ABSTRACT Behavioral models of interest in software engineering all have equivalent computational power because they are all Turing-equivalent. They do not, however, all have equivalent modeling power. Modeling power is a matter of expressive efficiency, which is difficult to measure because of its subjectivity. One non-subjective measure of modeling power compares models by investigating their representational succinctness. This paper investigates the modeling power of a newly proposed behavioral model, called state nets. The paper presents state nets, shows how the succinctness concept of modeling power can be useful in comparing new models with existing ones, and, as an example, proves that state nets have more modeling power than Petri nets.

Keywords: behavioral models, modeling power, succinctness measures, object-oriented behavioral models, state nets, Petri nets.

I. INTRODUCTION
An important goal of analysis is to capture and communicate, not only the static aspects of a system under study, but also its behavior. Furthermore, an increasing interest in parallel computations and concurrent programming has created a need for even more sophisticated analysis models that more effectively capture and communicate system behavior.

State nets constitute the behavioral model developed for a newly proposed Object-oriented Systems Analysis (OSA) model [EKW 92]. State nets give an analyst a way of effectively specifying both sequential and parallel systems in an object-oriented context. Both intra-object concurrency and inter-object concurrency are embedded in the state-net model.

A natural question to ask, however, is whether state nets are any better, and in what ways, than other behavioral models already in use. Although one model could be better than another in terms of its computational power, this is not an issue because all behavioral models commonly used in software development have the same computational power. The question of interest in comparing computationally equivalent models is expressive power. Expressive power, however, is elusive because it is largely subjective. One objective measure of expressive power has been suggested by Peterson, who calls expressive power, "modeling power," and defines it in terms of representational succinctness [Pet 81]. A model M1 is considered to have more modeling power than a model M2 if M1 uniformly represents the same information as M2 in a more succinct form. Although succinctness must be balanced by readability and understandability (e.g., Gödel numbering is not an ideal approach in practice), Peterson's measure of expressive power does provide insight in comparing behavioral models.

Several behavioral models are commonly used in software engineering, including decision tables [Mor 82], finite-state machines, PAISley [Zav 82], Petri nets [Pet 81], statecharts [Har 87], and others [Dav 90]. From among these, we choose Petri nets as a representative comparison model. We do so because Petri nets are widely used, formally defined, and basically address the same problems and issues as state nets. Like state nets, Petri nets are used to model concurrent systems in software engineering, (see for example its sample use to solve the Dining-Philosophers problem [Pet 81]).

Our purpose, in this paper, is threefold: 1) introduce state nets as a new and promising object-oriented behavioral model, 2) show how Peterson's notion of modeling power can be used to make comparisons between behavioral models, and 3) prove that basic state nets have more modeling power than extended Petri nets.

In Section II, we present informal definitions for Petri nets, state nets, and the notion of modeling power. We have chosen what we call a minimal-set approach for state nets, that is, we have restricted them to their simplest form. For Petri nets, we have chosen extended Petri nets because of their widespread use in this form. Section III formalizes the definitions of extended Petri nets and basic state nets. It also presents the idea of class inclusion used to formally define modeling power and the proof that basic state nets have more modeling power than extended Petri nets. Section IV concludes the paper.

II. INFORMAL DEFINITIONS
1. Petri Nets

Figure 1a is an example of an extended Petri net [Pet 81]. The circles labeled p1 through p4 are called places. The vertical line segments labeled t1 through t4 are called transitions. The lines labeled a1 through a12 are called arcs. Arcs are directed. Arcs ending with a small circle are called inhibitor arcs. All other arcs are called normal arcs.

The places of a Petri net may contain 0 or more tokens, represented as dots inside of places. Once tokens are assigned to places, the resulting Petri net is said to be marked. Informally, a marking of a Petri net is the number of tokens held by each one of its places. A marking can subsequently be formally defined as a function from the set of places to the set of non-negative integers and can be represented as an ordered n-tuple of non-negative integers, one for each place. The first integer in the n-tuple is the number of tokens in p1, the second...
one is the number of tokens in \( p_2 \), etc. The marking of the Petri net of Figure 1a is \( \langle 2,0,3,2 \rangle \).

Given a place and a transition, there may be at most one arc from the place to the transition and at most one arc from the transition to the place. Each place has at least one incoming or one outgoing arc. Each transition has at least one incoming and one outgoing arc. Inhibitor arcs can only go from places to transitions. No pair of places or pair of transitions can be joined by an arc. Normal arcs are weighted. Weights are positive integers, represented as labels on arcs.

Given a place \( p \), a transition \( t \), and an arc \( a \) from \( p \) to \( t \), if \( a \) is a normal arc then \( p \) is said to be an input place of \( t \). If \( a \) is an inhibitor arc then \( p \) is said to be an inhibitor place of \( t \). If, on the other hand, \( a' \) is a normal arc from \( t \) to some place \( p' \), then \( p' \) is called an output place of \( t \).

For a place \( p \), a transition \( t \), and a weight \( i \), the pair \( (p,t) \) represents an arc from place \( p \) to transition \( t \) with weight \( i \). The pair \( (t,p) \) represents an arc from transition \( t \) to place \( p \) with weight \( i \). By convention, \( i = 0 \) for inhibitor arcs. For example, in Figure 1a, \( a_1 = ((t_1,p_1), 1) \), \( a_9 = ((p_3,t_4), 3) \), and \( a_6 = ((p_2,t_1), 0) \).

A transition is enabled when each inhibitor place of the transition contains no tokens and every input place of the transition contains at least as many tokens as the weight of the arc that joins it to the transition. In Figure 1a, transitions \( t_1 \) and \( t_4 \) are enabled. Only enabled transitions may fire. When a transition fires, the marking of the Petri net is instantaneously altered: the number of tokens in each input place is reduced by the weight of the normal arc joining it to the transition, and the number of tokens in each output place is increased by the weight of the normal arc joining it to the transition. For example, Figure 1b is the resulting Petri net after \( t_4 \) of Figure 1a has fired.

Figure 1b - Petri Net of Figure 1a After \( t_4 \) Fired

The execution of a Petri net consists of successive firings of enabled transitions. The firing of enabled transitions is instantaneous (i.e., takes zero time), but non-simultaneous. If more than one transition is enabled, one of them is non-deterministically selected and fired. If at any point in time no transitions are enabled, the Petri net is said to be deadlocked and its execution halts.

2. State Nets
The definition of state nets presented in this section is minimal. It consists of the minimal subset extracted from the complete definition of state nets [EKW 92], that is necessary and sufficient to accomplish our goal of showing that state nets have more modeling power than Petri nets.

State nets are templates for the behavior of object instances of an object class. Figure 2 is an example of a state net for an object class called \( X \).

Figure 2 - State Net for the Object Class \( X \)

The rounded boxes are called states. Each state has a name, placed inside of the rounded box. In Figure 2, there are 5 states, labeled \( S_1 \) through \( S_5 \). A state may either be \( \text{on} \) or \( \text{off} \). When it first comes into existence, a state is off.

The rectangular boxes are called transitions. Transitions may be labeled by a number in square brackets, above the upper left corner of the box. In Figure 2, the transitions are numbered from \([1]\) to \([6]\). The top part of a transition box contains its trigger. A trigger gives the conditions that, when met, may cause the transition to fire. In Figure 2, the triggers are identified by the symbols \( T_1 \) through \( T_6 \). The bottom part of a transition box contains its action, if any. Actions are operations that take place when the transition fires. The actions of Figure 2 are identified by the symbols \( A_1 \), \( A_3 \), \( A_4 \), and \( A_5 \). Transitions \([2]\) and \([6]\) do not have actions. A transition may be \( \text{inactive, ready, starting, executing, or finishing} \). A transition is in exactly one of these states at any time. When it first comes into existence, a transition is in the inactive state.

Arrows (directed arcs) join one or more states to a single transition. Arrows are of three kinds: single-head/single-tail, single-head/multiple-tails, multiple-heads/single-tail. Arrows pointing to transitions are called in-arrows, while arrows pointing away from transitions are called out-arrows. A transition may have 0 or more in-arrows or out-arrows. An in-arrow will always have a single head and an out-arrow will always have a single tail.

A multi-tailed in-arrow pointing to some transition, \( t \), represents a prior-state conjunction of \( t \). A multi-headed out-arrow pointing away from some transition \( t \) represents a subsequent-state conjunction of \( t \). In Figure 2, the set \( \{ S_3, S_4 \} \) is a prior-state conjunction of transition \([1]\), and the set \( \{ S_1 \} \) is a subsequent-state conjunction of transition \([5]\). A prior state conjunction is said to be true when all of its states are on; otherwise, it is false.

Since each object in an object class has its own "executable" copy of the state net for an object class, state nets exhibit inter-object concurrency. Inter-object concurrency is exhibited when several objects are in various states and transitions at the same time. Intra-object concurrency is permitted by allowing an object instance to be in more than one state or transition of a single copy of a state at any point in time. Transitions both within a single copy of a state net and in different copies may fire simultaneously.

In order to explain the behavior (i.e., execution) of state nets, we first define the following:

- \( \text{T\_STATE}(t) \): the state of transition \( t \) \text{ inactive, ready, starting, executing, or finishing} \\
- \( \text{S\_STATE}(s) \): the state of state \( s \) \text{ on or off} \\
- \( \text{ENABLED}(t) \): true if and only if, either \( t \) has no prior state conjunctions, or at least one of \( t \)'s prior state conjunctions is true. \\
- \( \text{TRIGGER}(t) \): the Boolean value of the trigger of transition \( t \) \\
- \( \text{ACTION}(t) \): the action associated with \( t \) if any; else, NULL. \\
- \( \text{PSC}(t) \): the set of prior-state conjunctions of transition \( t \).
S_PSC(p): the set of states in prior-state conjunction p
SSC(t): the set of subsequent-state conjunctions of transition t.
S_SSC(q): the set of states in subsequent-state conjunction q
SELECTED(t): the selected prior-state conjunction of transition t, which may be NULL
READY_CONFLICT(t): true if and only if transition t is in the ready state and there exists another transition t' that is in the ready state and at least one true prior state conjunction of t' contains a state that is also in a true prior state conjunction of t
STARTING_CONFLICT(t): true if and only if t is in the ready state and there exists another transition t' that is in the starting state and SELECTED(t') contains a state that is also in a true prior-state conjunction of t
SELECTABLE(t): true for transition t if and only if READY_CONFLICT(t) and not STARTING_CONFLICT(t)

The execution of a copy C of state net SN can now be defined by the following algorithm, which independently executes for every transition t in C. (Note, if S is a set, then |S| is the cardinality of S.)

WHEN T_STATE(t) = inactive and ENABLED(t) and TRIGGER(t) THEN T_STATE(t) := ready
WHEN T_STATE(t) = ready and not ENABLED(t) or not TRIGGER(t) THEN T_STATE(t) := inactive
WHEN T_STATE(t) = ready and not READY_CONFLICT(t) and not STARTING_CONFLICT(t) THEN
  IF |PSC(t)| > 0 THEN SELECTED(t) := NULL
  ELSE
    non-deterministically select a true p from PSC(t)
    SELECTED(t) := p
    T_STATE(t) := starting
  WHEN T_STATE(t) = starting THEN
    IF SELECTED(t) not NULL THEN
      for all s in SELECTED(t), S_STATE(s) := off
      T_STATE(t) := executing
  WHEN T_STATE(t) = executing THEN
    IF ACTION(t) not NULL THEN
      perform ACTION(t)
      T_STATE(t) := finishing
  WHEN T_STATE(t) = finishing THEN
    IF |SSC(t)| > 0 THEN T_STATE(t) := inactive
    ELSE
      non-deterministically select q from SSC(t)
      for all s in S_SSC(q), S_STATE(s) := on
      T_STATE(t) := inactive
  IF there is more than one transition t, so that SELECTABLE(t) THEN
    non-deterministically select one of those transitions, t'
    non-deterministically select a true p from PSC(t')
    SELECTED(t') := p
    T_STATE(t') := starting

A transition is said to be firing when it is in the starting, executing, or finishing state. Several transitions may fire once and several states may be on at the same time. In Figure 2, when transition [4] is about to fire, the instance of X is in both state S3 and state S4.

If at any point in time, T_STATE(t) = inactive for all transitions, t, of all state nets in the system, and there does not exist a transition t' such that ENABLED(t') and TRIGGER(t') are true, then the state-net system is said to be deadlocked.

The memory of a state net is stored in an instance of an Object-Relationship Model (ORM), which is another subpart of an OSA model. ORMs are semantic data models and are described in [Embley Kurtz Woodfield 91]. An ORM is to a state net what a tape is to a Turing machine. Just as a Turing machine needs to know which character its tape head is pointing to, a state net needs to know which objects currently exist and which relationships among the objects currently hold in its associated ORM. The triggers of the transitions of the state net are then interpreted as queries on the object classes and relationship sets of the associated ORM, while the actions are updates to object classes and relationship sets. The presence (respectively, absence) of certain objects and relationships in the ORM causes certain triggers to be true and others to be false, thus resulting in certain transitions firing while others remain inactive.

Figure 3 shows a simple state net for the object class Savings Account, along with its associated ORM. The ORM consists of three object classes, Savings Account#, Savings Account, and Balance, and two relationship sets Savings Account# identifies Savings Account and Savings Account has Balance. The cardinality constraints for the relationship sets constrain a savings account to have exactly one savings account number and exactly one balance, constrain a savings account number to identify exactly one savings account, and constrain a balance to be associated with at least one savings account.

It is really the combination of a state net and its associated ORM that makes up the actual computational model. The state net in Figure 3, for example, makes use of the information in the ORM. In transition [1] both the trigger and the action reference the balance, and in transition [2] the action references the balance.

Figure 3 - State Net and Associated ORM for the Class Savings Account

3. Modeling Power

Traditionally, computational systems are compared on grounds of computational power. In this context, two systems are equivalent if they can solve exactly the same class of computational problems. Peterson's modeling power, defined in [Pet 81], introduces the issue of modeling efficiency. For two models to be equivalent, they must not only solve the same class of problems, but also do it with approximately the same efficiency.

A class of models, X, has at least as much modeling power as a class of models, Y, if there exists an algorithm that
transforms any instance \( y \) of \( Y \) into an equivalent instance \( x \) of \( X \), such that:

1. The number of structural components of \( x \) is at most a constant multiple of the number of structural components of \( y \). The constant is independent of \( x \) and \( y \).
2. Each state change in \( y \) can be simulated by at most a constant number of state changes in \( x \). The constant is independent of \( x \) and \( y \).
3. \( x \) deadlocks if and only if \( y \) deadlocks.

III. FORMALISM AND PROOF
The method of proof used in this section is taken from the work of Campbell [Cam 89]. We note at the outset that comparing extended Petri nets and basic state nets with respect to their relative modeling power is valid since both models have been shown to be Turing-equivalent, and hence solve the same class of computational problems [Pet 81, G-C 91].

1. Petri Nets, State Nets, and Modeling Power Revisited

a. Petri Nets
Formally, we define an extended Petri net, \( PN \), as a triple \( PN=(P, T_p, A_p) \), where \( P, T_p \), and \( A_p \) are disjoint, finite, non-empty sets: \( P \) is a set of places, \( T_p \) is a set of transitions, and \( A_p \) is a set of arcs. The size of \( PN \) is \( |P|+|T_p|+|A_p| \). The state of \( PN \) is its marking, and is a function from \( P \) to the set of non-negative integers. If \( Mk \) is a marking, we call \( T_e(Mk) \) the subset of transitions of \( T_p \) that are enabled. For example, \( Mk=<2,4,0,1> \) is the state of the Petri net of Figure 1b, and \( T_e(Mk)=\{ t_2, t_4 \} \). A deadlock state of \( PN \) is one in which no transitions are enabled; it satisfies \( T_e(Mk)=\{ \} \).

b. State Nets
We first address the issue of the ORM associated with a state net. An ORM is a repository of information. It consists of a set of object classes and the relationship sets between them. We define the content of an ORM to be the set of objects in each object class, and the set of relationships that hold in each relationship set of the ORM. For example, the content of the ORM of Figure 3 could be:

\[
\{(\text{Savings Account}, \{\text{S1, S2, S3}\})
\]

\[
(\text{Savings Account#}, \{(111-11, 222-22, 333-33)\}),
\]

\[
(\text{Balance}, \{\$300, \$1000\}),
\]

\[
(\text{Savings Account# identifies Savings Account}, \{(S1, 111-11), (S2, 222-22), (S3, 333-33)\}),
\]

\[
(\text{Savings Account has Balance}, \{(S1, \$300), (S2, \$1000), (S3, \$300)\})
\]

Formally, we define a state net, \( SN \), as a triple \( SN=(S, T_S, A_S) \), where \( S, T_S \), and \( A_S \) are disjoint, finite, (usually) non-empty sets: \( S \) is a set of states, \( T_S \) is a set of transitions, and \( A_S \) is a set of arrows. The size of \( SN \) must take into account the size of its associated ORM, therefore, the size of \( SN \) is \( |S|+|T_S|+|A_S|+|OC|+|RS| \), where \( OC \) is the set of object classes and \( RS \) the set of relationship sets in the ORM. A state of \( SN \) is a pair of functions \( < F_S, F_T > \), where \( F_S: S \rightarrow \{ \text{on, off} \} \), and \( F_T: T_S \rightarrow \{ \text{inactive, ready, starting, executing, finishing} \} \). If the states of \( S \) and the transitions of \( T_S \) are ordered (arbitrarily), then \( F_S \) and \( F_T \) may be expressed as ordered \( n \)-tuples. An example of a state for the state net of Figure 2 is given by \( < F_S, F_T >=< \text{on, off, off, off} >, < \text{inactive, inactive, starting, inactive, inactive, inactive} > \) (assuming that states and transitions are ordered according to their labeling numbers). A deadlocked state-net system is one in which all the transitions of all the state nets are inactive; every copy of state nets must satisfy \( F_T = < \text{inactive,...,inactive} > \).

c. Chain of States
A chain of states is a sequence of states from some model where each state in the sequence, except the first one, is the result of a change of state from the immediately preceding state in the chain. The length of a chain of states is the number of states in the chain.

For example, the Petri net of Figure 1b may go through the following chain of states of length 3: \(<2,0,3,2>, <2,4,0,1> \) (\( t_4 \) fired), \(<2,1,1,3> \) (\( t_3 \) fired). There are other possible chains of states for that Petri net. They depend on which transition fires at each step of the execution.

d. Class Inclusion
A class of models, \( X \), includes a class of models, \( Z \), if and only if there exist two constants, \( C \) and \( D \), such that for any model \( z \) of \( Z \), there exist a model \( x \) of \( X \) and a one-to-one function \( f \), from the states of \( z \) to the states of \( x \) such that the following four conditions are satisfied:

1. If \( q \) is a deadlock state of \( z \), then \( f(q) \) is a deadlock state of \( x \).
2. If there is a change of state from \( q \) to \( q' \) in \( z \), then there is a chain of states \( f(q), ..., f(q') \) in \( x \), where the number of states in the chain is less than or equal to \( C \).
3. Let \( q \) and \( q' \) be states of \( z \). If there exists a chain, \( f(q), ..., f(q') \), of more than one state in \( x \), where none of the intermediate states are in the image of \( f \), then there is a change of state from \( q \) to \( q' \) in \( z \).
4. If \( x \) has size \( N \), then \( z \) has size at most \( DN \).

e. Modeling Power
A class of models, \( X \), is defined to have more modeling power than a class of models, \( Z \), if and only if \( X \) includes \( Z \), but \( Z \) does not include \( X \). If \( X \) includes \( Z \) and \( Z \) includes \( X \), then \( X \) and \( Z \) are said to be equivalent.

2. State Nets Include Petri Nets
Let \( C=5 \) and \( D=3 \). Starting with a Petri net, \( PN \), we construct the corresponding state net, \( SN \), and the function \( f \), and show that the definition of class inclusion is satisfied.

a. Construction of the Corresponding State Net
Let \( TOK(p) \) be the number of tokens in place \( p \), and \( NTOK(p) \) be the number of tokens in \( p \) after a transition has fired. Let \( PN=(P, T_p, A_p) \). Construct \( SN=(S, T_S, A_S) \) as follows:

1. Start with \( S, T_S \), and \( A_S \) empty.
2. For every transition \( t \) in \( T_p \), add a transition \( t' \) to \( T_S \).
The trigger of \( t \) is a conjunction of conditions:

For each \(<p, r>, i>\) in \( A_p \), where \( i_o \), add the condition \( \text{TOK}(p) > i \).

For each \(<p,r>, 0>\) in \( A_p \), add the condition \( \text{TOK}(p)=0 \).

The action associated with \( r \) comprises the operations:

For every \(<s, p>, i>\) in \( A_p \), where \( i_o \),

- if there exists \(<s, p>, j>\) in \( A_p \), then
  add the operation \( \text{NTOK}(p)=\text{TOK}(p)+j \).

(Note, if \( j=i=0 \) then this operation is discarded.)

else

add the operation \( \text{NTOK}(p)=\text{TOK}(p)-i \).

For every \(<s, p>, i>\) in \( A_p \), where \( p \neq p \), add the operation \( \text{NTOK}(p')=\text{TOK}(p')+j \).

3. Add the single state, possibly deadlocked, to \( S_t \).

4. For every transition \( t \) in \( T_g \), add to \( A_g \) an in-arrow from the possibly deadlocked state to \( t' \), and an out-arrow from \( t' \) back to the possibly deadlocked state.

5. Add to \( T_g \) the transition [\( t_0 \)] with no associated action and with trigger "@start." Add to \( A_g \) an arrow from \([t_0]\) to the single state of \( S_t \). \([t_0]\) is the initial transition of \( S_t \).

Figure 4 depicts the state net corresponding to the Petri net of Figure 1a, along with its associated ORM. For convenience, the transitions of the state net (except for \([t_0]\)) have been labeled with the "primed" labels of their corresponding transitions in the Petri net (e.g., \([t_1]\) becomes \([t_1']\)). The double-headed arrows are a short-hand for the two single-headed arrows of step 4 of the algorithm.

The associated ORM consists of two object classes, Place and Non-Negative Integer Value, and a relationship set, Non-Negative Integer Value is the number of tokens in Place, which stores information about the number of tokens held by each place of \( PN \). This ORM is the same for every state net.

Figure 4 - State Net obtained from Petri Net of Figure 1a and Associated ORM

derived by the algorithm. Its size is 3, and thus, the size of \( SN \) is \( |S|+|T_g|+|A_g|+3 \).

We will denote by Image(\( t \)) the transition \( t' \) of \( SN \) derived from the transition \( t \) of \( PN \) in step 2 of the algorithm. Since the state-net system we shall consider consists of a single copy of the state-net system \( SN \) constructed by the algorithm, in all that follows \( SN \) shall refer to the state-net system.

b. The Function \( f \)

Let \( q \) be a state of \( PN \). The corresponding state \( f(q)=[F_S,F_t] \) of \( SN \) is given by:

(a) \( F_s \) (possibly deadlocked)=on

(b) For every \( t' \) in \( T_s \), if \( t=\text{Image}(t) \) for \( t \) in \( T_e \), then \( F_t(t')=\text{ready} \) else \( F_t(t')=\text{inactive} \)

(c) For every component of \( MK \), create the equivalent relationship in the relationship set Non-Negative Integer Value is the number of tokens in Place. Note, this step guarantees that \( f \) is one-to-one.

Recall that in \( PN \), only one transition fires at a time, and if several transitions are enabled, one of them is non-deterministically selected to fire. Similarly, in the derived \( SN \), the unique state, possibly deadlocked, is the prior state conjunction of all transitions, thus all transitions in the ready state are selectable, and one of them is non-deterministically selected to fire. Because of this non-deterministic equivalence between \( PN \) and \( SN \), we may assume that, if \( t_0 \) is the transition selected in \( PN \), and \( t'_s \) is the transition selected in \( SN \), then \( t'_s=\text{Image}(t_0) \).

c. Condition (1) Is Satisfied

Let \( q=MK \) be a deadlock state of \( PN \). Then \( T_e(MK)=\{\} \). Consequently, from the way \( f \) is defined, \( f(q)=[F_S,F_t] \) is such that \( F_t=\text{inactive,...,inactive} \). Moreover, due to step (c) of \( f \), every transition \( t' \) of \( T_s \) is such that TRIGGER(\( t' \)) is false. It follows that \( f(q) \) is a deadlock state of \( SN \).

d. Condition (2) Is Satisfied

Let \( q=MK \) and \( q'=MK' \) be states of \( PN \), and assume that there is a change of state from \( q \) to \( q' \) in \( PN \). Then there is a transition \( t \) in \( T_e(MK' \) \) such that when \( t \) fires \( PN \) goes from \( q \) to \( q' \). To \( q \) of \( PN \) corresponds \( f(q)=[F_S,F_t] \) of \( SN \) such that \( F_t(\text{Image}(t))=\text{ready} \). Let \( t'=\text{Image}(t) \). From the assumption made about \( f \), \( t' \) will fire. As \( t' \) fires it goes successively through the starting, executing, and finishing states. This sequence of 3 states consists of states of \( SN \) that are not in the image of \( f \). The firing of \( t' \) causes the content of the ORM associated with \( SN \) to be modified according to the action associated with \( t' \). From the construction of \( SN \), the new content of the ORM reflects the new marking, \( MK' \), of \( PN \) after its transition \( t \) fires. Moreover, in leaving its finishing state, \( t' \) causes the single state of \( SN \) to be turned on thus enabling all the transitions of \( SN \). The new content of the ORM causes the transitions of \( SN \), corresponding to the transitions of \( PN \) that are enabled under the new marking \( MK' \); to enter the ready state, while the other ones remain inactive. Therefore, \( SN \) winds up in \( f(q') \).

Consequently, if there is a change of state from \( q \) to \( q' \) in \( PN \), then there is a chain of states \( f(q),...,f(q') \) in \( SN \) such that none of the intermediate states are in the image of \( f \), and the number of states in the chain is equal to (and thus trivially less than or equal to) \( C=5 \).

e. Condition (3) Is Satisfied

Let \( SN \) be in state \( f(q) \) for some state \( q \) of \( PN \). Then, by construction, there is a unique transition \( t' \) in \( T_s \) such that \( F_t(t')=\text{ready} \) and \( t' \) is selected to fire. As \( t' \) fires, it goes through the starting, executing, and finishing states. This sequence of states consists of states that are not in the image of \( f \). Upon leaving the finishing state of \( t' \), \( SN \) winds up in the state \( [\text{on}],F_t \). In this state, all the transitions of \( T_s \) are enabled. In addition, the content of the ORM associated with \( SN \) has been altered by the execution of the action associated with \( t' \), and corresponds to a new marking of \( PN \). By construction of \( SN \), the transitions of \( PN \) that are enabled under this new marking map exactly to those transitions of \( T_s \) that have their trigger true. These transitions \( t_5 \) would therefore
enter the \textit{ready} state, and be such that \(F_{t}(t_{s}) = \text{ready}\), while all other transitions remain \textit{inactive}. Consequently, the state <=\textit{son}>\( \text{SN} \) is the image of some state \( q' \) of \( \text{PN} \). Since the action associated with \( t' \) reflects exactly the change in marking of \( \text{PN} \) when the transition \( t \) of \( \text{PN} \), such that \( t' = \text{Image}(t) \), fires, we see that \( q' \) belongs to the next state set of \( q \) (i.e., the set of states \( q' \) such that there is a change of state from \( q \) to \( q' \) in \( \text{PN} \)). Consequently, if there exists a chain of states, \( f(q), \ldots, f(q') \) of more than one state in \( \text{SN} \) (for some \( q, q' \), states of \( \text{SN} \)), none of the intermediate states are in the image of \( f \), then there is a change of state from \( q \) to \( q' \) in \( \text{PN} \).

\textbf{f. Condition (4) Is Satisfied}

Analyzing the algorithm given in a., we see that:

1. \( |T_{d}| = |T_{p}| + 1 \) (since \( T_{p} \) is non-empty)
2. \( |S| = |A_{s}| - 1 \) (since \( P \) is non-empty)
3. \( |A_{d}| = 2(|T_{d}| - 1) + 1 = 2|T_{p}| + 1 + 4|T_{p}| \)

Given the fact that in a Petri net, each transition must have at least one incoming and one outgoing arc, we have \( |A_{p}| \geq 2|T_{p}| \) or \(|T_{p}| \geq |A_{p}| / 2 \), and as a result of (3), we obtain \( (3') \) \( |A_{d}| \geq 2|A_{p}| \).

Combining (1), (2) and (3'), we can now write: \( |S| + |T_{d}| + |A_{d}| \geq 2|T_{p}| + 2|T_{p}| + 2|A_{p}| = 2(|P| + |T_{p}| + |A_{p}|) \). Now, since \( P, T_{p}, \) and \( A_{p} \) are non-empty, \( |P| + |T_{p}| + |A_{p}| > 1 + 1 + 1 = 3 \), and consequently, \( |S| + |T_{d}| + |A_{d}| = 3(|P| + |T_{p}| + |A_{p}|) \), that is, the size of \( \text{SN} \) is less than or equal to \( 3 \) times the size of \( \text{PN} \).

3. Petri Nets Do Not Include State Nets

To show that Petri nets do not include state nets, we will exhibit a counter-example. We first describe a special subclass of state nets and then show that Petri nets do not include that subclass.

\textbf{a. NP-State Nets}

Let \( n \) be a positive integer. Consider the state net and its associated ORM of Figure 5. The state net is called an NP-state net, and is denoted \( \text{NP-SN} \). \( \text{NP-SN} \) is such that \( \text{NP-SN} = (S, T_{p}, A_{s}) \) with \( |S| = n + 1 \), \( |T_{p}| = n + 2 \), \( |A_{s}| = n + 3 \), \( |OC| = 2n \), and \( |RS| = n \). Once transition \( \text{init} \) (the initial transition) fires, the states, \( I \) through \( n \), of \( \text{NP-SN} \) are turned on at once, and all the transitions, \( [1] \) through \( [n] \), are enabled. We call \( q0 \) this special state of \( \text{NP-SN} \).

In \( q0 \), since the prior state conjunctions of transitions \( [1] \) through \( [n] \) are mutually exclusive, any number of transitions may fire, and the subset of transitions that will fire is only dependent on the values of the triggers \( T1 \) through \( Tn \). Since each trigger can be either true or false, there are \( 2^n \) unique possible next states from \( q0 \).

\textbf{b. Petri Nets Do Not Include NP-State Nets}

Suppose that Petri nets do include state nets. We will show that we reach a contradiction, namely that condition (2) of the definition of class inclusion cannot be satisfied.

If Petri nets include state nets, then there exist constants \( C \) and \( D \) such that for any state net, \( \text{SN} = (S, T_{s}, A_{s}) \), there exists a Petri net, \( \text{PN} = (P, T_{p}, A_{p}) \), and a one-to-one function \( f \), from the states of \( \text{SN} \) to the states of \( \text{PN} \), such that \( |P| + |T_{p}| + |A_{p}| \)

\( D(|S| + |T_{s}| + |A_{s}| + |OC| + |RS|) \), and if \( q \) and \( q' \) are states of \( \text{SN} \) such that there is a change of state from \( q \) to \( q' \) in \( \text{SN} \), then there exists a chain, \( f(q), \ldots, f(q') \) of at most \( C \) states in \( \text{PN} \).

Let \( \text{SN} \) be a NP-state net, denoted \( \text{NP-SN} \). The size of \( \text{NP-SN} \) is given by \((n+1)+(n+2)+(n+3)+2n+n\leq 6n+6(n+1)\). But \( 6(n+1)+2n \leq 12(n+1) \) for all \( n \). Consequently, \( D(12n) = (12D)n \) is an upper bound on the size of \( \text{PN} \), and hence an upper bound on the number of transitions of \( \text{PN} \).

Let \( f(q) \) be a state of \( \text{PN} \) for some state \( q \) of \( \text{NP-SN} \), and consider chains of states of length \( k \) in \( \text{PN} \), with \( f(q) \) as a starting state. Since \( (12D)n \) is an upper bound on the number of transitions in \( \text{PN} \), and since state changes in \( \text{PN} \) are the result of a single transition firing, then, starting with \( f(q) \), and in each subsequent state, \( \text{PN} \) can choose to fire any one of at most \((12D)n \) transitions. Hence, \((12D)n^k = (12D)^k n^k \) is an upper bound on the maximum number of unique final states for all chains in \( \text{PN} \) of length \( k \) with \( f(q) \) as an initial state.

Extending the above result, we get \( P(n) = (12D)n^1 + (12D)^2 n^3 + \ldots + (12D)^C n^C \) as an upper bound on the maximum number of unique final states for all chains in \( \text{PN} \) of length less than or equal to \( C \), with \( f(q) \) as an initial state. Note that \( P(n) \) is a polynomial function of \( n \).

Consider \( \text{NP-SN} \) in the state \( q0 \) described in a. From \( q0 \), \( \text{NP-SN} \) has \( E(n) = 2^n \) unique possible next states, and \( E(n) \) is an exponential function of \( n \). Now, we know that there exists an integer \( n \), such that for all \( n, n' \), we have \( P(n) < E(n) \). Assume that \( \text{NP-SN} \) is the NP-state net such that \( |S| = n + 1 \), \( |T_{d}| = n + 2 \), \( |A_{s}| = n + 3 \), \( |OC| = 2n' \), and \( |RS| = n' \). In state \( q0 \), \( \text{NP-SN} \) has \( E(n') \) unique possible next states. But \( \text{PN} \) in state \( f(q0) \) has \( P(n') \) as an upper bound on the maximum number of unique final states for all chains of length \( C \) or less. Thus \( P(n') \) is an upper bound on the maximum number of unique states that \( \text{PN} \), in state \( f(q0) \), can use to match the next states of \( \text{NP-SN} \) in state \( q0 \).

Since \( P(n') < E(n') \), and the function \( f \) is one-to-one from the states of \( \text{NP-SN} \) to the states of \( \text{PN} \), there must exist a next possible state, \( q' \), of \( \text{NP-SN} \) in state \( q0 \), such that there are no chains of states in \( \text{PN} \) from \( f(q0) \) to \( f(q') \) of length less than or equal to \( C \). Consequently, \( \text{PN} \) fails to satisfy condition (2) of the definition of class inclusion.

4. Conclusion

It follows immediately from Section 2, Section 3, and the definition of modeling power of Section 1, that state nets have more modeling power than Petri nets.

IV. CONCLUDING REMARKS

Using the notion of modeling power defined by Peterson, we have compared two computational models: Petri nets and state nets. After defining Petri nets and state nets, we proved that basic state nets have more modeling power than extended Petri nets. In particular, for certain basic state nets, there is a state from which all possible transitions cannot be captured, without exponential blow-up, by an equivalent extended Petri net.
We took a minimal-set approach in the proof. The definition of state nets that we used is the necessary and sufficient subset of the full definition, to successfully complete the proof. Several important features were intentionally left-out. They all contribute to making state nets an attractive modeling device. These features include object orientation, high-level states and transitions, general constraints, exception handling, real-time constraints, and spawning new threads of control. All these state-net features are discussed in more detail in [EKW 92]. A formal definition for OSA, which includes state nets, has also been developed.

**BIBLIOGRAPHY**

[Cam 89]

[CEW 92]

[Dav 90]

[EKW 92]

[G-C 91]

[Har 87]

[JeR 91]

[Mor 82]

[Pet 81]

[Zav 82]