', T'\}$ to a pure state automaton:

1. Form a set of states for the pure state automaton by taking the cross product of states of the extended state automaton with the domains of each of its data objects $S' \times V_1 x V_2 x \ldots$. Note that each such state $(s, v_1, v_2, \ldots)$ gives a set of values $(v_1, v_2, \ldots)$ for all the data objects so that the guard of any action can be evaluated on the state and if true, the effect can be evaluated on the state; and
2. between all pairs of such states $(s, v_1, v_2, \ldots)$ and $(s', v_1', v_2', \ldots)$, create a transition labeled a if and only if there is a transition (s, a, s') in the extended state automaton, and the guard of action a evaluates to true on $(v_1, v_2, \ldots)$, and the result of evaluating the effect of a on $(v_1, v_2, \ldots)$ is $(v_1', v_2', \ldots)$.

Definition 2.1.22. An *execution fragment* of an extended state automaton is a sequence $s_0 a_1 s_1 a_2 s_2 \ldots$ where s_i is a state in its pure state automaton reduction and a_i is the label of a transition from s_i to s_{i+1} in the reduction. We call s_0 the *start state*. If the execution fragment is finite, we call the last state the *end state*. If the execution fragment is finite and contains exactly n actions, then we say that n is the *length* of the execution fragment. Otherwise we say the length of the execution fragment is infinite.

Definition 2.1.23. A finite execution fragment $s_0 a_1 s_1 a_2 s_2 \ldots s_n$ of an extended state automaton is *fair* if and only if there are no transitions out of s_n. An infinite execution fragment $s_0 a_1 s_1 a_2 s_2 \ldots$ is *fair* if there is no action whose guard evaluates to true on $s_k, s_{k+1}, \ldots$ for some $k \geq 0$ and is not one of the actions $a_{k+1}, a_{k+2}, \ldots$.

Definition 2.1.24. An *execution* $s_0 a_1 s_1 a_2 s_2 \ldots$ of an extended state automaton is any fair execution fragment of the automaton.

Definition 2.1.25. The *trace of an execution fragment* $s_0 a_1 s_1 a_2 s_2 \ldots$ is $a_1 a_2 \ldots$, the sequence of actions that appear in the execution fragment.

Definition 2.1.26. The *prefix of an execution fragment* $s_0 a_1 s_1 a_2 s_2 \ldots$ is another execution fragment $s_0 a_1 s_1 a_2 s_2 \ldots$ with the same sequence of states and actions whose length is less than or equal to the length of the original fragment.

We now interpret a Flow model as the asynchronous product of extended state automaton and define its execution as the execution of the extended state automaton. We first translate EPCs of the automaton into extended finite state automaton. Then we take the asynchronous product of the extended finite state automaton and say that the valid executions of the Flow model are exactly the valid executions of the asynchronous product.

Algorithm 2.1.27. To reduce an EPC $M = \{D, OC, IC, L\}$ in a Flow model F to an extended state automaton $\{S', D', L', T'\}$:

1. Take S' to contain exactly one state s; and
2. take $D' = D \cup OC \cup IC \cup$ {in-channels of EPCs in F such that there is a connection in F between an out-channel in OC and the in-channel}; and
3. For each action a in L, derive an action a' in L' as follows. Given a set of values v' for data objects in D', which is a value in the domain of a''s guard and effect, that implies a value v in the domain of a's guard and effect since it specifies a value of all the in-channels and data objects that belong to M. Let v'^* be the result of evaluating a''s effect on v' and v^* be the result of evaluating a's effect on v if they exist.
    a. a''s guard evaluates to true on v' if and only if a's guard evaluates to true on v. Thus the guard of a' is well defined.
    b. If a''s guard evaluates to true on v' and thus a's guard evaluates to true on v, then v'^* and v^* exist. For each data object that is neither an in-channel nor out-channel, the value of that data object in v'^* is identical to its value in v^*. For an in-channel C in D' that is not one of the in-channels that a is predicated on, if its value in v' is $e_1, e_2, ..., e_p$ where $p \geq 0$, then its value in v'^* is $e_1, e_2, ..., e_p, e_{p+1}, ..., e_{p+r}$, where $e_{p+1}, ..., e_{p+r}$ are the events in order of appearance in the subsequence of $(O_1, e_1), (O_2, e_2), ..., (O_p, e_p)$ in v^* containing all those ordered pairs for which there is a connection between the out-channel of the order pair and C. For an in-channel C in D' that is one of the in-channels that a is predicated on, if its value in v' is $e_1, e_2, ..., e_p$ where $p \geq 1$, then its value in v'^* is $e_2, ..., e_p, e_{p+1}, ..., e_{p+r}$, where $e_{p+1}, ..., e_{p+r}$ are events in order of appearance in the subsequence of $(O_1, e_1), (O_2, e_2), ..., (O_p, e_p)$ in v^* containing all those ordered pairs for which there is a connection between the out-channel of the order pair and C. Thus the effect of a' is well defined.
4. Let there be a transition (s, a', s) in T' if and only if a' is an action in L'.

Definition 2.1.28. An *execution fragment* $s_0 a_1 s_1 a_2 s_2 ...$ of a Flow model F is an execution fragment of the asynchronous product of the extended state automaton reductions of all the EPCs of F.

Definition 2.1.29. An *execution* $s_0 a_1 s_1 a_2 s_2 ...$ of a Flow model F is an execution of the asynchronous product of the extended state automaton reductions of all the EPCs of F.

Note that a state of the pure state automaton reduction of the extended state automaton reduction of F implies a state of Flow model F since it gives the value for all the data objects belonging to EPCs of F. Also, vice versa, the state of Flow model F

implies a state of the pure state automaton reduction of the extended state automaton reduction of F. From now on, both can be meant when we say the *state of* F.

Note that a state of F implies a local state for each EPC in F, an internal state for each EPC in F, and a control state for each action in F. To refer to the local state of an EPC in F implied by a state s of F, we say *the local state of the EPC in* s, and similarly for internal and control states.

We say that *an action is enabled in a state of* F if there is a transition out of that state labeled with the action in the pure state automaton reduction of the extended state automaton reduction of F. We say state s_i is the *result of executing* a_i *in* s_{i-1} if there is a transition (s_{i-1}, a_i, s_i) in the pure state automaton reduction of F.

Note that by the way we defined Flow actions and the way we translate them to actions of extended state automaton, several properties are true by definition which we explicitly state for convenience and reuse.

Lemma 2.1.30. For an EPC M in a Flow model F, and an action a belonging to M, it must be the case that the guard of a is *predicated on* the heads of a fixed finite set of in-channels belonging to M and the internal state of M. Specifically, associated with each action a must be a fixed finite set of in-channels $C_1, C_2, ..., C_n$ belonging to M such that:

1. Given a state of F, if the value of any of the in-channels $C_1, C_2, ..., C_n$ is such that the in-channel is empty in the state, then the action must not be enabled in the state; and
2. Given two states s_1 and s_2 of F, if none of the in-channels $C_1, C_2, ..., C_n$ are empty in s_1, and none of them are empty in s_2, and the event at the head of C_i in s_1 has the same value as the event at the head of C_i in s_2 for each in-channel $C_1, C_2, ..., C_n$, and the internal state of M is the same in s_1 and s_2 (this is the same as saying the control state of a is the same in s_1 and s_2), then a is enabled in s_1 if and only if it is enabled in s_2.

Furthermore it must be the case that a is *a function of* the control state of a. Precisely:

3. Given a state s of F, if a is enabled in s, and s' is the result of executing a in s, then:
    a. For any in-channel not connected to an out-channel belonging to M and is not one of the in-channels $C_1, C_2, ..., C_n$, the value of the in-channel must be the same in s and s'; and
    b. for any in-channel that is not connected to an out-channel belonging to M and is one of the in-channels $C_1, C_2, ..., C_n$, if its value in s is $e_1, e_2, ..., e_p$ where it must be the case $p \geq 1$ since a is enabled in s, then its value in s' must be $e_2, ..., e_p$; and
    c. for any in-channel that is connected to an out-channel belonging to M but is not one of the in-channels $C_1, C_2, ..., C_n$, if its value in s is $e_1, e_2, ..., e_p$, then its value in s' must be $e_1, e_2, ..., e_p, e_{p+1}, ..., e_{p+r}$, where $p, r \geq 0$; and

d. for any in-channel that is connected to an out-channel belonging to M and is one of the in-channels $C_1, C_2, ..., C_n$, if its value in s is $e_1, e_2, ..., e_p$ where it must be the case $p \geq 1$ since a is enabled in s, then its value in s' must be $e_2, ..., e_p, e_{p+1}, ..., e_{p+r}$, where $r \geq 0$.

4. Given two states s_1 and s_2 of F, if a enabled in both states, and s_1' and s_2' are the results of executing a in s_1 and s_2 respectively, and the control state of a is the same in s_1 and s_2, then:
   a. The internal state of M is the same in s_1' and s_2'; and
   b. if C_i and C_j are two in-channels such that C_i is connected to an out-channel of M if and only if C_j is connected to that out-channel, and neither C_i nor C_j are one of the in-channels $C_1, C_2, ..., C_n$, then we have that if the value of C_i is $e_1, e_2, ..., e_p$ in s_1 and $e_1, e_2, ..., e_p, e_{p+1}, ..., e_{p+r}$ in s_1' where $p, r \geq 0$, and the value of C_j is $e_1', e_2', ..., e_q'$ in s_2 where $q \geq 0$, then the value of C_j in s_2' is $e_1', e_2', ..., e_q', e_{p+1}, ..., e_{p+r}$; and
   c. if C_i and C_j are two in-channels such that C_i is connected to an out-channel of M if and only if C_j is connected to that out-channel, and both C_i and C_j are among the in-channels $C_1, C_2, ..., C_n$, then we have that if the value of C_i is $e_1, e_2, ..., e_p$ in s_1 where it must be the case $p \geq 1$ since a is enabled in s_1 and $e_2, ..., e_p, e_{p+1}, ..., e_{p+r}$ in s_1' where $r \geq 0$, and the value of C_j is $e_1', e_2', ..., e_q'$ in s_2 where $q \geq 1$ since a must be enabled in s_2, then the value of C_j in s_2' is $e_2', ..., e_q', e_{p+1}, ..., e_{p+r}$; and
   d. if C_i and C_j are two in-channels such that C_i is connected to an out-channel of M if and only if C_j is connected to that out-channel, and C_i is one of the in-channels $C_1, C_2, ..., C_n$ but C_j is not, then we have that the value of C_i is $e_1, e_2, ..., e_p$ in s_1 where it must be the case $p \geq 1$ since a is enabled in s_1 and $e_2, ..., e_p, e_{p+1}, ..., e_{p+r}$ in s_1' where $r \geq 0$ if and only if the value of C_j is $e_1', e_2', ..., e_q'$ in s_2 where $q \geq 0$ and the value of C_j in s_2' is $e_1', e_2', ..., e_q', e_{p+1}, ..., e_{p+r}$.

Proof:

True by the definition of Flow actions and their translation into actions of extended state automaton. ■

Lemma 2.1.31. For an EPC M and an action a of M, and another EPC M' possibly the same as M, and Flow models F_1 and F_2 possibly the same Flow model each containing both M and M', and in-channel C_1 of F_1 and in-channel C_2 of F_2 attached

to the same out-channels of M such that either both are in-channels that a is predicated on or both are not, and state s_1 of F_1 and s_2 of F_2 in which a is enabled, and state s_1' of F_1 and s_2' of F_2 which are the states resulting from executing a in s_1 and s_2 respectively, if the control state of a is the same in s_1 and s_2, and the internal state of M' is identical in s_1 and s_2, then the internal state of M' is identical in s_1' and s_2'. If the control state of a is identical in s_1 and s_2, and the state of C_1 in s_1 is identical to the state of C_2 in s_2, then the state of C_1 in s_1' is identical to the state of C_2 in s_2'. Lastly, if the control state of a is identical in s_1 and s_2, a executed in s_1 removes and adds the same events in the same sequence to C_1 as a executed in s_2 removes and adds to C_2.

Proof:

True by the definition of Flow actions and their translation into actions of extended state automaton. ■

Corollary 2.1.32. For a Flow model F, an action a of F enabled in two states s_1 and s_2 of F, and s_1' and s_2' the states of F resulting from executing a in s_1 and s_2 respectively, if the control state of a is the same in s_1 and s_2, and the internal state of an EPC or the state of an in-channel is the same in s_1 and s_2, then the internal state of that EPC or the state of that in-channel is the same in s_1' and s_2'.

Proof:

True by 2.1.31. ■

2.2 Fundamental Flow Model Properties And Constructs

We introduce some useful constructs and lemmas to use when reasoning about Flow models. We use the concept of concatenation of execution fragments to find executions of a Flow model later on.

Lemma 2.2.1. For a Flow model F with execution fragments $X_1 = s_0 a_1 s_1 ... s_n$ and $X_2 = s_n a_{n+1} s_{n+1} ...$, where X_1 is finite and the end state of X_1 is identical to the start state of X_2, the *concatenation* $X_1 \bullet X_2 = s_0 a_1 s_1 ... s_n a_{n+1} s_{n+1} ...$ of X_1 *and* X_2 is another execution fragment of F.

Proof:

By the definition of execution fragments, in $X_1 \bullet X_2$, every action a_i is enabled in s_{i-1} and s_i is the result of executing a_i in s_{i-1} because this is true for X_1 and X_2 separately. So $X_1 \bullet X_2$ is an execution fragment. ■

Later on, we describe ways to reason about Flow models by dividing them into pieces called sub-models. Intuitively, a sub-model is simply a subset of the EPCs in the full model and all the connections between them that appear in the full model. A state of the full model projected onto a sub-model consists of removing the states of the EPCs and in-channels not in the sub-model. An execution of the full model projected onto a sub-model consists of removing all the actions that do not belong to EPCs in the sub-model.

Definition 2.2.2. For a Flow model F, the Flow model F' is a *sub-model of* F and *induced from a set of EPCs* V' if and only if V' is a subset of the EPCs in F, and F' consists of EPC M of F if and only if $M \in V'$, and F' contains a connection between an out-channel of $M_1 \in V'$ and an in-channel of $M_2 \in V'$ if and only if there is such a connection in F. Note that sub-models are Flow models since they satisfy the definition of a Flow model.

Definition 2.2.3. For a Flow model F and a sub-model F' of F induced from V', the *projection of state* s *of* F *onto* F' is a state s' of F' such that the local state of each EPC in s' is identical to its corresponding local state in s.

Definition 2.2.4. For a Flow model F and a sub-model F' of F induced from V', the *projection of execution* $X = s_0 a_1 s_1 ...$ *of* F *onto* F' is an execution $X' = s_0' a_1' s_1' ...$ such that s_0' is the projection onto F' of the state immediately preceding the first action belonging to an EPC in V' to appear in X, a_i' is the i^{th} action belonging to an EPC in V' to appear in X, and s_i is the projection onto F' of the state immediately following the i^{th} action belonging to an EPC in V' that appears in X.

Note that the projection of execution X of F onto F' is not necessarily a legal execution of F'.

Lemma 2.2.5. For a Flow model F and a sub-model F' of F, if s' is a state of F', a is an action of an EPC M of F', and a state s of F projected onto F' equals s', then a is enabled in s if and only if it is enabled in s'. Suppose a is enabled in s' and s, let s'^* and s^* be the states resulting from executing a in s' and s respectively. Then the projection of s^* onto F' equals s'^*.

Proof:

By the definition of state projection, since the state of M and M 's in-channels are identical in s and s', and by the definition of Flow models, since a can only be predicated on these states, a is enabled in s if and only if it is enabled in s'.

Since a is enabled in s and s', and the state of M and M 's in-channels are identical in s and s', the control state of a is the same in s and s'. By the definition of projection, all the in-channels belonging to EPCs of F' are connected to the same out-channels of M as the corresponding EPCs of F and the internal state of all EPCs of F' and their in-channels are the same in s and s'. By lemma 2.1.31, the internal states of all EPCs of F' and the states of their in-channels are the same in $s*$ and $s'*$. This is the same as saying that the projection of $s*$ onto F' equals $s'*$. ■

Lemma 2.2.6. For a Flow model F and a sub-model F' of F, if $X' = s_0' a_1 s_1' ...$ is an execution fragment of F', and s_0 is a state of F such that s_0 projected onto F' equals s_0', then there exists a unique execution fragment $X = s_0 a_1 s_1 ...$ of F such that the i^{th} action is the same between the two executions and the projection of s_i onto F' equals s_i' for all i. We call X the *reverse projection of X' onto F from s_0*. Note that the projection of X onto F' equals X'.

Proof:

Consider action a_1 belonging to EPC M of F'. By lemma 2.2.5, a_1 must be enabled in s_0. Let s_1 be the result of executing a_1 in s_0. By lemma 2.2.5, the projection of s_1 onto F' equals s_1'. Repeating this with action a_2, we see that it is enabled in s_1 and that the projection of s_2 onto F' must be s_2'. We can repeat for all actions to show that the lemma holds. Note that the reverse projection must be unique because any other execution must have the same starting state s_0 and contain the same sequence of actions $a_1, a_2, ...$, so by definition of Flow models must contain the same sequence of states $s_1, s_2, ...$. ■

3. Reasoning About Flow Models

3.1 General Structure Of Proofs

Generally, we would like to prove that given a set of starting states of a Flow model as characterized by some boolean condition on the state of the model, all executions with those starting states have ending states that satisfy some other boolean condition. This runs along the lines of correctness proofs for computer programs in general stated in terms of pre-conditions and a post-conditions.

We first observe that most programs compute a specific ending state or output for a specific starting state or input. We first focus on building Flow models whose executions have this characteristic. A Flow model whose executions have this characteristic is said to be deterministically concurrent. We exploit deterministic concurrency later on to carry out correctness proofs in general.

Definition 3.1.1. A Flow model F is *deterministically concurrent* if and only all executions of F are finite and all executions of F that have the same start state have the same end state.

3.2 Action Independence And Execution Equivalence

We begin by defining what it means for two consecutive actions in an execution to be independent. Intuitively, they are independent if they can be executed in the reverse order, and the resulting state is the same. Thus is doesn't really matter which executes first and which second. This definition is similar to definitions of independence used in partial order reductions. [4, 6, 7, 13, 19]

Definition 3.2.2. Let $X = s_0 a_1 s_1 ... s_{i-1} a_i s_i a_{i+1} s_{i+1} ...$ be any execution of a Flow model F. Two consecutive actions a_i and a_{i+1} of an execution X are *independent* if and only if:

1. a_{i+1} is enabled in state s_{i-1} and a_i is enabled in the state that results from executing a_{i+1} in state s_{i-1}; and
2. the state resulting from executing a_{i+1} followed by a_i from state s_{i-1} is the same as s_{i+1}.

We use the concept of independence to define what it means for two executions to be equivalent. Intuitively, two executions are equivalent if they have the same start state and can be derived from each other by switching the order that independent actions within them are executed. The significance is that, since switching the order that independent actions are executed do not change the resulting state, two equivalent finite executions that have the same start state must have the same end state. We can use this to prove that

an entire group of executions that have the same start state have the same end state by proving a single execution has that end state and all other executions are equivalent to it.

Definition 3.2.3. Let $X = s_0 a_1 s_1 ... s_{i-1} a_i s_i a_{i+1} s_{i+1} ...$ be any execution of a Flow model F and let a_i and a_{i+1} be independent actions. We say that the execution $X' = s_0 a_1 s_1 ... s_{i-1} a_{i+1} s_i' a_i s_{i+1} ...$, with a_i and a_{i+1} switched, and s_i replaced by s_i', the state resulting from executing a_{i+1} in state s_{i-1}, is *derived by independence* from X. Note that derivation by independence, viewed as a relation between executions, is symmetric.

Lemma 3.2.4. If $X = s_0 a_1 s_1 ...$ is an execution of Flow model F, and $X' = s_0' a_1' s_1' ...$ is derived by independence from X with actions a_i and a_{i+1} of X switched, then X' is an execution of F. Furthermore, if X is infinite, so is X'. And if $X = s_0 a_1 s_1 ... s_n$ is finite, then $X' = s_0' a_1' s_1' ... s_n'$ is also finite, of the same length, and $s_n = s_n'$.

Proof:

To show that X' is an execution of F, we need to show that for all $1 \le j \le$ length of X', a_j' is enabled in s_{j-1}' and executing a_j' in s_{j-1}' results in s_j'. For $1 \le j \le i-1$ and $j \ge i+2$, this is true since $s_{j-1} = s_{j-1}'$, $a_j = a_j'$, $s_j = s_j'$ by definition of derived by independence, and a_j is enabled in s_{j-1} and executing a_j in s_{j-1} results in s_j since X is an execution of F. Furthermore, this is true for $j = i$ and $j = i+1$ by the definition of derived by independence.

If X is infinite, so is X' since we neither add to nor delete actions from X in deriving X'. This is sufficient to show that X' is an infinite execution of F if X is infinite.

If X' is finite, we also need to show that no actions are enabled in its end state. If $X = s_0 a_1 s_1 ... a_n s_n$ is finite, X' must be finite and of the same length since we neither add nor delete actions to X in deriving X'. Also, note that $n > i$, so by the reasoning above $s_n = s_n'$, which implies there are no actions enabled in s_n' since this is the case for s_n, and X' is a finite execution of F with $s_n = s_n'$ of the same length as X.

Last of all, we need to show that X' is fair. Let action a be enabled in s_j'. We need to show that it is never the case that a does not appear in X' after s_j' and is enabled in $s_{j+1}', s_{j+2}', ...$. Suppose there is such an action. Then it must be enabled in $s_{i+1}', s_{i+2}', ...$ and does not appear in X' after s_{i+1}'. However, since $s_{i+k}' = s_{i+k}$ and $a_{i+k+1}' = a_{i+k+1}$ for $k \ge 1$, this would mean that a is enabled in $s_{i+1}, s_{i+2}, ...$ and does not appear in X after s_{i+1}. This would imply X is not a fair, which is a contradiction. ■

Definition 3.2.5. Let X be any execution of a Flow model F. We say that X' is *equivalent by independence* to X provided that there is a finite sequence of executions

$X_1, X_2, ..., X_n$ such that $X = X_1$, $X_n = X'$, and X_{i+1} is derived by independence from X_i for all $1 \le i \le n-1$. Note that equivalence by independence, viewed as a relation between executions, is reflexive, symmetric and transitive.

Lemma 3.2.6. If X is an execution of Flow model F, and X' is equivalent by independence to X, then X' is an execution of F. Furthermore, if X is infinite, so is X'. And if $X = s_0 a_1 s_1 ... s_n$ is finite, then $X' = s_0' a_1' s_1' ... s_n'$ is also finite, of the same length, and $s_n = s_n'$.

Proof:

Let $X_1, X_2, ..., X_m$ be the finite sequence of executions such that $X = X_1$, $X_m = X'$, and X_{i+1} is derived by independence from X_i for all $1 \le i \le m-1$. Since X_2 is derived by independence from X, it must be an execution of F by lemma 3.2.4. Furthermore, X_2 is infinite if X is infinite, and X_2 is finite with the same length and same end state if X is finite. Similarly, X_3 must have all these same properties since it is derived by independence from X_2. Repeating this reasoning for $X_3, ..., X_m$, this shows that $X_m = X'$ must be a fair execution of F that is infinite if X is infinite, and finite with the same length and end state if X is finite. ■

We want to construct or identify Flow models that are deterministically concurrent. The significance of this is that to perform the task of finding the end state of an execution corresponding to a specific start state for such a model, all we will need to do is consider one feasible execution and we are guaranteed all other executions from the same start state will have the same end state. Thus, performing this typical task on a concurrent model of computation becomes no more difficult than performing the same task on a serial model.

We begin by introducing two conditions which, if satisfied by a Flow model in which all executions are finite, guarantees that it is deterministically concurrent. Intuitively, the enable immutability condition requires that once an action is enabled, it cannot be disabled by any other action until it executes. The enable independence condition requires that if two actions are enabled at the same time, then it doesn't matter which executes first, the resulting state will be the same. As we will see later on, it will be easy to construct or identify Flow models that satisfy these conditions.

Definition 3.2.7. An action a of Flow model F is *enable immutable* if and only if for all executions of F, if a is enabled in state s_i, then it remains enabled in $s_i, s_{i+1}, ...$ for as long as it is not executed.

Definition 3.2.8. An action a of Flow model F is *enable independent* if and only if for all executions of F, if a is the $(i+1)^{th}$ action, a' the i^{th} action, and a is enabled in state s_{i-1}, then a and a' are independent.

Theorem 3.2.9. If all executions of Flow model F are finite and all actions of F are both enable immutable and enable independent, then F is deterministically concurrent.

Proof:

Consider any two finite executions $X = s_0 a_1 s_1 ...$ and $X' = s_0 a_1' s_1' ...$ of F having the same start state. We prove that X is equivalent by independence to X' using induction on the length of the prefix of X'.

In the base case, consider the prefix of X' of length one: $P_1 = s_0 a_1' s_1'$. We want to show that there is a sequence of executions $X_1, X_2, ..., X_n$ each derived by independence from the previous one with $X = X_1$ such that P_1 is a prefix of X_n. P_1 tells us that a_1' is enabled in s_0, so by enable immutability and fairness, a_1' must execute at some point in X. Say that a_1' first occurs as the i^{th} action of X. If $i = 1$, then we are done since P_1 is already a prefix of X. If $i > 1$, a_1' must be enabled in state s_{i-2} by enable immutability, and therefore must be independent with the action that precedes it by enable independence. Thus, there is an execution X_2 of F that is derived by independence from X with a_1' as the $(i-1)^{th}$ action. By repeated application of this reasoning, there is a sequence $X_1, X_2, ..., X_n$ each derived by independence from the previous one with $X = X_1$ such that a_1' as the first action of X_n, and consequentially P_1 is a prefix of X_n.

For the inductive hypothesis, assume that for a prefix $P_k = s_0 a_1' s_1' ... s_k'$ of length k, there is a sequence of executions $X_1, X_2, ..., X_n$ each derived by independence from the previous one with $X = X_1$ such that P_k is a prefix of X_n. For the inductive case, we want to show that this is true for a prefix $P_{k+1} = s_0 a_1' s_1' ... s_k' a_{k+1}' s_{k+1}'$ of length $k+1$.

By our inductive hypothesis, there exists an execution $X_n = s_0 a_1' s_1' ... a_k' s_k' ...$ of F that has P_k as a prefix and can be derived from X by a sequence of executions each derivable by independence from the previous. Consider action a_{k+1}'. P_{k+1} tells us that a_{k+1}' is enabled in s_k', so by enable immutability and fairness, a_{k+1}' must execute in X_n after s_k'. Say that a_{k+1}' first occurs as the i^{th} action of X_n, $i > k$. If $i = k+1$, then we are done since X_{k+1} is already a prefix of X_n and X_n can be derived from X by a sequence of executions each derivable by independence from the previous. If $i > k+1$, a_{k+1}' must be enabled in state s_{i-2} by enable immutability, and therefore must be independent with the action that precedes it by enable independence. Thus, there is an execution $X_{n,2}$ of F that is derived by independence from X_n with a_{k+1}' as the $(i-1)^{th}$ action. By repeated application of this reasoning, there is a sequence $X_n, X_{n,2}, ..., X_{n,m}$ each derived by independence from the previous one such that a_{k+1}' is the $(k+1)^{th}$ action of $X_{n,m}$, and consequentially P_{k+1} is a prefix of $X_{n,m}$.

Since X_n can be derived from X by a sequence of executions each derivable by independence from the previous and $X_{n,m}$ can be derived from X_n by a sequence of executions each derivable by independence from the previous, it is clear that $X_{n,m}$ can be derived from X by a sequence of executions each derivable by independence from the previous. Furthermore, since P_{k+1} is a prefix of $X_{n,m}$, the inductive case is proven.

This shows that for any prefix of X', there is an execution X_n equivalent by independence to X that has the same prefix. The exact same reasoning applies with the roles of X and X' in reverse to show that for any prefix of X, there is an execution X_n equivalent by independence to X' that has the same prefix. This implies that X and X' have the same length since if one is longer, taking itself as the prefix, there is no way it can be a prefix of an execution that is equivalent by independence to the other, which must have the same length as the other by lemma 3.2.6.

Take the prefix of X' to be X' itself, which we can do since we know X' is finite. There is an execution X_n equivalent by independence to X that has X' as the prefix. Since X_n has the same length as X by lemma 3.2.6, it has the same length as X', and so must be identical to X'. This shows that X and X' are equivalent by independence, which implies they have the same end state by lemma 3.2.6. This shows that any two executions of F from the same start state have the same end state, thus F is deterministically concurrent. ■

At a glance, it might seem difficult to verify that an action is enable immutability and independence. The worst scenario would be if every two consecutive actions in every execution of a Flow model must be checked to verify that all actions of the model are enable immutable and independent. This would make theorem 3.2.9 useless in practice for identifying Flow models that are deterministically concurrent.

Fortunately, there are easier ways to show that an action is enable immutable and independent. Intuitively, we show that in a Flow model, as long as an action that is enabled cannot be disabled by an action belonging to the same EPC, then it is enable immutable in relation to all actions. We also show that as long as an action is enable independent in relation to actions belonging to the same EPC or EPCs that output to the same in-channels as its EPC, then it is enable independent in relation to all actions. Thus, we are able to check that an action is enable immutable and independent by comparing it against far fewer than the total number of actions in the model.

Definition 3.2.10. In a Flow model F, an action a of an EPC M is *locally enable immutable* if and only if for all executions of F, a is enabled in state s_{i+1} if it is enabled in s_i and a_{i+1} is a different action of M.

Lemma 3.2.11. In an execution $X = s_0 a_1 s_1 ... s_i a_{i+1} s_{i+1} ...$ of a Flow model F, if an action a of EPC M is enabled in state s_i and a_{i+1} does not affect the internal state of M and

does not remove events from M 's in-channels, then a is still enabled in s_{i+1}. Similarly, if a is disabled in state s_i and a_{i+1} does not affect the internal state of M and does not add or remove events from M 's in-channels, then a is still disabled in s_{i+1}.

Proof:

By definition of Flow models, an action a of EPC M is predicated on the events at the heads of a fixed number of in-channels $C_1, C_2, ..., C_n$ of M and the internal state of M. More precisely, an event at the head of each of the in-channels $C_1, C_2, ..., C_n$ is necessary for a to be enabled , and if in two states of the Flow model, the internal state of M in each state is identical, and in each state there is an event at the head of each in-channel $C_1, C_2, ..., C_n$, and in each state the corresponding events at the head of the in-channels are identical, then a is enabled in one state if and only if a is enabled in the other state.

If a is enabled in s_i, there must be an event at the head of each in-channel $C_1, C_2, ..., C_n$. Since a_{i+1} does not affect the internal state of M and does not remove events from M 's in-channels, the internal state of M in s_{i+1} must be the same, and there is an event at the head of each in-channel $C_1, C_2, ..., C_n$, and that event must be the same as the corresponding event in s_i. Thus, by definition of Flow models, a must be still enabled in s_{i+1} since it is enabled in s_i.

If a is disabled in s_i, either one of the in-channels $C_1, C_2, ..., C_n$ is empty, or there is an event at the head of each in-channel $C_1, C_2, ..., C_n$. In the first case, any in-channel $C_1, C_2, ..., C_n$ that is empty in s_i must still be empty in s_{i+1} since a_{i+1} does not add to the in-channels of M, so a is still disabled in s_{i+1}. In the second case, since a_{i+1} does not affect the internal state of M and does not remove events from M 's in-channels, the internal state of M in s_{i+1} must be the same, and there is an event at the head of each in-channel $C_1, C_2, ..., C_n$, and that event must be the same as the corresponding event in s_i. Thus, by definition of Flow models, a must be still disabled in s_{i+1} since it is disabled in s_i. ■

Corollary 3.2.12. In an execution $X = s_0 a_1 s_1 ... s_i a_{i+1} s_{i+1} ...$ of a Flow model F, if an action a of EPC M is enabled in state s_i and a_{i+1} is not an action of M, then a is still enabled in s_{i+1}. Similarly, if a is disabled in state s_i and a_{i+1} is not an action of M and does not belong to an EPC that has an out-channel connected to an in-channel of M, then a is still disabled in s_{i+1}.

Proof:

By definition of Flow models, if a_{i+1} is not an action of M then a_{i+1} cannot affect M 's internal state, nor can it remove events from M 's in-channels. So, by lemma 3.2.11, if a is enabled in s_i, then it is still enabled in s_{i+1}. If a_{i+1} is not an action of M and

does not belong to an EPC that has an out-channel connected to an in-channel of M, then a_{i+1} cannot affect M's internal state, nor can it add or remove events from M's in-channels. So, by lemma 3.2.11, if a is disabled in s_i, a is still disabled in s_{i+1}. ■

Theorem 3.2.13. In a Flow model F, if action a of EPC M is locally enable immutable, then a is enable immutable.

Proof:

Suppose a is enabled in state s_i. If a is the $(i+1)^{th}$ action, then the condition for enable immutability is satisfied. If a_{i+1} is an action of M other than a, then a is enabled in s_{i+1} by local enable immutability. If a_{i+1} is not an action of M, by corollary 3.2.12, a is still enabled in s_{i+1}. By repeated application of this reasoning, a must be enabled in $s_i, s_{i+1}, ...$ for as long as it is not executed, and thus is enable immutable. ■

Definition 3.2.14. In a Flow model F, an action a of an EPC M is *locally enable independent* if and only if for all executions of F, if a is the $(i+1)^{th}$ action, a' is the i^{th} action and belongs either to M or a different EPC M' such that there is an out-channel of M and an out-channel of M' connected to the same in-channel, and a is enabled in state s_{i-1}, then a and a' are independent.

Lemma 3.2.15. In an execution $X = s_0 a_1 s_1 ... s_i a_{i+1} s_{i+1} ...$ of a Flow model F, if an action a of EPC M is enabled in state s_i and a_{i+1} does not affect the internal state of M and does not remove events from M's in-channels, then in the two states resulting from executing a in s_i and executing a in s_{i+1} contains identical states for the internal state of M. Furthermore a executed in s_i adds a sequence of events $e_1 e_2 ... e_n$ to an in-channel C if and only if a executed in s_{i+1} adds the same sequence of events on C.

Proof:

By definition of Flow models, an action a of EPC M is function of the events at the head of the same in-channels $C_1, C_2, ..., C_n$ of M that it is predicated on, the internal state of M. More precisely, let s_1 and s_2 be two states in which a is enabled and let s_1' and s_2' be the resulting states from executing a in s_1 and s_2 respectively. If the internal state of M in s_1 and in s_2 are identical, and the corresponding events at the heads of $C_1, C_2, ..., C_n$ are identical in s_1 and in s_2, then the internal state of M in s_1' and s_2' are identical. Also, under the same assumptions, if C' is an in-channel connected to an out-channel of M and it is not itself one of the in-channels $C_1, C_2, ..., C_n$ and C''s state in s_1 is $e_1 e_2 ... e_p$ and C''s state in s_2 is $e_1' e_2' ... e_q'$, then C''s state in s_1' is $e_1 e_2 ... e_p e_{p+1} ... e_{p+r}$ and C''s state in s_1' is $e_1' e_2' ... e_q' e_{p+1} ... e_{p+r}$ where $r \geq 0$. Also under the same assumptions, if C' is an in-channel connected to an out-channel of M and it is one of the

in-channels $C_1, C_2, ..., C_n$ and C''s state in s_1 is $e_1 e_2 ... e_p$ and C''s state in s_2 is $e_1' e_2' .. e_q'$, then C''s state in s_1' is $e_2 ... e_p e_{p+1} ... e_{p+r}$ and C''s state in s_1' is $e_2' .. e_q' e_{p+1} ... e_{p+r}$ where $r \geq 0$. Also under the same assumptions, if C' is one of the in-channels $C_1, C_2, ..., C_n$ and not connected to an out-channel of M and C''s state in s_1 is $e_1 e_2 ... e_p$ and C''s state in s_2 is $e_1' e_2' .. e_q'$, then C''s state in s_1' is $e_2 ... e_p$ and C''s state in s_2' is $e_2' .. e_q'$. Also, if C' is neither one of the in-channels $C_1, C_2, ..., C_n$ nor connected to an out-channel of M, then C''s state is identical in s_1 and s_1', and in s_2 and s_2'. Lastly, for EPC M' different from M, the internal state of M' is identical in s_1 and s_1', and in s_2 and s_2'.

Since a is enabled in s_i, there must be an event at the head of each in-channel $C_1, C_2, ..., C_n$. Since a_{i+1} does not affect the internal state of M and does not remove events from M's in-channels, the internal state of M in s_{i+1} must be the same, and there is an event at the head of each in-channel $C_1, C_2, ..., C_n$, and that event must be the same as the corresponding event in s_i. By lemma 2.1.30, a is also enabled in s_{i+1}. So by definition of Flow models, all of the above must be true for the two states resulting from executing a in s_i and executing a in s_{i+1}. This is the same as saying that the two states resulting from executing a in s_i and executing a in s_{i+1} contain identical states for the internal state of M, and that a executed in s_i adds a sequence of events $e_1 e_2 ... e_n$ to an in-channel C if and only if a executed in s_{i+1} adds the same sequence of events on C. ■

Corollary 3.2.16. In an execution $X = s_0 a_1 s_1 ... s_i a_{i+1} s_{i+1} ...$ of a Flow model F, if an action a of EPC M is enabled in state s_i and a_{i+1} does not belong to M, then in the two states resulting from executing a in s_i and executing a in s_{i+1} contains identical states for the internal state of M. Furthermore a executed in s_i adds a sequence of events $e_1 e_2 ... e_n$ to an in-channel C if and only if a executed in s_{i+1} adds the same sequence of events to C.

Proof:

By definition of Flow models, if a_{i+1} is not an action of M, then a_{i+1} cannot affect M's internal state, nor can it remove events from M's in-channels. So, by lemma 3.2.15, the two states resulting from executing a in s_i and executing a in s_{i+1} contains identical states for the internal state of M. Furthermore a executed in s_i adds a sequence of events $e_1 e_2 ... e_n$ to an in-channel C if and only if a executed in s_{i+1} adds the same sequence of events on C. ■

Theorem 3.2.17. In an execution $X = s_0 a_1 s_1 ... s_{i-1} a_i s_i a_{i+1} s_{i+1} ...$ of a Flow model F, if an action a_{i+1} of EPC M is enabled in state s_{i-1} and a_i belongs to a different EPC M'

such that none of M''s out-channels are connected to the same in-channels as M's out-channels, then a_i and a_{i+1} are independent.

Proof:

Let s_i' represent the state resulting from executing a_{i+1} in s_{i-1}. By corollary 3.2.12, since a_{i+1} does not belong to the same EPC as a_i, a_i must be enabled in s_i'. Let s_{i+1}' represent the state resulting from executing a_{i+1} followed by a_i in s_{i-1}. We must show that s_{i+1}' is identical to s_{i+1}.

By corollary 3.2.16, the internal state of M' is identical in s_{i+1}' as in s_i. Since a_{i+1}, belonging to a different EPC, does not affect the internal state of M' by the definition of Flow models, it must be the case that the internal state of M' is identical in s_i as in s_{i+1}. Thus, the internal state of M' is identical in s_{i+1}' as in s_{i+1}. Similarly, by corollary 3.2.16, the internal state of M is identical in s_i' as in s_{i+1}. Since a_i, belonging to a different EPC, does not affect the internal state of M by the definition of Flow models, it must be the case that the internal state of M is identical in s_i' as in s_{i+1}'. Thus, the internal state of M is identical in s_{i+1}' as in s_{i+1}. For the internal states of all other EPCs, since by the definition of Flow models, neither a_i nor a_{i+1} affects them, they must be identical in s_{i+1}' as in s_{i-1}, and identical in s_{i+1} and s_{i-1}, and therefore identical in s_{i+1}' and s_{i+1}.

Now the only states left to consider are the states of internal channels. There are nine possibilities. Each in-channel can be either connected to out-channels of M, connected to out-channels of M', or connected to neither. Note that being connected to both is not an option by out assumptions. Also, each in-channel can be either one of the in-channels of M' that a_i is predicated on, one of the in-channels of M that a_{i+1} is predicated on, or neither. The cross product gives nine possibilities for in-channels.

Suppose that an in-channel is not connected to M or M''s out-channels and is not one of the channels that either a_i or a_{i+1} is predicated on. Then, by the definition of Flow models, it is not affected by either action. Therefore, the state of the in-channel is identical in s_{i+1} and s_{i-1}, in s_{i+1}' and s_{i-1}, and consequently in s_{i+1} and s_{i+1}'.

Suppose that an in-channel is not connected to M or M''s out-channels and is one of the in-channels that a_i is predicated on. Suppose $e_1e_2...e_n$ is its state in s_{i-1}. By definition of Flow models, its state in s_i is $e_2...e_n$ and its state in s_{i+1} is $e_2...e_n$ since a_i removes the first event and otherwise does not add events and a_{i+1} does not add events. Since a_{i+1} does not add events, its state in s_i' is $e_1e_2...e_n$ and its state in s_{i+1}' is $e_2...e_n$ since a_i removes the first event and does not add events. So its state is identical in s_{i+1} and s_{i+1}'. Similarly, suppose that an in-channel is not connected to M or M''s out-channels and is one of the in-channels that a_{i+1} is predicated on. Suppose $e_1e_2...e_n$ is its state in s_{i-1}. By definition of Flow models, its state in s_i' is $e_2...e_n$ and its state in s_{i+1}' is $e_2...e_n$ since a_{i+1} removes the first event and otherwise does not add events and a_i does not add events. Since a_i does not add events, its state in s_i is $e_1e_2...e_n$ and its state in s_{i+1} is $e_2...e_n$ since

a_{i+1} removes the first event and does not add events. So its state is identical in s_{i+1} and s_{i+1}'.

Suppose that an in-channel is connected to one or more out-channels of M but not M' and is neither predicated on by a_i nor a_{i+1}. Suppose $e_1e_2...e_n$ is its state in s_{i-1}. Since a_i neither adds nor removes events from it, its state in s_i is $e_1e_2...e_n$. a_{i+1} may add but cannot remove events from it, so its state in s_{i+1} is $e_1e_2...e_ne_{n+1}...e_{n+r}$ where $r \geq 0$, with $e_{n+1}...e_{n+r}$ the sequence of events added by a_{i+1}. By corollary 3.2.16, a_{i+1} executed in s_{i-1} adds the same events to the in-channel, so its state in s_i' is $e_1e_2...e_ne_{n+1}...e_{n+r}$, and its state in s_{i+1}' is the same since a_i cannot affect it. So the state of the in-channel is identical in s_{i+1} and s_{i+1}'. Similarly, suppose that an in-channel is connected to one or more out-channels of M' but not M and is neither predicated on by a_i nor a_{i+1}. Suppose $e_1e_2...e_n$ is its state in s_{i-1}. Since a_{i+1} neither adds nor removes events from it, its state in s_i' is $e_1e_2...e_n$. a_i may add but cannot remove events from it, so its state in s_{i+1}' is $e_1e_2...e_ne_{n+1}...e_{n+r}$ where $r \geq 0$, with $e_{n+1}...e_{n+r}$ the sequence of events added by a_i. By corollary 3.2.16, a_i executed in s_{i-1} adds the same events to the in-channel, so its state in s_i is $e_1e_2...e_ne_{n+1}...e_{n+r}$, and its state in s_{i+1} is the same since a_{i+1} cannot affect it. So the state of the in-channel is identical in s_{i+1} and s_{i+1}'.

Suppose that an in-channel is connected to one or more out-channels of M but not M' and is predicated on by a_i. Suppose $e_1e_2...e_n$ is its state in s_{i-1}. Since a_i does not add but must remove the head event from it, its state in s_i is $e_2...e_n$. a_{i+1} may add but cannot remove events from it, so its state in s_{i+1} is $e_2...e_ne_{n+1}...e_{n+r}$ where $r \geq 0$, with $e_{n+1}...e_{n+r}$ the sequence of events added by a_{i+1}. By corollary 3.2.16, a_{i+1} executed in s_{i-1} adds the same events to the in-channel, so its state in s_i' is $e_1e_2...e_ne_{n+1}...e_{n+r}$, and a_i does not add but must remove the head event from it so its state in s_{i+1}' is $e_2...e_ne_{n+1}...e_{n+r}$. So the state of the in-channel is identical in s_{i+1} and s_{i+1}'. Similarly, suppose that an in-channel is connected to one or more out-channels of M' but not M and is predicated on by a_{i+1}. Suppose $e_1e_2...e_n$ is its state in s_{i-1}. Since a_{i+1} does not add but must remove the head event from it, its state in s_i' is $e_2...e_n$. a_i may add but cannot remove events from it, so its state in s_{i+1}' is $e_2...e_ne_{n+1}...e_{n+r}$ where $r \geq 0$, with $e_{n+1}...e_{n+r}$ the sequence of events added by a_i. By corollary 3.2.16, a_i executed in s_{i-1} adds the same events to the in-channel, so its state in s_i is $e_1e_2...e_ne_{n+1}...e_{n+r}$, and a_{i+1} does not add but must remove the head event from it so its state in s_{i+1} is $e_2...e_ne_{n+1}...e_{n+r}$. So the state of the in-channel is identical in s_{i+1} and s_{i+1}'.

Suppose that an in-channel is connected to one or more out-channels of M but not M' and is predicated on by a_{i+1}. Suppose $e_1e_2...e_n$ is its state in s_{i-1}. Since a_i cannot affect its state, its state in s_i is $e_1e_2...e_n$. a_{i+1} may add events to it and must remove the

event at the head of it, so its state in s_{i+1} is $e_2...e_n e_{n+1}...e_{n+r}$ where $r \geq 0$, with $e_{n+1}...e_{n+r}$ the sequence of events added by a_{i+1}. By corollary 3.2.16, a_{i+1} executed in s_{i-1} adds the same events to the in-channel and also must remove the event at the head, so its state in s_i' is $e_2...e_n e_{n+1}...e_{n+r}$, and a_i does not affect it, so its state in s_{i+1}' is the same. So the state of the in-channel is identical in s_{i+1} and s_{i+1}'. Similarly, suppose that an in-channel is connected to one or more out-channels of M' but not M and is predicated on by a_i. Suppose $e_1 e_2...e_n$ is its state in s_{i-1}. Since a_{i+1} cannot affect its state, its state in s_i' is $e_1 e_2...e_n$. a_i may add events to it and must remove the event at the head of it, so its state in s_{i+1}' is $e_2...e_n e_{n+1}...e_{n+r}$ where $r \geq 0$, with $e_{n+1}...e_{n+r}$ the sequence of events added by a_i. By corollary 3.2.16, a_i executed in s_{i-1} adds the same events to the in-channel and also must remove the event at the head, so its state in s_i is $e_2...e_n e_{n+1}...e_{n+r}$, and a_{i+1} does not affect it, so its state in s_{i+1} is the same. So the state of the in-channel is identical in s_{i+1} and s_{i+1}'.

The above shows that the state of any data object is identical in s_{i+1} and s_{i+1}', and so s_{i+1} and s_{i+1}' are identical states. This implies that a_i and a_{i+1} are independent. ■

Theorem 3.2.18. In a Flow model F, if an action a is locally enable independent, then it is enable independent.

Proof:

Suppose a is the $(i+1)^{th}$ action in an execution $X = s_0 a_1 s_1 ... s_{i-1} a_i s_i a_{i+1} s_{i+1} ...$ of F and is enabled in state s_{i-1}. If a' is the i^{th} action and an action of M or a different EPC that has an out-channel that is connected the same in-channel as one of M's out-channels, then a and a' are independent by the local enable independence assumption.

If a' is the i^{th} action and belongs to M' not equal to M and none of M''s out-channels are connected to the same in-channels as M's out-channels, then by theorem 3.2.17, a and a' are independent.

In all cases a and a' are independent, so a is enable independent. ■

Theorem 3.2.19. In a Flow model F, if all executions are finite and all actions are locally enable immutable and independent, then F is deterministically concurrent.

Proof:

By theorems 3.2.13 and 3.2.18, all actions of F are enable immutable and enable independent. By theorem 3.2.9, F is deterministically concurrent. ■

Theorem 3.2.19 essentially allows us to verify that an action is enable immutable and independent by comparing it only against other actions of the same EPC and other EPCs that output to the same in-channels. In essence, this allows us to verify that a Flow model

is deterministically concurrent by considering each EPC individually, or at most along with other EPCs that output to the same in-channels.

We seek to make verifying deterministic concurrency even easier by introducing simple conditions that are sufficient for local enable immutability and independence. Intuitively, if an EPC has the property that no more than one of its actions can be enabled at a time, then its actions are trivially enable immutable and independent with regards to themselves. If we also restrict the model so that no in-channel is written to by more than one out-channel, then the actions would be enable immutable and independent with regards to all actions, and thus the model is deterministically concurrent as long as all executions are finite.

Definition 3.2.20. An EPC M is *deterministic* if and only if for each unique state of M, consisting of its internal state and the state of all of its in-channels, no more than one action of M is enabled.

Lemma 3.2.21. In a Flow model F, if an EPC M is deterministic, then all its actions are enable immutable.

Proof:

Suppose an action a of M is enabled in state s_{i+1} of an execution X of F. It cannot be the case that a is also enabled in state s_i and action a_{i+1} is a different action of M since that would imply two actions of M are enabled in s_i, contradicting the assumption that M is deterministic. Thus, a is trivially locally enable immutable, and by theorem 3.2.13 is enable immutable. ■

Lemma 3.2.22. In a Flow model F, if an EPC M is deterministic and has no out-channels that are connected to in-channels that out-channels of other EPCs also connect to, then all its actions are enable independent.

Proof:

Suppose an action a of M is enabled in state s_{i+1} of an execution X of F. It cannot be the case that a is also enabled in state s_i and action a_{i+1} is a another action of M since that would imply two actions of M are enabled in s_i, contradicting the assumption that M is deterministic. Also, trivially, a_{i+1} cannot be an action of a different EPC that writes to the same in-channels as M since there are no such EPCs by assumption. Thus, a is trivially locally enable independent, and by theorem 3.2.18 is enable independent. ■

Theorem 3.2.23. In a Flow model F, if all executions are finite, all EPCs are deterministic, and no in-channels are connected to by more than one out-channel, then F is deterministically concurrent.

Proof:

Any action is enable immutable by lemma 3.2.21 and enable independent by lemma 3.2.22. Thus, by theorem 3.2.9, F is deterministically concurrent. ■

The above allows us to construct or verify deterministically concurrent Flow models rather easily as long as we know all their executions are finite. The only conditions we need to satisfy are that no in-channel is connected to by more than one out-channel and each EPC is deterministic. Alternatively, an EPC does not need to be deterministic as long as all actions belonging to the same EPC can be verified to be enable immutable and independent relative to each other. We will show below an easily checked condition for all executions to be finite.

3.3 Graph Representations Of Flow Models

It is intuitive and useful to conceptualize a Flow model as a graph. Useful sufficient conditions for deterministic concurrency can be stated in terms of graph properties.

Intuitively, the graph representation of a Flow model consists of a vertex for each EPC in the graph, and a directed edge from vertex A to B if there is a connection from an out-channel of EPC A to EPC B. The edge is labeled with the name of the out-channel and in-channel.

Definition 3.3.1. The *graph* $G(V,E)$ *of Flow model* F consisting of only EPCs is a labeled directed graph where vertex $v \in V$ if and only if there is an EPC in F named v, and edge $(v_1, v_2) \in E$ labeled (l_1, l_2) if and only if there is a connection from an out-channel named l_1 of an EPC in F named v_1 to an in-channel named l_2 of an EPC in F named v_2.

There are useful ways to modularize reasoning about a complicated Flow model using divide and conquer. The division of a Flow model into smaller models can be stated in terms of a unidirectional cut of the model's graph. Intuitively, a unidirectional cut of a directed graph partitions the vertices into two sets such that all edges that go between sets run in one direction. Each of the resulting two pieces then represents a smaller Flow model which can then be reasoned about individually.

Definition 3.3.2. A *unidirectional cut* of a graph $G(V,E)$ is a cut of G into two sets V_1 and V_2 such that any cut edge $(v_1, v_2) \in E$ has $v_1 \in V_1$ and $v_2 \in V_2$.

For convenience, we say *an EPC of* V to mean an EPC which is represented by a vertex in set V. Also, we write *a sub-model induced by* $V' \subseteq V$ to mean the sub-model induced by the set of EPCs represented by vertices in V'.

3.4 Graph Acyclicness And Finiteness Of Executions

In order to verify that a Flow model is deterministically concurrent, we must verify that all its executions are finite. We show that by dividing a Flow model into two smaller sub-models via a unidirectional cut, the full model can be shown to have only finite executions by showing that each sub-model has only finite executions. We also show that acyclicness of a Flow model's graph is a sufficient condition for all executions to be finite.

Lemma 3.4.1. For a Flow model F with graph $G(V,E)$ that can be unidirectional cut into V_1 and V_2 such that edges are directed from V_1 to V_2 and no in-channel is connected to both an out-channel from an EPC of V_1 and an out-channel from an EPC of V_2, if X is an execution of F with more than or equal to m actions belonging to EPCs of V_1, then there is an execution $X' = s_0 a_1 s_1 ... s_{m-1} a_m s_m ...$ equivalent by independence to X such that a_i is the i^{th} action belonging to an EPC of V_1 to appear in X for $1 \le i \le m$.

Proof:

Consider the first action a belonging to an EPC M of V_1 to appear in X. If it is not the first action of X, then the previous action a' must belong to an EPC in V_2. Say that a' is the j^{th} action and a is the $(j+1)^{th}$. Since there are no connections from out-channels of EPCs in V_2 to in-channels of EPCs in V_1 by the definition of unidirectional cuts, a is enabled in s_{j-1} if and only if it is enabled in s_j by corollary 3.2.12. So both a and a' are enabled in s_{j-1}, belong to different EPCs, and the EPCs do not write to the same in-channels by assumption, which means a and a' are independent by theorem 3.2.17. So, by using derivation by independence to switch a and a' around, we see that X is equivalent by independence to an execution where the first action belonging to an EPC of V_1 appears as the j^{th} action. By repeated application of this reasoning, we see that X is equivalent by independence to an execution where the first action belonging to an EPC of V_1 appears as the first action. Now consider the second action belonging to an EPC of V_1 to appear in this new execution. Since all actions between it and the first belong to EPCs in V_2, we can use the same technique as above to show that this new execution, and thus X, is equivalent by independence to another execution that has the first and second actions belonging to EPCs of V_1 appear as the first and second actions. By repeated application of this reasoning, we see that if there are at least m actions belonging to EPCs of V_1 in X, then there is an execution $X' = s_0 a_1 s_1 ... s_{m-1} a_m s_m ...$ equivalent by independence to X such that a_i is the i^{th} action belonging to an EPC in V_1 to appear in X for $1 \le i \le m$. ■

Lemma 3.4.2. For a Flow model F partitioned into two sets V_1 and V_2, if $X = s_0 a_1 s_1 ... s_{m-1} a_m s_m ...$ is an execution of F such that for all actions a_i $i < m$, a_i belongs to an EPC of V_1, then $X_1 = s_0' a_1 s_1' ... s_{m-1}'$ where s_i' is the projection of s_i onto V_1 is a prefix of an execution of the sub-model F_1 induced by V_1. Also, if all actions a_j $j \geq m$ belongs to an EPC of V_2, then X projected onto the sub-model F_1 induced by V_1 is an execution of F_1 and X projected onto the sub-model F_2 induced by V_2 is an execution of F_2.

Proof:

Let $X_1 = s_0' a_1 s_1' ... s_{m-1}'$ either be the first $m-1$ actions of X where all actions belong to EPCs of V_1 and s_i' is the projection of s_i onto V_1 or if there is no a_i belonging to an EPC of V_1 for $i \geq m$, the projection of X onto the sub-model F_1 induced by V_1. Consider action a_j $1 \leq j \leq m-1$ of X_2 belonging to EPC M. We need to show that a_j is enabled in s_{j-1}' and executing a_j in s_{j-1}' results in s_j'. By definition of state projection, the internal state of M and all its in-channels in s_{j-1}' are identical to their corresponding states in s_{j-1} of X. Since a_j is enabled in s_{j-1} and is predicated only on the internal state of M and M's in-channels by definition of Flow models, it must be enabled in s_{j-1}'. Also since a_j is a function of the internal state of M and M's in-channels by definition of Flow models, and those are identical in s_{j-1}' and s_{j-1}, the internal state of M as a result of executing a_j in s_{j-1}' must be identical to the internal state of M in s_j, which by definition of state projection is true of s_j'. Also, since the state of an in-channel in s_{j-1}' is identical to its state in s_{j-1}, and a_j adds and removes the same events from it whether executed from s_{j-1}' or s_{j-1}, its state as a result of executing a_j in s_{j-1}' must be identical to its state in s_j, which by definition of state projection is true of s_j'. Lastly, the internal state of any other EPC of V_1 does not changed from s_{j-1} to s_j by definition of Flow models, and so the internal state of an EPC different from M does not change from s_{j-1}' to s_j' by definition of state projection, which is correct by definition of Flow models. This shows that a_j is enabled in s_{j-1}' and executing a_j in s_{j-1}' results in s_j'. This also shows that X_1 is the prefix of an execution of F_1. To show that X_1 is also an execution of F_1 in case there are no a_i in X belonging to an EPC of V_1 for $i \geq m$, we also need to show that no actions belonging to EPCs of V_1 are enabled in s_{m-1}'. If there are, then the same action must be enabled in s_{m-1} since the internal state of M and M's in-channels are identical in the two states. If an action is enabled in s_{m-1}, it would never get executed because all subsequent actions belong to EPCs of V_2 by assumption, and remain enabled for all subsequent states because actions from EPCs of

V_2 cannot disable it by corollary 3.2.12. This would imply X is not fair, a contradiction. This shows that X_1 is an execution of F_1.

Similarly, if there are no a_i in X belonging to EPCs of V_1 for $i \geq m$, we can argue that $X_2 = s_{m-1}'a_m s_m' a_{m+1} s_{m+1}'...$, the projection of X onto the sub-model F_2 induced by V_2, is an execution of F_2. Consider action a_j $j \geq m$ of X_2 belonging to EPC M'. We need to show that a_j is enabled in s_{j-1}' and executing a_j in s_{j-1}' results in s_j'. By definition of state projection, the internal state of M' and all its in-channels in s_{j-1}' are identical to their corresponding states in s_{j-1} of X. Since a_j is enabled in s_{j-1} and is predicated only on the internal state of M' and M''s in-channels by definition of Flow models, it must be enabled in s_{j-1}'. Also since a_j is a function of the internal state of M' and M''s in-channels by definition of Flow models, and those are identical in s_{j-1}' and s_{j-1}, the internal state of M' as a result of executing a_j in s_{j-1}' must be identical to the internal state of M' in s_j, which by definition of state projection is true of s_j'. Also, since the state of an in-channel in s_{j-1}' is identical to its state in s_{j-1}, and a_j adds and removes the same events from it whether executed from s_{j-1}' or s_{j-1}, its state as a result of executing a_j in s_{j-1}' must be identical to its state in s_j, which by definition of state projection is true of s_j'. Lastly, the internal state of any other EPC of V_2 does not change from s_{j-1} to s_j by definition of Flow models, and so the internal state of an EPC different from M' does not change from s_{j-1}' to s_j' by definition of state projection, which is correct by definition of Flow models. This shows that a_j is enabled in s_{j-1}' and executing a_j in s_{j-1}' results in s_j'. If X_2 is finite, we need to show that no actions belonging to EPCs of V_2 are enabled in its last state, say s_{m+n}'. If there are, then the same action must be enabled in s_{m+n}, the last state of X, since the internal state of M' and M''s in-channels are identical in the two states. But this would contradict the fact that X is an execution of F. If X_2 is infinite, we need to show no action belonging to an EPC of V_2 is enabled forever without getting executed. If there is such an action, and it is enabled in $s_{m+i}', s_{m+i+1}',...$ and is not one the actions $a_{m+i}, a_{m+i+1},...$, it must be enabled in $s_{m+i}, s_{m+i+1},...$ and is not one of the actions $a_{m+i}, a_{m+i+1},...$, which implies X is not fair. But this is a contradiction. This shows that X_2 is an execution of F_2. ■

Theorem 3.4.3. For a Flow model F with graph $G(V, E)$ that can be unidirectional cut into V_1 and V_2 such that edges are directed from V_1 to V_2 and no in-channel is connected to both an out-channel from an EPC of V_1 and an out-channel from an EPC of V_2, if all executions of the sub-model induced by V_1 are finite and all executions of the sub-model induced by V_2 are finite, then all executions of F are finite.

Proof:

If an execution X of F is infinite, it must contain infinite number of operations belonging to EPCs of V_1 or infinite number of operations belonging to EPCs of V_2.

Suppose it contains infinite number of operations belonging to EPCs of V_1. Suppose m is the maximum length of executions of the sub-model F_1 induced by V_1. By lemma 3.4.1, there is an execution $X' = s_0 a_1 s_1 ... s_{m-1} a_m s_m a_{m+1} s_{m+1} ...$ of F equivalent by independence to X such that a_i belongs to an EPC of V_1 for $1 \le i \le m+1$. By lemma 3.4.2, $s_0' a_1 s_1' ... s_{m-1}' a_m s_m' a_{m+1} s_{m+1}'$ where s_i' is the projection of s_i onto V_1 is a prefix of an execution of F_1. But this is impossible since the longest execution of F_1 contains m actions. So X must contain a finite number of operations belonging to EPCs of V_1.

Suppose X contains an infinite number of operations belonging to EPCs of V_2. Since it can only contain a finite number of operations belonging to EPCs of V_1, let m be the number of operations in X belonging to EPCs of V_1. By lemma 3.4.1, there is an execution $X' = s_0 a_1 s_1 ... s_{m-1} a_m s_m ...$ of F equivalent by independence to X such that all actions a_i $i \le m$ belong to EPCs of V_1, and all actions a_j $j \ge m+1$ belong to EPCs of V_2. Note that X' must be infinite with an infinite number of actions belonging to EPCs of V_2. By lemma 3.4.2, the projection of X' onto V_2, which must be infinite, must be an execution of the sub-model F_2 induced by V_2. But this is a contradiction because F_2 has only finite executions by assumption.

This shows that X can only contain finite number of actions belonging to EPCs of V_1 and finite number of actions belonging to EPCs of V_2. Therefore, X must be finite. ■

Theorem 3.4.3. If the graph of a Flow model F is acyclic, then all executions of F are finite.

Proof:

We prove this using induction on the number of vertices. For the base case, suppose there is only one vertex in the graph of F and with no edge from it to itself. This means there is only one EPC M in F and none of its out-channels are connected to its in-channels. By definition of Flow models, each action execution removes at least one event from M 's in-channels. Also by definition of Flow models, there are a finite number of events in M 's in-channels in any start state. Lastly, no action execution adds any event to M 's in-channels since there are no out-channels connected to them. Consequentially, all executions must contain only a finite number of action executions and are finite.

For the inductive hypothesis, assume all executions of Flow models that have graphs which are acyclic and contain k or less vertices are finite. For the inductive case, we want to show that this is true for any Flow model F the graph $G(V, E)$ of which is acyclic and contains $k+1$ vertices.

From graph theory, we know that a directed acyclic graph has at least one vertex v with out-degree zero. This means $V_1 = V / \{v\}$ and $V_2 = \{v\}$ gives a unidirectional cut of G such that edges are directed from V_1 to V_2. Also since V_2 contains only one vertex

that does not have a self edge, and there are no edges that are directed from v to vertices in V_1 and thus no connections from the out-channels of v to EPCs of V_1, there is no in-channel that is connected to both an out-channel from an EPC of V_1 and an out-channel from an EPC of V_2. By the inductive hypothesis, all the executions of the Flow model represented by the sub-model induced by V_1, the graph of which must be acyclic and contains k vertices, are finite. Also by the inductive hypothesis, all the executions of the sub-model induced by V_2, the graph of which contains a single vertex acyclic graph, are finite. Therefore, by theorem 3.4.3, all executions of F are finite. ■

Using the above, the easiest deterministically concurrent Flow models to construct or identify are those which contain only deterministic EPCs, have no in-channels connected to by more than one out-channel, and have acyclic graph representations. We can get around the acyclicness condition by dividing up a Flow model into two smaller sub-models using a unidirectional cut and showing each sub-model has only finite executions, perhaps using acyclicness. We can get around the deterministic condition for an EPC as long as we show all its actions are enable immutable and independent relative to each other.

3.5 Unidirectional Cuts And Modularized Reasoning

We saw that in a complex Flow model that does not have an acyclic graph representation, it is possible to simplify the task of proving all executions are finite by using a unidirectional cut to divide it into smaller sub-models and proving each sub-model's executions are finite.

In the same way, when we are analyzing a Flow model for which all executions are not equivalent by independence, we can use unidirectional cuts to divide the model into smaller sub-models and reason about them individually to find the full model's executions and end states. All executions may not be equivalent by independence, for example, if a Flow model is not deterministically concurrent. Although it is desirable for a program to result in one end state for each start state, a Flow model may not be deterministically concurrent if more than one end state represents the same result to the user of the program. Executions may also not be equivalent by independence if the Flow model is deterministically concurrent, but not all executions that end in the same state given the same start state are equivalent by independence. This is possible because the set of executions that are equivalent by independence to an execution may be a proper subset of all the executions that have the same start and end states as that execution.

In these cases, when not all executions from the same start state are equivalent by independence, we would like to find the smallest set of executions to which all other executions from the same start state are equivalent by independence. Such a set would also give us all the possible end states of the Flow model from that start state.

Definition 3.5.1. For a Flow model F and a starting state s_0 of F, a set of executions $\{X_1, X_2, ...\}$ is a *canonical set of executions of F from s_0* if and only if all executions in the set have start state s_0, any execution of F with start state s_0 is equivalent by independence to one of the executions in the set , and no two different executions in the set are equivalent by independence to each other.

We describe an algorithm for dividing up a Flow model into sub-models so that its canonical set of executions from a start state can be found by finding the canonical sets of executions of its sub-models and then concatenating executions from the canonical sets of its sub-models. A precise description of the algorithm follows.

Algorithm 3.5.2. For a Flow model F with graph $G(V, E)$ and a starting state s_0 of F:

1. Form a unidirectional cut of G, partitioning the vertices into V_1 and V_2, such that no in-channel belonging to an EPC of V_2 is connected to both an out-channel belonging to an EPC of V_1 and an out-channel belonging to an EPC of V_2. Let F_1 be the sub-model induced by V_1 and F_2 the sub-model induced by V_2. We require that all executions of F_1 are finite.
2. Form a new augmented Flow model from F_1 called F_1^+ by adding, for each in-channel belonging to an EPC of V_2 connected to out-channels belonging to EPCs of V_1, an EPC with no actions, no internal state, and one in-channel connected to the same out-channels belonging to EPCs of V_1 that the in-channel is connected to. Call these EPCs *event recorders*.
3. Find the canonical set of executions of F_1^+ from a starting state s_0*. In s_0*, the state of all in-channels of event recorders are empty and the state of everything else is identical to their state in s_0. Let S represent the canonical set. Let $P_i = s_{i,0} a_1 s_{i,1} ... s_{i,l_i}$ represent the i^{th} element of the set. Note that $s_{i,0} = s_0*$. Note that since all executions of F_1 are finite, P_i must be finite in length, so l_i is the length of P_i.
4. For each P_i, let $R_i = s_0 a_1 s_1 ... s_{l_i}$ be the unique execution fragment of F that is the reverse projection onto F from s_0 of the projection of P_i onto F_1. For each P_i, find R_i.
5. For each fragment $R_i = s_0 a_1 s_1 ... s_{l_i}$, find the canonical set of executions of F_2 from a starting state $s_{l_i}*$ which is the projection of s_{l_i} onto F_2. Let S_i represent the canonical set of executions of F_2 from $s_{l_i}*$. Let $P_{i,j} = s_{i,j,l_i} a_{l_i+1} s_{i,j,l_i+1} ...$ represent the j^{th} element of the set. Note that $s_{i,j,l_i} = s_{l_i}*$.

6. For each $P_{i,j}$, let $R_{i,j} = s_{l_i} a_{l_i+1} s_{l_i+1} \ldots$ be the unique execution fragment of F that is the reverse projection onto F from s_{l_i} of $P_{i,j}$. For each $P_{i,j}$, find $R_{i,j}$.
7. Form the set $S^* = \{R_1 \bullet R_{1,1}, R_1 \bullet R_{1,2}, \ldots, R_2 \bullet R_{2,1}, R_2 \bullet R_{2,2}, \ldots\}$, which is a canonical set of executions of F from s_0.

Note that it can be applied repeatedly to divide sub-models into even smaller sub-models so that the task of finding the canonical set of executions for the entire model becomes divided into simpler and simpler tasks. We now prove that each step of the algorithm is possible and correct in its assertions.

Theorem 3.5.3. Corresponding to (7), $S^* = \{R_1 \bullet R_{1,1}, R_1 \bullet R_{1,2}, \ldots, R_2 \bullet R_{2,1}, R_2 \bullet R_{2,2}, \ldots\}$ is a canonical set of executions of F from s_0.

Proof:

Let $R_i \bullet R_{i,j} = s_0 a_1 s_1 \ldots$ be an element of S^*. First we show that $R_i \bullet R_{i,j}$ is a legal execution of F. Note that R_i and $R_{i,j}$ are execution fragments of F and by construction, s_{l_i} is the last state of R_i and the first state of $R_{i,j}$. This shows that $R_i \bullet R_{i,j}$ is a legal concatenation of two execution fragments of F, which is itself an execution fragment of F by lemma 2.2.1. This implies that for all actions a_k in $R_i \bullet R_{i,j}$, a_k must be enabled in s_{k-1} and the result of executing a_k in s_{k-1} is s_k.

Suppose that $R_{i,j}$ is finite, so that $R_i \bullet R_{i,j} = s_0 a_1 s_1 \ldots s_n$ is finite. We need to show that there are no actions of F enabled in the end state. For an action a belonging to an EPC M of V_1, we know that a is disabled in s_{i,l_i} since P_i is an execution of F_1^+, which implies that a is disabled in s_{l_i} since the internal state of M and the state of M's in-channels are identical in s_{l_i} and s_{i,l_i} by definition 2.2.3 and lemma 2.2.6. Also, since all actions in $R_i \bullet R_{i,j}$ after s_{l_i} belong to EPCs of V_2, none of which can add or remove events from M's in-channels by assumption or change the internal state of M by definition of Flow models, a must remain disabled until s_n by corollary 3.2.12. This shows no actions belonging to EPCs of V_1 are enabled in s_n. For an action a belonging to an EPC M' of V_2, we know that a is disabled in $s_{i,j,n}$ since $P_{i,j}$ is an execution of F_2, which implies that a is disabled in s_n by lemma 2.2.5 since the projection of s_n onto F_2 equals $s_{i,j,n}$. This shows no action of F is enabled in s_n, which shows that $R_i \bullet R_{i,j}$ is an execution of F if it is finite.

Suppose that $R_{i,j}$ is infinite, so that $R_i \bullet R_{i,j} = s_0 a_1 s_1 \ldots$ is infinite, we need to show that this execution is fair. First, no action a belonging to an EPC M of V_1 is enabled in any state after s_{l_i} for the same reason as above. Suppose there is an action a belonging

to an EPC M' of V_2 that is enabled in $s_k, s_{k+1}...$ and is none of the actions $a_k, a_{k+1}...$. Without loss of generality, we assume $k \geq l_i$. Since the projections of s_m onto F_2 equals $s_{i,j,m}$ for $m \geq l_i$, a is enabled in $s_{i,j,k}, s_{i,j,k+1}...$ of $P_{i,j}$ by lemma 2.2.5 and is not one of the actions $a_k, a_{k+1}...$. But then this contradicts the fact that $P_{i,j}$ is an execution of F_2. This shows that $R_i \bullet R_{i,j}$ must be fair. In all cases, $R_i \bullet R_{i,j}$ is an execution of F.

Next we must show that all executions of F are equivalent by independence to some execution $R_i \bullet R_{i,j}$ in S. Let $X' = s_0' a_1' s_1'...$ be an execution of F. Note s_0' is the same as s_0. First we show X' can contain only a finite number of actions belonging to EPCs of V_1. Suppose it doesn't. Let m be the length of the longest execution of F_1. Lemma 3.4.1 says that there is an execution equivalent by independence to X' such that the first $m+1$ actions belong to EPCs of V_1. Lemma 3.4.2 says that this implies there exists a prefix of length $m+1$ of an execution of F_1, which is a contradiction since the longest execution of F_1 is length m. This shows that X' contains a finite number of actions belonging to EPCs of V_1. Let n be the number of actions belonging to EPCs of V_1 in X'.

Lemma 3.4.1 then says that X' is equivalent by independence to an execution $X'' = s_0'' a_1'' s_1''...s_n'' a_{n+1}'' s_{n+1}''...$ such that a_k'' belongs to an EPC of V_1 for $1 \leq k \leq n$ and a_k'' belongs to an EPC of V_2 for $k \geq n+1$. Note s_0'' is the same as s_0', which is the same as s_0. Lemma 3.4.2 then says that the projection $X_1'' = s_{1,0}'' a_1'' s_{1,1}''...s_{1,n}''$ of X'' onto V_1 is an execution of F_1 and the projection $X_2'' = s_{2,n}'' a_{n+1}'' s_{2,n+1}''...$ of X'' onto V_2 is an execution of F_2. Note that $s_{1,k}''$ is the projection of s_k'' onto V_1 and $s_{2,k}''$ is the projection of s_k'' onto V_2.

Since X_1'' is an execution of F_1 which is a sub-model of F_1^+, by lemma 2.2.6, there is a reverse projection $X_1''^+ = s_{1,0}''^+ a_1'' s_{1,1}''^+...s_{1,n}''^+$ of X_1'' onto F_1^+ from $s_{1,0}''^+$, which has the same state for all the EPCs of V_1 and their in-channels and empty states of all in-channels of the event recorders. Also, no actions of F_1^+ are enabled in $s_{1,n}''$ since all actions of F_1^+ are also actions of F_1 and this would mean by lemma 2.2.5 that an action of F_1 is enabled in $s_{1,n}$, contradicting the fact that X_1'' is an execution of F_1. So $X_1''^+$ is an execution of F_1^+. Note that this implies X'' has the properties that its k^{th} action is the same as the k^{th} action of $X_1''^+$ for $1 \leq k \leq n$, and the projection of the k^{th} state of X'' onto F_1 is equal to the projection of the k^{th} state of $X_1''^+$ onto F_1 for $0 \leq k \leq n$. Also, the k^{th} action of X'' is the same as a_k'' of X_2'' for $k \geq n+1$, and the projection of the k^{th} state of X'' onto F_2 is equal to $s_{2,k}''$ for $k \geq n$.

Since $X_1''^+$ is an execution of F_1^+, it is equivalent by independence to an $P_i \in S$. This means there is a finite sequence of executions, $X_1''^+(1), X_1''^+(2)...X_1''^+(p)$, such that $X_1''^+(1) = X_1''^+$, $X_1''^+(p) = P_i$, and each is derived by independence from the

previous. Let a_k'' and a_{k+1}'' be the two actions switched around from X_1''$^+$(1) to X_1''$^+$(2). Let M_k and M_{k+1} be the EPCs of V_1 to which a_k'' and a_{k+1}'' belong respectively. We show that these two actions are also independent in X'' so that there is an execution X''(2) equivalent by independence with X'' that have the two actions switched around. We know a_k'' and a_{k+1}'' are independent in X_1''$^+$(1). That means a_{k+1}'' is enabled in $s_{1,k-1}$''$^+$. That means a_{k+1}'' is enabled in $s_{1,k-1}$'' by lemma 2.2.5. That means a_{k+1}'' is enabled in s_{k-1}'' by lemma 2.2.5. Let $s_{1,k}$''$^+$* and s_k''* be the states resulting from executing a_{k+1}'' in $s_{1,k-1}$''$^+$ and s_{k-1}'' respectively. a_{k+1}'' executed in $s_{1,k-1}$''$^+$ must result in the same state for M_k and all its in-channels as a_{k+1}'' executed in $s_{1,k-1}$'' by lemma 2.2.5, which must be the same as the state for M_k and all its in-channels in the state resulting from executing a_{k+1}'' in s_{k-1}'' by lemma 2.2.5. Since a_k'' is enabled in $s_{1,k}$''$^+$*, it must then be enabled in s_k''* since the state of M_k and all its in-channels have the same state in $s_{1,k}$''$^+$* and s_k''*. Now consider s_{k+1}''*, the result of executing a_{k+1}'' followed by a_k'' in s_{k-1}''. Let $s_{1,k+1}$''$^+$* represent the state resulting from executing a_{k+1}'' followed by a_k'' in $s_{1,k-1}$''$^+$, which we know is equal to $s_{1,k+1}$''$^+$ since a_k'' and a_{k+1}'' are independent in X_1''$^+$(1). Let $s_{1,k+1}$''* represent the state resulting from executing a_{k+1}'' followed by a_k'' in $s_{1,k-1}$'', which is the projection of both $s_{1,k+1}$''$^+$* and s_{k+1}''* onto F_1 by lemma 2.2.5. Also note that $s_{1,k+1}$'' is the projection of both $s_{1,k+1}$''$^+$ and s_{k+1}'' onto F_1 by lemma 2.2.5. We need to show that s_{k+1}''* is the same as s_{k+1}''. Consider the internal state of an EPC of V_1 or one of its in-channels. Its state in $s_{1,k+1}$''$^+$* is the same as its state in $s_{1,k+1}$''*, and its state in $s_{1,k+1}$''* is the same as its state in s_{k+1}''* by lemma 2.2.5. But its state in $s_{1,k+1}$''$^+$* is the same as its state in $s_{1,k+1}$''$^+$ since a_k'' and a_{k+1}'' are independent in X_1''$^+$(1), and its state in $s_{1,k+1}$''$^+$ is the same as its state in $s_{1,k+1}$'', and its state in $s_{1,k+1}$'' is the same as its state in s_{k+1}'' by lemma 2.2.5. So its state is the same in s_{k+1}'' and s_{k+1}''*. Now consider an EPC of V_2 or one of its in-channels not connected to an out-channel belonging to an EPC of V_1. By definition of Flow models, their states are not affected by a_k'' or a_{k+1}'', so their states in s_{k+1}''* is the same as their states in s_{k-1}'' which is the same as their states in s_{k+1}''. Lastly, consider an in-channel C belonging to an EPC of V_2 connected to one or more out-channels belonging to M_k and/or M_{k+1}. There is an event recorder's in-channel connected to the same out-channels of M_k and/or M_{k+1} in F_1^+. Suppose C's state in s_{k-1}'' is $e_1e_2...e_q$, then its state in s_{k+1}'' is $e_1e_2...e_qe_{q+1}...e_{q+r}$ where $q,r \geq 0$. This implies executing a_k'' followed by a_{k+1}'' in s_{k-1}'' adds $e_{q+1}...e_{q+r}$ to C. This implies executing a_k'' followed by a_{k+1}'' in $s_{1,k-1}$''$^+$ adds $e_{q+1}...e_{q+r}$ to the corresponding recorder's in-

channel by lemma 2.1.31 since the control state of a_k'' is the same in s_{k-1}'' and $s_{1,k-1}''^{+}$, and the control state of a_{k+1}'' is the same in s_k'' and $s_{1,k}''^{+}$. This implies if the state of the event recorder's in-channel is $e_1'e_2'...e_t'$ in $s_{1,k-1}''^{+}$, then its state in $s_{1,k+1}''^{+}$ is $e_1'e_2'...e_t'e_{q+1}...e_{q+r}$. But this is its state in $s_{1,k+1}''^{+}*$, which implies executing a_{k+1}'' followed by a_k'' in $s_{1,k-1}''^{+}$ adds $e_{q+1}...e_{q+r}$ to the event recorder's in-channel. Since the control state of a_{k+1}'' is the same in s_{k-1}'' and $s_{1,k-1}''^{+}$, and the control state of a_k'' is the same in $s_k''*$ and $s_{1,k}''^{+}*$, lemma 2.1.31 implies executing a_{k+1}'' followed by a_k'' in $s_{1,k-1}''$ adds $e_{q+1}...e_{q+r}$ to C. This implies the state of C in $s_{k+1}''*$ is $e_1e_2...e_qe_{q+1}...e_{q+r}$, which is identical to its state in s_{k+1}''. This shows that s_{k+1}'' and $s_{k+1}''*$ are identical, which shows that a_k'' and a_{k+1}'' are independent in X''. Note that in a new execution of F derived by independence from X'' by switching a_k'' and a_{k+1}'' around, s_{k+1}'' is not changed by definition of independence, so that any sequence of such derivations will not change s_n'' in the derived execution.

This shows that X'' is equivalent by independence to another execution $X''(2)$ of F such that its k^{th} action is the same as the k^{th} action of $X_1''^{+}(2)$ for $1 \le k \le n$, and the projection of its k^{th} state onto F_1 is equal to the projection of the k^{th} state of $X_1''^{+}(2)$ onto F_1 for $0 \le k \le n$, and its k^{th} action is the same as a_k'' of X_2'' for $k \ge n+1$, and the projection of its k^{th} state onto F_2 is equal to $s_{2,k}''$ for $k \ge n$. By repeated application of this reasoning, we can find $X''(3)$ from $X_1''^{+}(3)$ and so on until we have $X''' = s_0'''a_1'''s_1'''...s_n''a_{n+1}''s_{n+1}''...$, which is equivalent by independence to X'' and thus X', and has the properties that its k^{th} action is the same as the k^{th} action of P_i for $1 \le k \le n$, and the projection of its k^{th} state onto F_1 is equal to the projection of the k^{th} state of P_i onto F_1 for $0 \le k \le n$, and its k^{th} action is the same as a_k'' of X_2'' for $k \ge n+1$, and the projection of its k^{th} state onto F_2 is equal to $s_{2,k}''$ for $k \ge n$.

Now consider X_2''. Since it is an execution of F_2, and its starting state is the projection of s_n'' onto F_2, which from X''' we see is the last state of the reverse projection onto F from s_0 of the projection of P_i, it is equivalent by independence to an $P_{i,j} \in S_i$. This means there is a finite sequence of executions, $X_2''(1), X_2''(2)...X_2''(p)$, such that $X_2''(1) = X_2''$, $X_2''(p) = P_{i,j}$, and each is derived by independence from the previous. Let a_k'' and a_{k+1}'' be the two actions switched around from $X_2''(1)$ to $X_2''(2)$, $k \ge n+1$. Let M_k and M_{k+1} be the EPCs of V_1 to which a_k'' and a_{k+1}'' belong respectively. We show that these two actions are also independent in X''' so that there is an execution $X'''(2)$ equivalent by independence with X''' that have the two actions switched around. We know a_k'' and a_{k+1}'' are independent in $X_2''(1)$. That means a_{k+1}'' is enabled in $s_{2,k-1}''$. By lemma 2.2.5, this implies it is enabled in s_{k-1}''. Let $s_{2,k}''*$ and

s_k''* be the states resulting from executing a_{k+1}'' in $s_{2,k-1}$'' and s_{k-1}'' respectively. The states of all EPCs of V_2 and their in-channels must be the same in $s_{2,k}$''* and s_k''* by lemma 2.2.5. Since a_k'' is enabled in $s_{2,k}$''*, it must then be enabled in s_k''* since the state of the EPC M_k of V_1 to which it belongs and all its in-channels have the same state in $s_{2,k}$''* and s_k''*. Now consider s_{k+1}''*, the result of executing a_{k+1}'' followed by a_k'' in s_{k-1}''. Let $s_{2,k+1}$''* represent the state resulting from executing a_{k+1}'' followed by a_k'' in $s_{2,k-1}$'', which we know is equal to $s_{2,k+1}$'' and is the projection of s_{k+1}''* onto F_2 by lemma 2.2.5. Also note that $s_{2,k+1}$'' is the projection of s_{k+2}'' onto F_2. We need to show that s_{k+1}'' and s_{k+1}''* are identical. Consider the internal state of an EPC of V_2 or one of its in-channels. By lemma 2.2.5, its state in s_{k+1}''* is the same as its state in $s_{2,k+1}$''*, and its state in $s_{2,k+1}$''* is the same as its state in $s_{2,k+1}$'', and its state in $s_{2,k+1}$'' is the same as its state in s_{k+1}'' by lemma 2.2.5. This shows its state is the same in s_{k+1}'' and s_{k+1}''*. Consider an EPC of V_1 or one of its in-channels. By the definition of the unidirectional cut and Flow models, their states are not affected by a_k'' or a_{k+1}'', so their states in s_{k+1}''* are the same as their states in s_{k-1}'', and their states in s_{k-1}'' are the same as their states in s_{k+1}''. This shows that s_{k+1}'' and s_{k+1}''* are identical, which shows that a_k'' and a_{k+1}'' are independent in X'''.

This shows that X''' is equivalent by independence to another execution X'''(2) of F such that its k^{th} action is the same as the k^{th} action of P_i for $1 \le k \le n$, and the projection of its k^{th} state onto F_1 is equal to the projection of the k^{th} state of P_i onto F_1 for $0 \le k \le n$, and its k^{th} action is the same as the $(k-n)^{th}$ action of X_2''(2) for $k \ge n+1$, and the projection of its k^{th} state onto F_2 is equal to the $(k-n)^{th}$ state of X_2''(2) for $k \ge n$. By repeated application of this reasoning, we can find X'''(3) from X_2''(3) and so on until we have X'''', which is equivalent by independence to X''', X'' and X', and has the properties that its k^{th} action is the same as the k^{th} action of P_i for $1 \le k \le n$, and the projection of its k^{th} state onto F_1 is equal to the projection of the k^{th} state of P_i onto F_1 for $0 \le k \le n$, and its k^{th} action is the same as the $(k-n)^{th}$ action of $P_{i,j}$ for $k \ge n+1$, and the projection of its k^{th} state onto F_2 is equal to the $(k-n)^{th}$ state of $P_{i,j}$ for $k \ge n$. But then, this means that X'''' is the same as $R_i \bullet R_{i,j}$ and so X' is equivalent by independence to an element of S*.

Lastly, we need to show that no two different executions in S* are equivalent by independence to each other. Suppose there are two elements of S*, $R_i \bullet R_{i,j}$ and $R_i' \bullet R_{i,j}'$, that are equivalent by independence to each other. Then there is a finite sequence of executions $X_1, X_2, ..., X_m$ each derived by independence from the previous

such that the first equals $R_i \bullet R_{i,j}$ and the last equals $R_i' \bullet R_{i,j}'$. First we show that R_i must be identical to R_i'.

Suppose that there exists at least one derivation by independence in the sequence which switches the order of two operations both belonging to EPCs of V_1. If not, then we already have that R_i is identical to R_i'. So there is an execution X_d in the $X_1, X_2, ..., X_m$ such that the next derivation by independence switches two actions belonging to EPCs of V_1. Let the two actions switched be a_k and a_{k+1} belonging to M_k and M_{k+1}, possibly the same EPC, of V_1. Let s_{k-1} and s_k be the states in which a_k and a_{k+1} execute and s_{k+1} the result of their execution in $R_i \bullet R_{i,j}$. Let s_{k-1}' and s_k' be the states in which a_k and a_{k+1} execute and s_{k+1}' the result of their execution in execution X_d. We want to show that the independence of a_k and a_{k+1} in X_d implies their independence in $R_i \bullet R_{i,j}$.

First, we show that the internal state of all EPCs of V_1 and the state of their in-channels, in particular the control state of any action belonging to an EPC of V_1, is the same in the state in which an action belonging to an EPC of V_1 executes in $R_i \bullet R_{i,j}$ and the corresponding state in which it executes in X_d. We show this by induction on the sequence of executions $X_1, X_2, ..., X_d$, the sequence of executions leading up to X_d each derived by independence from the previous. This is trivially true in X_1 since it is identical to $R_i \bullet R_{i,j}$. Suppose it is true in X_p where X_p occurs in the sequence before X_d. The next derivation by independence must involve an action belonging to an EPC of V_2 since the derivation after X_d is assumed to be the first that involves switching two actions both belonging to EPCs of V_1. First, note that the state in which an action executes for any action before or after the two switched actions in X_{p+1} is not changed from the state in which it executes in X_p by definition of derivation by independence, so the internal state of any EPC of V_1 and the states of its in-channels are the same in the state in which an action belonging to an EPC of V_1 executes in X_{p+1} and the corresponding state in which it executes in X_p if the action is before or after the two switched actions. If in the derivation from X_p to X_{p+1}, both actions belong to EPCs of V_2, then this already shows that internal state of any EPC of V_1 and the states of its in-channels are the same in the state in which an action belonging to an EPC of V_1 executes in X_{p+1} and the corresponding state in which it executes in X_p, which is the same as their states in the corresponding state in which the action executes in $R_i \bullet R_{i,j}$ by induction. If in the derivation from X_p to X_{p+1}, the first action belongs to an EPC of V_1 and the second belongs to an EPC of V_2 in X_p, executing the second action first in X_{p+1} has no effect on any of the EPCs of V_1 or their in-channels since it belongs to an EPC of V_2 and by definition of unidirectional cuts. So the internal state of all EPCs of V_1 and the state of their in-channels are the same in the state in which the switched action belonging

to an EPC of V_1 executes in X_p and the corresponding state in which it executes X_{p+1}. If in the derivation from X_p to X_{p+1}, the first action belongs to an EPC of V_2 and the second belongs to an EPC of V_1 in X_p, since the action that belongs to an EPC of V_2 has no effect on any of the EPCs of V_1 or their in-channels by definition of unidirectional cuts, the internal states of all EPCs of V_1 and the state of their in-channels in the state in which the switched action belonging to an EPC of V_1 executes in X_p is the same as their states in the state in which the switched action belonging to an EPC of V_2 executes in X_p. But this is the state in which the switched action belonging to an EPC of V_1 executes in X_{p+1}. So in all cases, the internal states of all EPCs of V_1 and the state of their in-channels in the state in which the an action belonging to an EPC of V_1 executes in X_p is the same as their states in the corresponding state when the action executes in X_p. This shows that the property holds in X_d by induction.

The above implies the internal states of M_k, M_{k+1} and the states of their in-channels must be identical in s_{k-1} and s_{k-1}'. Since a_k and a_{k+1} are both enabled in s_{k-1}', they must both be enabled in s_{k-1} since the states they are predicated on are identical in s_{k-1} and s_{k-1}'. Since the control state of a_{k+1} is the same in s_{k-1} and s_{k-1}', and all EPCs of V_1 and the state of their in-channels are the same in s_{k-1} and s_{k-1}', by lemma 2.1.31 all EPCs of V_1 and the state of their in-channels must also be the same in s_k* and s_k*', the states resulting from executing a_{k+1} in s_{k-1} and s_{k-1}' respectively. Specifically, M_k and its in-channels must have the same states in s_k* and s_k*'. Since a_k is enabled in s_k*', it must be enabled in s_k* since the states it is predicated on are identical in s_k* and s_k*'. Let $s_{k+1}*$ and $s_{k+1}*'$ be the states resulting from executing a_{k+1} followed by a_k in s_{k-1} and s_{k-1}' respectively. Note that $s_{k+1}*'$ is identical to s_{k+1}' since a_k and a_{k+1} are assumed to be independent in X_d. Also note that the internal states of all EPCs of V_1 and the states of their in-channels are the same in s_{k+1}' and s_{k+1} by lemma 2.1.31 since they are the same in s_{k-1} and s_{k-1}', so that a_k has the same control state when executed in s_{k-1} and s_{k-1}' which results in the same states for them in s_k and s_k', and since a_{k+1} then has the same control state when executed in s_k and s_k'. We need to show that $s_{k+1}*$ is identical to s_{k+1}. This is true for the internal state of any EPC different from M_k, M_{k+1}, and any of its in-channels not connected to M_k and M_{k+1} since neither a_k nor a_{k+1} changes their state by definition of Flow models. This is also true for any EPC of V_1 and their in-channels since we have already shown that their states are identical in s_k* and s_k*', which means the control state of a_k is the same in s_k* and s_k*', so their states must also be the same in $s_{k+1}*$ and $s_{k+1}*'$ by lemma 2.1.31. But their states in $s_{k+1}*'$ are identical to their states in s_{k+1}', which is identical to their states in s_{k+1}. This leaves only any in-

channel C belonging to an EPC of V_2 and connected to in-channels of M_k and/or M_{k+1} to be considered. Suppose the state of C is $e_1e_2...e_r$ in s_{k-1}' and $e_1e_2...e_re_{r+1}...e_{r+t}$ in $s_{k+1}*'$ where $r,t \geq 0$. This means a_{k+1} executed in s_{k-1}' followed by a_k executed in s_k*' adds $e_{r+1}...e_{r+t}$ to C. This also means a_k executed in s_{k-1}' followed by a_{k+1} executed in s_k' adds $e_{r+1}...e_{r+t}$ to C since the two actions are independent in X_d so the state of C must also be $e_1e_2...e_re_{r+1}...e_{r+t}$ in s_{k+1}'. Suppose the state of C is $e_1'e_2'...e_u'$ in s_{k-1}. a_{k+1} executed in s_{k-1} followed by a_k executed in s_k* must also add $e_{r+1}...e_{r+t}$ to C by lemma 2.1.31 since the control state of a_{k+1} is the same in s_{k-1} and s_{k-1}', and the control state of a_k is the same in s_k* and s_k*'. So the state of C in $s_{k+1}*$ must be $e_1'e_2'...e_u'e_{r+1}...e_{r+t}$. a_k executed in s_{k-1} followed by a_{k+1} executed in s_k must also add $e_{r+1}...e_{r+t}$ to C by lemma 2.1.31 since the control state of a_k is the same in s_{k-1} and s_{k-1}', and the control state of a_{k+1} is the same in s_k and s_k'. So the state of C in s_{k+1} must be $e_1'e_2'...e_u'e_{r+1}...e_{r+t}$, the same as its state in $s_{k+1}*$. This shows $s_{k+1}*$ must be identical to s_{k+1}, which shows that a_k and a_{k+1} are independent in $R_i \bullet R_{i,j}$. So there is an execution that is derived by independence from $R_i \bullet R_{i,j}$ that has a_k and a_{k+1} switched.

From the above, we also know that this execution derived from $R_i \bullet R_{i,j}$ has the property that the internal state of all EPCs of V_1 and the state of their in-channels in any state in which an action belonging to an EPC of V_1 executes is the same as their states in the corresponding state when the action executes in X_{d+1}. This allows us to apply the same reasoning as above to show that the next time in the sequence $X_1, X_2,..., X_d$ that two actions both belonging to EPCs of V_1 are switched around, the same two actions can in the derived execution. By repeated application of this reasoning, we see that there is a sequence of executions, each having all actions belonging to EPCs of V_1 precede all actions belonging to EPCs of V_2, each derived by independence from the previous by switching actions only belonging to EPCs of V_1, with the first equaling $R_i \bullet R_{i,j}$ and the last equaling $R_i' \bullet R_{i,j}'$. Let $X_1', X_2',..., X_g'$ be the sequence.

Consider P_i and P_i', the executions of F_1^+ and members of S that when projected onto F_1 and then reverse projected onto F from s_0, equals R_i and R_i' respectively. We show that the above implies P_i must be equivalent by independence to P_i'. Suppose two actions a_k and a_{k+1} both belonging to EPCs M_k and M_{k+1}, possibly the same EPC, of V_1 are independent in $R_i \bullet R_{i,j}$. By the definition 2.2.3 and lemma 2.2.6, they are consecutive actions in P_i. We first show that they must be independent in P_i. The proof is very similar to the proof that two actions belonging to EPCs of V_1 are independent in $R_i \bullet R_{i,j}$ if they are independent in X_d. Let s_{k-1} and s_k be the states in which a_k and a_{k+1} executes respectively and s_{k+1} the result of their execution in $R_i \bullet R_{i,j}$. Let s_{k-1}' and s_k' be the states in which a_k and a_{k+1} executes respectively and s_{k+1}' the result of their

execution in execution P_i. Note that the internal states of all EPCs of V_1 and the states of their in-channels in s_{k-1} and s_{k-1}' are identical by definition 2.2.3 and lemma 2.2.6. Specifically, M_{k+1} and its in-channels must have the same states in s_{k-1} and s_{k-1}'. Since a_{k+1} is enabled in s_{k-1}, it must be enabled in s_{k-1}' since the states it is predicated on are identical in s_{k-1} and s_{k-1}'. Also, since the control state of a_{k+1} is the same in s_{k-1} and s_{k-1}', and all EPCs of V_1 and the state of their in-channels are the same in s_{k-1} and s_{k-1}', by lemma 2.1.31 all EPCs of V_1 and the state of their in-channels must also be the same in s_k * and s_k*', the states resulting from executing a_{k+1} in s_{k-1} and s_{k-1}' respectively. Specifically, M_k and its in-channels must have the same states in s_k * and s_k*'. Since a_k is enabled in s_k *, it must be enabled in s_k*' since the states it is predicated on are identical in s_k * and s_k*'. Let s_{k+1} * and s_{k+1}*' be the states resulting from executing a_{k+1} followed by a_k in s_{k-1} and s_{k-1}' respectively. Note that s_{k+1} * is identical to s_{k+1} since a_k and a_{k+1} are independent in $R_i \bullet R_{i,j}$. Also note that the internal states of all EPCs of V_1 and the states of their in-channels are the same in s_{k+1} and s_{k+1}' since they are the same in s_{k-1} and s_{k-1}', which means a_k has the same control state in s_{k-1} and s_{k-1}', which shows by lemma 2.1.31 that they have the same state in s_k and s_k', which means a_{k+1} has the same control state in s_k and s_k', which shows by lemma 2.1.31 that they have the same states in s_{k+1} and s_{k+1}'. We need to show that s_{k+1}*' is identical to s_{k+1}'. This is true for any EPC of V_1 and their in-channels since we have already shown that their states are identical in s_k * and s_k*', and since the control state of a_k is the same in s_k * and s_k*', their states must also be the same in s_{k+1} * and s_{k+1}*'. But their states in s_{k+1} * are identical to their states in s_{k+1}, which are identical to their states in s_{k+1}'. This shows their states are identical in s_{k+1}*' and s_{k+1}'. This leaves only an in-channel C' belonging to an event recorder. By definition of event recorders, there is an in-channel C in F belonging to an EPC of V_2 connected to the exact same out-channels as C'. Suppose the state of C is $e_1e_2...e_r$ in s_{k-1} and $e_1e_2...e_re_{r+1}...e_{r+t}$ in s_{k+1} * where $r,t \geq 0$. This means a_{k+1} executed in s_{k-1} followed by a_k executed in s_k * adds $e_{r+1}...e_{r+t}$ to C. This also means a_k executed in s_{k-1} followed by a_{k+1} executed in s_k adds $e_{r+1}...e_{r+t}$ to C since the two actions are independent in $R_i \bullet R_{i,j}$ and C's state in s_{k+1} is $e_1e_2...e_re_{r+1}...e_{r+t}$. Suppose the state of C' is $e_1'e_2'...e_u'$ in s_{k-1}'. a_{k+1} executed in s_{k-1}' followed by a_k executed in s_k*' must also add $e_{r+1}...e_{r+t}$ to C' by lemma 2.1.31 since the control state of a_{k+1} is the same in s_{k-1} and s_{k-1}', and the control state of a_k is the same in s_k * and s_k*'. So the state of C' in s_{k+1}*' must be $e_1'e_2'...e_u'e_{r+1}...e_{r+t}$. a_k executed in s_{k-1}' followed by a_{k+1} executed in s_k' must also add $e_{r+1}...e_{r+t}$ to C by lemma 2.1.31 since the control state of a_k is the same in s_{k-1} and s_{k-1}', and the control state of a_{k+1} is the same in s_k and s_k'. So the state of C' in s_{k+1}' must be

$e_1'e_2'...e_u'e_{r+1}...e_{r+t}$, the same as its state in s_{k+1}*'. This shows s_{k+1}*' must be identical to s_{k+1}', which shows that a_k and a_{k+1} are independent in P_i. So there is an execution that is derived by independence from P_i that has a_k and a_{k+1} switched. Furthermore, the above shows that this derived execution, when projected onto F_1 and then reverse projected onto F from s_0, equals the prefix of X_2' containing all the actions belonging to EPCs of V_1. This means we can apply the same reasoning above to show that one can derive by independence another execution from this execution that has the same property in relation to X_3'. By repeated application of this reasoning, we see that P_i must be equivalent by independence to P_i', which implies they must be identical since they belong to the same canonical set. This implies that R_i and R_i' must be identical.

Note that since R_i and R_i' are identical, P_i and P_i' are identical, and so $P_{i,j}$ and $P_{i,j}'$, the executions of F_2 from which $R_{i,j}$ and $R_{i,j}'$ are reverse projected, must be in the same canonical set of F_2 from the projection onto F_2 of the end state of R_i.

Again, let $X_1, X_2, ..., X_m$ be the sequence of executions each derived by independence from the previous such that the first equals $R_i \bullet R_{i,j}$ and the last equals $R_i' \bullet R_{i,j}'$. We show that $R_{i,j}$ must be identical to $R_{i,j}'$. Note that there exists at least one derivation by independence in the sequence which switches the order of two operations both belonging to EPCs of V_2. If not, then we already have that $R_{i,j}$ is identical to $R_{i,j}'$. So there is an execution X_d in the sequence of executions each derived by independence from the previous such that the next derivation by independence switches two actions belonging to EPCs of V_2. We want to show that if two consecutive actions both belonging to EPCs of V_2 are independent in X_d, then they are independent in $R_i \bullet R_{i,j}$.

We first show that for an execution of an action a belonging to an EPC M of V_2, in the state in which it executes in X_d and in the state in which it executes in $R_i \bullet R_{i,j}$, its control state must be the same. We show this by induction on $X_1, X_2, ..., X_d$, the sequence of executions leading up to X_d each derived by independence from the previous. This is trivially true in X_1 since it is identical to $R_i \bullet R_{i,j}$. Suppose it is true in X_p where X_p occurs in the sequence before X_d. The next derivation by independence must involve an action belonging to an EPC of V_1 since the derivation after X_d is assumed to be the first that involves switching two actions both belonging to EPCs of V_2. First, note that the state in which an action executes for any action before or after the two switched actions in X_{p+1} is not changed from the state in which they execute in X_p by definition of derivation by independence. If in the derivation from X_p to X_{p+1}, both actions belong to EPCs of V_1, then this already shows that the control state of any action a belonging to an EPC M of V_2 when it executes in X_{p+1} and X_p are the same. If in the derivation from X_p to X_{p+1}, the first action a_k belongs to an EPC M of V_2 and the

second action a_{k+1} belongs to an EPC M' of V_1 in X_p, and s_{k-1} is the state in which a_k executes in X_p and a_{k+1} executes in X_{p+1}, since a_{k+1} can only add events to the in-channels of M and a_k is already enabled in s_{k-1}, executing a_{k+1} in s_{k-1} cannot change the control state of a_k by definition of Flow models. So the control state of a_k is the same in the states in which it executes X_{p+1} and X_p. If in the derivation from X_p to X_{p+1}, the first action a_k belongs to an EPC M of V_1 and the second action a_{k+1} belongs to an EPC M' of V_2 in X_p, and s_{k-1} is the state in which a_k executes in X_p and a_{k+1} executes in X_{p+1}, and s_k is the state resulting from executing a_k in s_{k-1}, since the actions are independent then a_{k+1} must be enabled in s_{k-1}, which implies its control state in s_{k-1} and s_k must be the same since a_k can only add events to M''s in-channels, which implies its control state in the state in which it executes is the same in X_{p+1} and X_p. In all cases, the control state of any action a belonging to an EPC M of V_2 in the state in which it executes in X_{p+1} and X_p are the same. This shows that the control state of any action a belonging to an EPC M of V_2 in the state in which it executes in X_d and $R_i \bullet R_{i,j}$ are the same by induction.

Let the two actions that are to be switched in X_d be a_k and a_{k+1} belonging to M_k and M_{k+1}, possibly the same EPC, of V_2. Let s_{k-1} and s_k be the states in which a_k and a_{k+1} execute respectively and s_{k+1} the result of their execution in $R_i \bullet R_{i,j}$. Let s_{k-1}' and s_k' be the states in which a_k and a_{k+1} execute respectively and s_{k+1}' the result of their execution in X_d. We want to show that the independence of a_k and a_{k+1} in X_d implies their independence in $R_i \bullet R_{i,j}$. First we show that the independence of a_k and a_{k+1} in X_d implies a_{k+1} is enabled in s_{k-1}. Suppose a_{k+1} is not enabled in s_{k-1}. First consider the case that one of the in-channels C of M_{k+1} on which a_{k+1} is predicated is empty in s_{k-1}. We show that this implies it must be empty in s_{k-1}', which is a contradiction since a_{k+1} is enabled in s_{k-1}'. Since a_{k+1} is enabled in s_k but C is empty in s_{k-1}, it must be the case that a_k executed in s_{k-1} adds an event to C. This means that no actions belonging to EPCs of V_1 can add events to C since by assumption, no in-channel belonging to an EPC of V_2 can be connected both to an out-channel belonging to an EPC of V_1 and an out-channel belonging to an EPC of V_2. Also, the actions that belong to EPCs of V_2 that precede a_k in X_d are the exactly same as those that precede a_k in $R_i \bullet R_{i,j}$ since no two such actions has been switched by assumption. Furthermore each such action adds and removes the same number of events from C by lemma 2.1.31 since their control states are the same in the states in which they execute in $R_i \bullet R_{i,j}$ and X_d. Lastly, none of the actions belonging to EPCs of V_1 that precede a_k adds any events to C in either $R_i \bullet R_{i,j}$ or X_d since their out-channels are not connected to C. This shows C must have the

same number of events in s_{k-1} and s_{k-1}', so it must be empty in s_{k-1}'. This is a contradiction to the fact that a_{k+1} is enabled in s_{k-1}'. So, if a_{k+1} is disabled in s_{k-1}, all the in-channels on which it is predicated must be non-empty. Note that this implies a_k belongs to the same EPC since otherwise a_k cannot change M_{k+1}'s internal state or the events that a_{k+1} is predicated on, and a_{k+1} must still be disabled in s_k which is a contradiction. But then, we know the internal state of M_{k+1} is the same in s_{k-1} and s_{k-1}' since it is the same as the internal state of M_k, which is the same in s_{k-1} and s_{k-1}' since it is part of the control state of a_k. Also, the event at the head of an in-channel predicated upon by a_{k+1} but not a_k must be the same in s_{k-1} and s_{k-1}' since it is the same in s_k and s_k', and the event was present at the head of the in-channel in s_{k-1} and s_{k-1}' since a_k executed in s_{k-1} and s_{k-1}' could not have removed it. Lastly, the event at the head of an in-channel predicated upon both by a_{k+1} and a_k must be identical in s_{k-1} and s_{k-1}' since it is part of the control state of a_k in s_{k-1} and s_{k-1}'. This shows that the control state of a_{k+1} must be the same in s_{k-1} and s_{k-1}', and so a_{k+1} cannot be disabled in s_{k-1} since it is enabled in s_{k-1}'. This shows that if a_{k+1} is enabled in s_{k-1}', it must be enabled in s_{k-1}. Also note that this shows a_{k+1} has the same control state in s_{k-1} and s_{k-1}'. Let s_k* and s_k*' be the states resulting from executing a_{k+1} in s_{k-1} and s_{k-1}' respectively.

We next want to show that the control state of a_k in s_k* and s_k*' are the same. Suppose a_{k+1} and a_k belong to different EPCs, then by the definition of Flow models a_{k+1} can only add events to the in-channels of M_k. Since a_k is already enabled in s_{k-1} and s_{k-1}' with the same control state, executing a_{k+1} in s_{k-1} and s_{k-1}' cannot change the control state of a_k, and so the control state of a_k in s_k* and s_k*' are the same. Suppose a_{k+1} and a_k belong to the same EPC. Note that we showed above a_{k+1} has the same control state in s_{k-1} and s_{k-1}' in this case. The internal state of the EPC in s_k* and s_k*' then must be the same since a_{k+1} has the same control state in s_{k-1} and s_{k-1}'. The event at the head of an in-channel predicated upon by a_k but not a_{k+1} are the same in s_{k-1} and s_{k-1}' since they are part of the control state of a_k, so they must be the same in s_k* and s_k*' since a_{k+1} cannot remove events from these in-channels. Now consider an event at the head of an in-channel predicated upon by both a_k and a_{k+1}. By definition of Flow models, a_{k+1} executed in s_{k-1} and s_{k-1}' removes the first event from it, as does a_k executed in s_{k-1} and s_{k-1}'. a_k executed in s_{k-1} and s_{k-1}' either adds to the in-channel or does not. Suppose it doesn't. Then there must be a second event in the channel in both states, since in both states a_{k+1} predicated on the same channel is enabled after a_k executes. Furthermore, the second event must be the same in both states since a_{k+1} sees the same event in s_k and s_k', and the event is part of its control state. This implies a_k sees the same event on the in-channel in s_k* and s_k*'. Suppose executing a_k in s_{k-1}

and s_{k-1}' does add one or more events to the in-channel. Since an EPC of V_2 adds events to the in-channel, no action belonging to EPCs of V_1 can add to the same in-channel by assumption. Note that we showed above that such an in-channel must have the same state in s_{k-1} and s_{k-1}'. Since the in-channel has the same state in s_{k-1} and s_{k-1}', and the control state of a_{k+1} in s_{k-1} and s_{k-1}' is the same, by lemma 2.1.31 the state of the in-channel and thus the event at the its head must be the same in s_k * and s_k*'. This shows that the control state of a_k in s_k * and s_k*' are the same which means a_k is enabled in s_k * since it is enabled in s_k*'.

Let s_{k+1} * and s_{k+1}*' be the states resulting from executing a_k in s_k * and s_k*' respectively. To show that a_k and a_{k+1} are independent in $R_i \bullet R_{i,j}$ if they are independent in X_d, we must show that s_{k+1} * is identical to s_{k+1}. Note that since a_k and a_{k+1} are independent in X_d, s_{k+1}*' is identical to s_{k+1}' by the definition of independence. Consider the internal state of any EPC of V_1 or the state of their in-channels or the internal state of any EPC of V_2 that is not M_k or M_{k+1}, or the state of their in-channels which are not connected to out-channels of M_k or M_{k+1}. By definition of unidirectional cuts and Flow models, they are not affected by a_k or a_{k+1}. So their states in s_{k+1} * are the same as their states in s_{k-1}, which are the same as their states in s_{k+1}. The same is true for the internal state of any EPC that is not M_k or M_{k+1} and any in-channels that do not belong to M_k or M_{k+1} and are not connected to out-channels of M_k or M_{k+1}. The internal state of M_k and M_{k+1} must be the same in s_{k+1} * and s_{k+1}*' since we showed they are the same in s_{k-1} and s_{k-1}', so executing a_{k+1} in s_{k-1} and s_{k-1}' results in the same internal states for them in s_k * and s_k*' by lemma 2.1.31 since we showed the control states of a_{k+1} are the same in s_{k-1} and s_{k-1}', and executing a_k in s_k * and s_k*' results in the same internal states for them in s_{k+1} * and s_{k+1}*' by lemma 2.1.31 since we showed the control states of a_k are the same in s_k * and s_k*'. But then their internal states in s_{k+1}*' are identical to their internal states in s_{k+1}', which are identical to their internal states in s_{k+1}. Note the latter identity is true by lemma 2.1.31 also, since their internal states are identical in s_{k-1} and s_{k-1}', and the control state of a_k is the same in s_{k-1} and s_{k-1}', and the control state of a_{k+1} is the same in s_k and s_k'. Now consider an in-channel C that belongs to M_k or M_{k+1} and is connected to out-channels of M_k and/or M_{k+1}. If the in-channel is not predicated upon by a_k or a_{k+1}, let $e_1 e_2 ... e_r$ be its state in s_{k-1}', then its state in s_{k+1}' and s_{k+1}*' is $e_1 e_2 ... e_r e_{r+1} ... e_{r+t}$, where $r, t \geq 0$. This implies executing a_k followed by a_{k+1} and a_{k+1} followed by a_k in s_{k-1}' both adds $e_{r+1} ... e_{r+t}$ to C. This implies executing a_k followed by a_{k+1} in s_{k-1} adds $e_{r+1} ... e_{r+t}$ to C since we showed a_k has the same control state when executed in s_{k-1}' and s_{k-1}, and a_{k+1} has the same control state when executed in s_k' and s_k. Similarly, executing a_{k+1} followed by a_k in s_{k-1} adds

$e_{r+1}...e_{r+t}$ to C since we showed a_{k+1} has the same control state when executed in s_{k-1}' and s_{k-1}, and a_k has the same control state when executed in s_k*' and s_k*. This implies the state of C is the same in $s_{k+1}*$ and s_{k+1}. Suppose C is predicated upon by a_k but not a_{k+1} or a_{k+1} but not a_k. Let $e_1e_2...e_r$ be its state in s_{k-1}', with $r \geq 1$ since both a_k and a_{k+1} must be enabled in s_{k-1}'. Then its state in s_{k+1}' and $s_{k+1}*'$ is $e_2...e_re_{r+1}...e_{r+t}$, where $t \geq 0$. This implies executing a_k followed by a_{k+1} and a_{k+1} followed by a_k in s_{k-1}' both removes the first event from C, which is guaranteed to exist in s_{k-1}', and adds $e_{r+1}...e_{r+t}$ to C. This implies executing a_k followed by a_{k+1} in s_{k-1} removes the first event from C which is guaranteed to exist in s_{k-1} since a_{k+1} is enabled in s_{k-1} and adds $e_{r+1}...e_{r+t}$ to C by lemma 2.1.31 since we showed a_k has the same control state when executed in s_{k-1}' and s_{k-1}, and a_{k+1} has the same control state when executed in s_k' and s_k. Similarly, executing a_{k+1} followed by a_k in s_{k-1} removes the first event from C which is guaranteed to exist in s_{k-1} and adds $e_{r+1}...e_{r+t}$ to C by lemma 2.1.31 since we showed a_{k+1} has the same control state when executed in s_{k-1}' and s_{k-1}, and a_k has the same control state when executed in s_k*' and s_k*. This implies the state of C is the same in $s_{k+1}*$ and s_{k+1}. Suppose C is predicated upon by both a_k and a_{k+1}. If it is not connected to the out-channels of M_k or M_{k+1}, then C must have two events in s_{k-1} since neither a_k nor a_{k+1} adds events to it and each removes an event from it when executed in either order. The state of C in either s_{k+1} and $s_{k+1}*$ is then its state in s_{k-1} with two events from its head removed, so it has the same state in s_{k+1} and $s_{k+1}*$. If it is connected to out-channels of M_k or M_{k+1}, then it cannot be connected to out-channels belonging to EPCs of V_1. We showed that such an in-channel has the same state in s_{k-1} as in s_{k-1}'. Since C has the same state in s_{k-1} as in s_{k-1}', then it must have the same state in $s_{k+1}*'$ and $s_{k+1}*$ by lemma 2.1.31 since a_{k+1} has the same control state when executed in s_{k-1}' and s_{k-1}, and a_k has the same control state when executed in s_k*' and s_k*, and C must have the same state in s_{k+1}' and s_{k+1} by lemma 2.1.31 since a_k has the same control state when executed in s_{k-1}' and s_{k-1}, and a_{k+1} has the same control state when executed in s_k' and s_k. This shows C has the same state in s_{k+1} and $s_{k+1}*$. This shows that s_{k+1} and $s_{k+1}*$ are the same, which implies that a_k and a_{k+1} are independent in $R_i \bullet R_{i,j}$. So there is an execution that is derived by independence from $R_i \bullet R_{i,j}$ that has a_k and a_{k+1} switched.

From the above, we also know that this execution derived from $R_i \bullet R_{i,j}$ has the property that the control state of an action belonging to an EPC of V_2 in the state in which it executes is the same as its control state in the state in which it executes in X_{d+1}. Furthermore, the order in which actions belonging to EPCs of V_2 are executed is the same as in X_{d+1}. This allows us to reuse the reasoning above to show that any two actions

belonging to EPCs of V_2 that are independent in X_{d+1} are also independent in the derived execution. By repeated application of this reasoning, we see that there is a sequence of executions, each having all actions belonging to EPCs of V_1 precede all actions belonging to EPCs of V_2, each derived by independence from the previous by switching actions only belonging to EPCs of V_2, with the first equaling $R_i \bullet R_{i,j}$ and the last equaling $R_i' \bullet R_{i,j}'$. Let $X_1', X_2', ..., X_g'$ be the sequence.

Consider $P_{i,j}$ and $P_{i,j}'$, the executions of F_2 and elements of S_i that are the projections onto F_2 of $R_i \bullet R_{i,j}$ and $R_i' \bullet R_{i,j}'$. We show that the above implies $P_{i,j}$ must be equivalent by independence to $P_{i,j}'$. Suppose two actions a_k and a_{k+1} belonging to EPCs M_k and M_{k+1}, possibly the same EPC, of V_2 are independent in $R_i \bullet R_{i,j}$. By the definition 2.2.4, they are consecutive actions in $P_{i,j}$. We show that they must be independent in $P_{i,j}$. Let s_{k-1} and s_k be the states in which a_k and a_{k+1} executes respectively and s_{k+1} the result of their execution in $R_i \bullet R_{i,j}$. Let s_{k-1}' and s_k' be the states in which a_k and a_{k+1} executes respectively and s_{k+1}' the result of their execution in execution $P_{i,j}$. Since $P_{i,j}$ is a projection of $R_i \bullet R_{i,j}$ onto F_2, and a_k and a_{k+1} belong to EPCs of F_2, the internal state of M_{k+1} and the states of its in-channels must be the same in s_{k-1} and s_{k-1}'. Since a_{k+1} is enabled in s_{k-1}, it must be enabled in s_{k-1}'. Since the internal state of all EPCs of V_2 and the state of their in-channels are the same in s_{k-1} and s_{k-1}' by definition 2.2.4, all EPCs of V_2 and the state of their in-channels must also be the same in s_k* and s_k*', the states resulting from executing a_{k+1} in s_{k-1} and s_{k-1}' respectively by lemma 2.1.31, since the control state of a_{k+1} is the same in s_{k-1} and s_{k-1}'. Since a_k is enabled in s_k*, it must be enabled in s_k*'. Also, its control state must be the same in s_k* and s_k*'. Let $s_{k+1}*$ and $s_{k+1}*'$ be the states resulting from executing a_k in s_k* and s_k*' respectively. Since the internal state of all EPCs of V_2 and the state of their in-channels are the same in s_k* and s_k*', and the control state of a_k is the same in s_k* and s_k*', the internal state of all EPCs of V_2 and the state of their in-channels are the same in $s_{k+1}*$ and $s_{k+1}*'$. But since the internal state of all EPCs of V_2 and the state of their in-channels are the same in $s_{k+1}*$ and s_{k+1} because they are independent in $R_i \bullet R_{i,j}$, and the same in s_{k+1} and s_{k+1}' because s_{k+1} is a projection of s_{k+1}' onto F_2, it must be the case that the internal state of all EPCs of V_2 and the state of their in-channels are the same $s_{k+1}*'$ and s_{k+1}'. This shows a_k and a_{k+1} are independent in $P_{i,j}$. This also shows that an execution of F_2 can be derived by independence from $P_{i,j}$ such that it equals the projection of X_2' onto F_2. Since this execution is the projection of X_2' onto F_2, the above reasoning can be repeated to show that another execution of F_2 can be derived from it that is the projection of X_3' onto F_3. By repeated application of this reasoning,

we see that there is a sequence of executions of F_2, each derived by independence from the previous, such that the first equals $P_{i,j}$ and the last equals the projection of $R_i' \bullet R_{i,j}'$ onto F_2, which is $P_{i,j}'$. This shows $P_{i,j}$ and $P_{i,j}'$ are equivalent by independence, which means they must be the same execution since we showed they are in the same canonical set. This implies that $R_{i,j}$ and $R_{i,j}'$ must be identical.

Since we showed that both R_i and R_i' are identical and $R_{i,j}$ and $R_{i,j}'$ are identical, this shows that if $R_i \bullet R_{i,j}$ and $R_i' \bullet R_{i,j}'$ are equivalent by independence, they must be identical. This shows that S is a canonical set of executions of F from s_0. ■

3.6 Hoare Triples And Feasible Traces

Our ultimate goal as previously stated is to be easily able to show that all the end states of a Flow model, given a set of possible start states as characterized by some boolean pre-condition, satisfy some other boolean post-condition. This is a general and common way to characterize the correctness of any program.

In the previous sections however, we have focused on easy ways to find the end states of a Flow model given a specific start state. In this section, we show how this can be used to show that all possible end states of a Flow model, given a set of start states as described by some boolean pre-condition, satisfy some boolean post-condition.

The best case is if, given a set of start states of a Flow model, there is an execution from each starting state whose trace is the same. If the model also has the property that all executions from a specific starting state are equivalent by independence, then we simply need to show that the common trace satisfies the post condition and this implies that all executions from the set of start states satisfy the post condition. We describe a way to do this using Hoare triples. [10]

Definition 3.6.1. For a Flow model F, a finite trace $a_1 a_2 ... a_n$ of F, a boolean function on the state of F f_{pre}, and a boolean function on the state of F f_{post}, $[f_{pre}]a_1 a_2 ... a_n[f_{post}]$ if and only for any state s of F, if f_{pre} evaluates to true on s then there is an execution fragment of F with trace $a_1 a_2 ... a_n$ such that s is its starting state and f_{post} evaluates to true on s', its ending state.

Lemma 3.6.2. For a Flow model F, a finite trace $a_1 a_2 ... a_n$ of F where $[f_{pre}]a_1 a_2 ... a_n[f_{post}']$, and a finite trace $a_{n+1} a_{n+2} ... a_{n+m}$ of F where $[f_{pre}']a_{n+1} a_{n+2} ... a_{n+m}[f_{post}]$, if for any state s of F, f_{pre}' evaluates to true on s if f_{post}' evaluates to true on s, then $[f_{pre}]a_1 a_2 ... a_n a_{n+1} ... a_{n+m}[f_{post}]$.

Proof:

For any state s_0 of F satisfying f_{pre}, let $s_0a_1s_1a_2s_2...s_{n-1}a_ns_n$ be the execution fragment such that s_n satisfies f_{post}'. Since s_n satisfies f_{post}', it satisfies f_{pre}', so there is an execution fragment $s_na_{n+1}s_{n+1}a_{n+2}s_{n+2}...s_{n+m-1}a_{n+m}s_{n+m}$ such that s_{n+m} satisfies f_{post}. Then the execution fragment $s_0a_1s_1a_2s_2...s_{n-1}a_ns_n \bullet s_na_{n+1}s_{n+1}a_{n+2}s_{n+2}...s_{n+m-1}a_{n+m}s_{n+m}$ has trace $a_1a_2...a_na_{n+1}...a_{n+m}$ and the property that s_0 is its starting state and its ending state satisfies f_{post}. ■

The above provides a means to show that from any state that satisfies a pre-condition, there is an execution with a specific trace, and that all executions with that trace from any starting state satisfying the pre-condition also satisfies the post-condition with its end state. Combined with equivalence by independence, this can be sufficient to show that all executions from start states that satisfy the pre-condition have end-states that satisfy the post-condition.

Theorem 3.6.3. For a Flow model F, a boolean function on the state of F f_{pre}, and a boolean function on the state of F f_{post}, if there exists a trace $a_1a_2...a_n$ of F such that $[f_{pre}]a_1a_2...a_n[f_{post}]$, and all guards of all actions of F evaluates to false on any state s of F on which f_{post} evaluates to true, and all executions of F from a specific start state are equivalent by independence, then f_{post} evaluates to true on the ending state of any execution of F that has a starting state on which f_{pre} evaluates to true.

Proof:

For any state s_0 of F satisfying f_{pre}, let $s_0a_1s_1a_2s_2...s_{n-1}a_ns_n$ be the execution fragment with trace $a_1a_2...a_n$ such that s_n satisfies f_{post}. This is an execution since no actions are enabled in s_n. All executions from start state s_0 must be equivalent by independence to this execution, and thus must have end state s_n by lemma 3.2.6. Thus the end state of all executions from start state s_0 has end states that satisfy f_{post}. ■

The above is the best case in a type of strategy for proving the correctness of Flow models as characterized by pre-conditions and post-conditions.

Given a particular starting state s_0, if we want to show that all executions with that starting state has an ending state that satisfies some post-condition, all we need to do is to show that the ending states of all executions in a canonical set from s_0 satisfies the post-condition. Using the tools described above, we construct Flow models that minimizes the size of the canonical set, ideally so that there is just one execution in the set.

Given a pre-condition on starting states, we do not have to consider any starting state in particular. Using Hoare triples, we can show that from any starting state that satisfies

the pre-condition, there exists an execution with a common trace, and regardless of the starting state, the ending states of executions with that trace from starting states satisfying the pre-condition satisfy the post-condition. Then we can use equivalence by independence to generalize this to all executions from those starting states to show that all executions from start states that satisfy the pre-condition have an end states that satisfy the post-condition.

Essentially we have presented a way to prove that all executions of a Flow model from a set of start states have end states that satisfy some post-condition by showing only a few representative executions among them have end-states that satisfy the post-condition.

4. Conclusion and Future Work

We have described a formal model of computation called Flow whose goal is to allow users to describe concurrent systems easily and prove useful properties about the systems easily.

We formally define our model as the asynchronous product of extended state automaton. Using the formal definition, we prove properties about Flow models which allow the user to easily reason about the models they build. We show that if the model satisfies certain easy to check properties, then all executions of the model are guaranteed to have the same end state that have the same start state. This allows the user to find the end state given a start state by simply looking at the end state of one possible execution and not all possible executions. If the model has multiple end states from a start state, we show a technique where, by dividing the larger model into smaller sub-models, we can find the smallest set of representative executions such that any execution from the same start state is guaranteed to have the same end state as one of the representative executions. We show that if the model satisfies certain easy to check properties, then all executions of the model are guaranteed to be finite. If the model does not satisfy these properties, we show a technique where, by dividing the larger model into smaller sub-models, all executions of the model can be shown to be finite by showing that the executions of sub-models are finite. Lastly, we show how to use Hoare triples [10] to prove that all executions with the same trace from a group of possible starting states as characterized by some pre-condition all have end states that satisfy some post-condition. When combined with a model where all executions from the same start state has the same end state, this technique can be used to show all executions with start states that satisfy some pre-condition have end states that satisfy some post-condition by looking at just one feasible trace.

Flow models allow the user to specify all possible concurrent executions from some start state. The next step is, given an underlying architecture with multiple processors which may either share memory or must communicate over a network, and usage patterns which may change over time, to find the optimal execution of the Flow model on a real machine or network of machines. In other words, a Flow model specifies many possible executions that lead to the correct end result. Which execution maximizes efficiency, however, is a function of both the underlying architecture and usage patterns. The next step is to elucidate this function, which would allow Flow models to be compiled into real programs that take optimal advantage of the underlying architecture and adapt to usage patterns, so that programmers in the real world can begin using not only as a modeling tool, but as an actual programming language for concurrent systems.

Bibliography

[1] Afek, Y., Attiya, H., Fekete, A., Fischer, M., Lynch, N., Mansour, Y., Wang, D., and Zuck, L. 1994. *Reliable communication over unreliable channels.* J. ACM 41, 6 (Nov. 1994), 1267-1297.

[2] Chandy, K.M., Aydemir, B.E., Karpilovsky, E.M. and Zimmerman, D.M. *Event-Driven Architectures for Distributed Crisis Management*. 2003. Presented at the 15th IASTED International Conference on Parallel and Distributed Computing and Systems, November 2003.

[3] Chandy, K.M., Aydemir, B.E., Karpilovsky, E.M. and Zimmerman, D.M. *Event Webs for Crisis Management*. 2003. Presented at the 2nd IASTED International Conference on Communications, Internet and Information Technology, November 2003.

[4] Emerson, E.A., Jha, S. and Peled, D. 1997. *Combining Partial Order and Symmetry Reductions.* TACAS 1997: 19-34.

[5] Fekete, A., Lynch, N., Mansour, Y., and Spinelli, J. 1993. *The impossibility of implementing reliable communication in the face of crashes*. J. ACM 40, 5 (Nov. 1993), 1087-1107.

[6] Gerth, R., Kuiper, R., Peled, D., and Penczek, W. 1999. *A partial order approach to branching time logic model checking.* Inf. Comput. 150, 2 (May. 1999), 132-152.

[7] Godefroid, P. 1996 *Partial-Order Methods for the Verification of Concurrent Systems: an Approach to the State-Explosion Problem*. Springer-Verlag New York, Inc.

[8] Gray, J. and Reuter, A. 1992 *Transaction Processing: Concepts and Techniques*. 1st. Morgan Kaufmann Publishers Inc.

[9] Hadzilacos, V. and Toueg, S. 1993. *Fault-tolerant broadcasts and related problems.* In *Distributed Systems (2nd Ed.)*, S. Mullender, Ed. Acm Press Frontier Series. ACM Press/Addison-Wesley Publishing Co., New York, NY, 97-145.

[10] Hoare, C. A. R. *An axiomatic basis for computer programming*. 1969. Communications of the ACM, 12(10):576–585, October 1969.

[11] Holzmann, G.J. and Peled, D. 1994. *An Improvement in Formal Verification*. Proc. FORTE 1994 Conference, Bern, Switzerland.

[12] Holzmann, G.J., Peled, D. and Yannakakis, M. 1996. *On nested depth-first search*, in *The Spin Verification System*, pp. 23-32, American Mathematical Society, 1996. (Proc. of the 2nd Spin Workshop.)

[13] Holzmann, G.J. and Puri A. 1998. *A minimized automaton representation of reachable states.* Software Tools Technol. Transfer 3 (1998) 1.

[14] Kupferman, O., Vardi, M. Y., and Wolper, P. 2000. *An automata-theoretic approach to branching-time model checking.* J. ACM 47, 2 (Mar. 2000), 312-360.

[15] Lea, D. and Lea, D. 2000 *Concurrent Programming in Java: Design Principles and Patterns*. Addison-Wesley Longman Publishing Co., Inc.

[16] Lynch, N. A. 1996 *Distributed Algorithms*. Morgan Kaufmann Publishers Inc.

[17] Manna, Z. and Pnueli, A. 1992 *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag New York, Inc.

[18] Peierls, T., Goetz, B., Bloch, J., Bowbeer, J., Lea, D., and Holmes, D. 2006. *Java Concurrency in Practice*. Addison-Wesley Professional.

[19] Peled, D. *All from One, One for All: on Model Checking Using Representatives.* 1993. In Proceedings of the 5th international Conference on Computer Aided Verification (June 28 - July 01, 1993). C. Courcoubetis, Ed. Lecture Notes In Computer Science, vol. 697. Springer-Verlag, London, 409-423.

[20] Schmidt, Douglas, M. Stal, H. Rohnert and F. Buschmann, *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects*. Wiley, 2000.

[21] Weikum, G. and Vossen, G. *Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery*. 2001. Morgan Kaufmann Publishers Inc.