ON ADDING CONCURRENCY TO THE FORMAL DEVELOPMENT METHODOLOGY(FDM)

Mark Nixon

Jeanette Wing

CONTENTS

1. INTRODUCTION

This paper seeks to define an approach to adding concurrency capability to FDM. In this context, the *Ina Jo** specification language and the system of logic underlying the FDM tools are treated as an extended first-order predicate logic (FOPL). This approach preserves the intent of FDM as far as we have been able to determine except for certain foundational issues which, as we point out below, have not been adequately formalized in the past. In the following sections, we give precise definitions of those aspects of FDM's syntax, semantics and proof-theory affected by our approach and then show how these may be extended to encompass a class of deterministic and non-deterministic temporal operators used in the expression of concurrent system requirements. A short but highly illustrative example then follows, together with a discussion of how it uses the FDM concurrency extensions to express a simple liveness property. The final section contains detailed proofs of all derived theorems and rules of inference presented below.

We have focussed our efforts on defining a fully formal syntax, semantics and proof theory for the temporal extensions to FDM for single-level *Ina Jo* specifications. By focussing on single-level specifications, we gain the immediate advantage that we deal with only one level of granularity in the state space. *Mappings*, used in multi-level specifications, introduce possibly many levels of granularity. Not until a better understanding of their current formal meaning and intended use is achieved, can mappings be addressed at the same level of formal detail as what follows. One of the most significant results of this work is that it provides a formal definition of single-level *Ina Jo* specifications.

There are two practical matters bearing on the extension of FDM to temporal logic that deserve mention at the outset. First, our approach would be worth implementing if FDM is not near the end of its life cycle. It might be more cost effective to build a next-generation formal verification system which takes the temporal logic approach into account at the outset. Secondly, the sophistication required of FDM users will be increased by the expressibility the extensions bring to the *Ina Jo* language. Greater expressibility requires learning more linguistic distinctions and the associated axioms, theorems and rules that involve them. This is a natural cost for greater expressibility, however.

What we have done for the FDM concurrency enhancement is provide for assertions whereby FDM users may state assumptions that currently are not expressible in the *Ina Jo* language, e.g., that the state space implicit in a specification is complete or that the structure of time implicit in a specification is deterministic and linear.

2. PRELIMINARIES

The formal proof system used for the *Ina Jo* language is assumed to be standard first-order predicate calculus with equality with the usual axioms and rules of inference, e.g., substitution for equality, modus ponens, and generalization. In order to define the nondeterministic state machines underlying the *Ina Jo* language and its relation to FDM's proof system, we need to define the class of models from which state machines are constructed, and the notions of truth and validity for these models. In what follows, we provide these definitions first for the current *Ina Jo* language, and then make necessary extensions to the definitions to handle our temporal logic enhancements.

---

* *Ina Jo* is a Trademark of System Development Corporation, A Burroughs Company.

These enhancements are based largely upon and combine formal techniques due to Ben-Ari, Kripke, Manna and Pnueli ([BP80], [Kr63], [Man81], [MP81a], [MP81b]). They represent traditionally well-founded extensions to the sort of first-order predicate logic underlying FDM.

3. INTERPRETATION OF *INA JO* ASSERTIONS

3.1 Syntax

The extended BNF for the *Ina Jo* assertion language is summarized as follows. The usual order of precedence of Boolean connectives is used and the elimination of redundant parentheses is allowed.

```
Assn    ::=    ~Assn | Assn BinOp Assn | `('Assn`)' | Assn `=>' Assn `<>' Assn
             | Quant Binding {, Binding} `('Assn`)' | Term `=' Term | Term
Term    ::=    Var | `N"'Var | Func_Name`('Term {, Term}`)'
             | `N"'Func_Name`('Term {, Term}`)'
BinOp   ::=    `&' | `|' | `->' | `<->'
Quant   ::=    `A"' | `E"'
Binding ::=    Id {, Id} : Type_Name
```

3.2 *Ina Jo* Models, Truth, and Validity

Adapting the methods of Kripke [Kr63], we define an *Ina Jo* model structure as an ordered quintuple, <INIT, STATE, DOM, TRANS, EVAL>, of:

i. a set, INIT, of alternative initial states;

ii. a set of states, STATE, INIT $\subseteq$ STATE;

iii. a primitive domain, DOM, of typed values;

iv. a finite set, TRANS, of binary state transition relations on STATE;

v. a semantic evaluation function, EVAL, for the class of *Ina Jo* assertions (in Assn).

We first present the definitions for an *Ina Jo* machine, its states, transforms, and computation paths, all in terms of components of the above structure. We then define the two notions, truth in an *Ina Jo* model and validity.

Let ID be a set of identifiers. A *machine*, M$\subseteq$ID, is a set of *Ina Jo state variables* distinguished by declaration in an *Ina Jo* specification in variables. A *state*, s $\in$ STATE, of a machine, M, is a function,

$$s : ID \rightarrow VAL$$

where VAL, the set of primitive semantic values, is defined as follows:

$$AL = DOM^{DOM^i}$$

Let there be the class of all functions, f,

$$(f : DOM^i \rightarrow DOM) \in VAL$$

each mapping i-tuples of DOM into DOM. We consider simple *Ina Jo* state variables, x, as zero-placed functions, so that we have for s ∈ STATE,

$$s(x) \in DOM^{DOM^0} = DOM^{\{<>\}} = DOM$$

as primitive semantic values. For *Ina Jo* state variables, f, of finite non-zero degree *j*, used as function symbols, we have:

$$s(f) \in DOM^{DOM^j} = DOM^{\{<v_1,\ldots,v_j>\}}$$

as primitive semantic values. The type of a function variable, f, is the type of the value of the range of f.

A binary state-transition relation, *tr* ∈ TRANS, is such that for every s_1, s_2 ∈ STATE, there is at least one x ∈ M, {v,v'} ⊆ VAL and <x,v> in s_1 such that:

$$tr\ (s_1,\ s_2)\ \text{iff}\ s_2 = s_1\ [<x,v'>/<x,v>]$$

That is, a state, s_2, is obtained from a state, s_1, and a state-transition, *tr*, by replacing the assignment, <x,v> ∈ s_1, of some element, v ∈ VAL, to some state variable, x ∈ M, with another, <x,v'>. We define the binary relation of *immediate accessibility* among states as the union over all the state transitions so that:

```
R = { <s,t> | for some tr ∈ TRANS, <s,t> ∈ tr }
```

When <s,t> ∈ R we say that t is an immediate successor (descendent) of s. To capture the concept of non-ending time, we require that R be total. Let the relation, R*, of *accessibility* be the reflexive transitive closure of R. We say that a *computation path* is a countable sequence, $<s_j>$, of states of M such that $tr(s_j, s_{j+1})$ for some state transform, *tr* ∈ TRANS ⊆ R.

To obtain semantic interpretations for the assertions in Assn, we first distinguish the STATE-induced assignment function:

```
A: ID X STATE → VAL
```

given, for all s, s' ∈ STATE, and x ∈ ID, by the conditions:

```
A(x) s = s(x),    for x ∈ M.
A(x) s = A(x) s'  for x ∈ (Id - M).
```

The A-induced valuation function:

```
V: Term X STATE X STATE → VAL
```

is given, for all s, s' ∈ STATE, and *Ina Jo* terms, x, t_i, $1 \le i \le n$, and $f(t_1,\ldots,t_n)$, by the recursive equations:

```
V(x) s s'                 = A(x) s
V(N"x) s s'               = A(x) s'
V(f(t1,...,tn)) s s'      = s(f)(V(t1) s s', ..., V(tn) s s')
V(N"f(t1,...,tn)) s s'    = s'(f)(V(t1) s s', ..., V(tn) s s')
```

Here we have defined the semantics of N" in such a way that for non-Boolean functional terms, the operator applies to the denotation of the function symbol but does not distribute to the denotations of the function arguments. This captures the intended interpretation of N" in current *Ina Jo* [SA85, p.24, section 6].

Third, we distinguish the V-induced Boolean-valued interpretation function:

```
EVAL: Assn X STATE X STATE → {true, false} ⊆ VAL
```

given, for all s, s' ∈ STATE such that R(s,s'), by the recursive equations:

```
EVAL(t) s s'               = V(t) s s', for Boolean-valued terms t,
EVAL(t1 = t2) s s'         = V(t1) s s' = V(t2) s s',
EVAL(~a) s s'              = ~EVAL(a) s s',
EVAL(A"x:T (a[x]))) s s'  = ∧ { EVAL(a[x]) s s': ∀v ∈ M (EVAL(v) s s' = EVAL(v) s s' ) },
EVAL(E"x:T (a[x]))) s s'  = ∨ { EVAL(a[x]) s s' : ∀v ∈ M (EVAL(v) s s' = EVAL(v) s s' ) },
EVAL(a1#a2) s s'           = EVAL(a1) s s' # EVAL(a2) s s', for # ∈ BinOp,
EVAL(a1=>a2) s s'          = (EVAL(a1) s s'→ EVAL(a2) s s'),
EVAL(a1=>a2<>a3) s s'      = (EVAL(a1) s s'→ EVAL(a2) s s') &
                             (~EVAL(a1) s s'→ EVAL(a3) s s')
```

where a[x] is an arbitrary ASSN string of *Ina Jo* with free occurrences of x ∈ (ID - M). (The notations, ∧ { v : F } and ∨ { v : F }, used above denote the conjunction and disjunction, respectively, over sets of values, v, satisfying the specified conditions, F.)

The basic formal semantic notions for *Ina Jo* are then defined for *Ina Jo* assertions, a ∈ Assn, over *Ina Jo* model structures, K = <INIT, STATE, DOM, TRANS, EVAL>, as follows:

i. a is *true at* s_i *on K* iff for every A-induced valuation, V, over s_i, s_{i+1} such that R*(s_i,s_{i+1}), EVAL (a) s_i,s_{i+1} = true.

ii. a is *true on K* iff a is true at some initial state s_0∈ INIT on K.

iii. a is *valid* iff a is true on every *Ina Jo* model structure, K, in which case we write |= a.

4. EXTENDING THE *INA JO* LANGUAGE WITH TEMPORAL LOGIC

4.1 Enriching the *Ina Jo* Vocabulary

We add to the syntax for Assn as follows:

```
Assn     ::= ... | UnOp Assn | Assn TBinOP Assn
UnOp     ::= AH" | EH" | AV" | EV" | AN" | EN" | N"
TBinOp   ::= ... | AU" | EU" | AP" | EP" | AB" | EB"
```

The new operators cited above come in two varieties: *deterministic* operators beginning with the letter 'A', and *non-deterministic* operators beginning with the letter 'E'. The intention is to specify which R*-successor states are to be taken into account in evaluating strings in Assn. The deterministic variety call for evaluation across all R*-successor states to a given state (hence the 'A') while the non-deterministic variety call for evaluation in some successor state (hence the 'E' for "exists").

Note that the grammatical role of the 'N"' operator has been generalized to apply to compound Boolean-valued strings in Assn rather than merely to atomic Terms (Boolean and other). The other operators cited above apply to strings in Assn as unary prefix operators, e.g.,

```
ah"(a)   -- henceforth a.
ev"(a)   -- eventually a.
an"(a)   -- in the next state a is the case.
```

or as binary infix operators, e.g.,

```
(a au" b)   -- a is the case until b is the case.
(a ep" b)   -- a's being the case precedes b's being the case.
(a ab" b)   -- a is the case before b is the case.
```

Their formal meanings are captured in the next section which presents extensions to the model-theoretic definitions given in the last section. In section 5.3, we present definitions of the *before* and *precedes* operators in terms of the *henceforth, eventually, next* and *until* operators.

4.2 Extending the EVAL Function

We extend EVAL as follows for non-Boolean terms, t, Boolean atoms a1 and a2, and s, s', s'' ∈ STATE such that R(s,s') and R (s', s''):

```
EVAL(N"(a) s s'    =
 if a is t                 then V(N"t) s s'
 if a is (t1=t2)           then EVAL(t1=t2) s s'
 if a is ~a1               then EVAL(~N"a1) s s'
 if a is A"x:T (a1)        then EVAL(A"x:T (N"a1)) s s'
 if a is E"x:T (a1)        then EVAL(E"x:T (N"a1)) s s'
 if a is (a1 # a2)         then EVAL(N"a1 # N"a2) s s', for # in BinOp
 if a is (a1 => a2 <> a3)  then EVAL(N"a1 => N"a2 <> N"a3) s s'
 if a is #a1               then EVAL(# a1) s' s'', for # in UnOp
 if a is (a1 # a2)         then EVAL(a1 #a2) s' s'', for # in TBinOp

EVAL(AN"(a) s s'       = EVAL(N"a)s s' & ∧ { EVAL(N"a) s' t : R(s',t) }
EVAL(EN"(a) s s'       = EVAL(N"a)s s' | ∨ { EVAL(N"a) s' t : R(s',t) }
EVAL(AH"(a) s s'       = EVAL(a) s s' & ∧ { EVAL(AH"a) s' t : R(s',t) }
EVAL(EH"(a) s s'       = EVAL(a) s s' & ∨ { EVAL(EH"a) s' t : R(s',t) }
EVAL(AV"(a) s s'       = EVAL(a) s s' | ∧ { EVAL(AV"a) s' t : R(s',t) }
EVAL(EV"(a) s s'       = EVAL(a) s s' | ∨ { EVAL(EV"a) s' t : R(s',t) }
EVAL(a1 AU" a2) s s'   = EVAL(a2) s s' | ( EVAL(a1) s s' & ∧ { EVAL(a1 AU" a2) s' t : R(s',t) } ;
EVAL(a1 EU" a2) s s'   = EVAL(a2) s s' | ( EVAL(a1) s s' & ∨ { EVAL(a1 AU" a2) s' t : R(s',t) } ;
```

The definitions of V, truth on an *Ina Jo* model structure, and validity remain the same.

4.2.1 New-Value Operations in Extended FDM

On the formal semantics just presented, the intended interpretation of the three new-value operators, N", AN" and EN", provides values for *Ina Jo* expressions across R-successor states, s', of a state s. In the case of the N" operator, the R-successor, s', is specified. In the case of AN", the operation is taken as the *conjunction* over every R-successor, s', of the value of its operand. In the case of EN", the operation is taken as the *disjunction* over every R-successor, s', of the value of its operand. Note, in particular, that N" has broader application than AN" and EN" since it may take non-Boolean terms as operands.

In the next section, we present the proof-theoretic counterparts to our formal semantical definitions. We give axiom schemata and rules of inference that capture the semantical behavior of the three new-value operators.

4.3 Extending FDM's Deductive Methods

We extend FDM's basis for first-order predicate logic (FOPL) with the following axiom schemata:

```
N1.  |- en"a <-> ~an"~a
N2.  |- an"(a1 -> a2) -> (an"a1 -> an"a2)
N3.  |- an"a -> n"a
N4.  |- n"~a <-> ~n"a
N5.  |- n"(a & b) <-> (n"a & n"b)

A1.  |- av"a <-> ~eh"~a
A2.  |- ah"(a1 -> a2) -> (ah"a1 -> ah"a2)
A3.  |- ah"a -> an"a & an"ah"a
A4.  |- ah"(a -> an"a) -> (a -> ah"a)
A5.  |- (a1 au" a2) <-> (a2 | a1 & an"(a1 au" a2))
A6.  |- (a1 au" a2) -> av"a2

E1.  |- ev"a <-> ~ah"~a
E2.  |- ah"(a1 -> a2) -> (eh"a1 -> eh"a2)
E3.  |- eh"a -> a & en"eh"a
E4.  |- ah"a -> eh"a
E5.  |- ah"(a -> en"a) -> (a -> eh"a)
E6.  |- (a1 eu" a2) <-> (a2 | a1 & en"(a1 eu" a2))
E7.  |- (a1 eu" a2) -> av"a2

Q1.  |- a"x:TYPE ( an"a ) <-> an"a"x:TYPE ( a )
Q2.  |- e"x:TYPE ( an"a ) <-> an"e"x:TYPE ( a )
Q3.  |- a"x:TYPE ( ah"a ) <-> ah"a"x:TYPE ( a )
Q4.  |- a"x:TYPE ( eh"a ) <-> eh"a"x:TYPE ( a )
```

We then add the following primitive rules of inference:

```
R1: NEC                  (necessitation)
     |- a
    ---------
     |- ah"a

R2: ENINST               (en"-instantiation)
     |- en"a
     |- n"a -> b         where b has no occurrences of terms prefixed by n".
    ---------
     |- b

R3: ANGEN                (an"-generalization)
     |- n"a
    ---------
     |- an"a
```

R2 allows the elimination of EN" in favor of N" for the duration of a sub-proof ending with a predicate expression without occurrences of N". This corresponds to the usual sort of natural deduction rule for existential quantification elimination in which the existential quantifier is dropped and all occurrences of its bound variable are treated as free occurrences for the duration of the sub-proof. The sub-proof must end in a step without such free occurrences of the initially bound variable.

R3 corresponds, on the other hand, to a rule of universal generalization in which the universal quantification of a predicate expression may be deduced from a restricted sub-proof of that expression. The sub-proof must not refer to any steps occurring before and outside its scope with free occurrences of the quantified variable. The two rules, R2 and R3 together, capture the idea that the AN" and EN" new-value operators implicitly bind R-successor state references and that occurrences of N" indicate unbound or free R-successor state references.

We next extend the stock of temporal operators through the following four forms of syntactic elimination:

```
AB:  (a ab" b)   =df.   av"b -> (~b au" a)
EB:  (a eb" b)   =df.   av"b -> (~b eu" a)
AP:  (a ap" b)   =df.   ~(~a au" b)
EP:  (a ep" b)   =df.   ~(~a eu" b)
```

Note the differences in intended meaning among the *before*, *precedes* and *until* varieties of operators. The *precedes* operators imply that a *precedes* b only if b is not already the case in a given state, whereas the *before* operators do not.

We now list certain useful derived rules of inference. Complete proofs as well as an explanation of all proof notation used below are contained in the final section.

```
DR0: AHIMP
     |- a -> b
    -----------------
     |- ah"a -> ah"b

DR1: EHIMP
     |- a -> b
    -----------------
     |- eh"a -> eh"b

DR2: AVIMP
     |- a -> b
    -----------------
     |- av"a -> av"b

DR3: EVIMP
     |- a -> b
    -----------------
     |- ev"a -> ev"b

DR4: ANI
     |- a
    ---------
     |- an"a

DR5: ANIMP
     |- a -> b
    -----------------
     |- an"a -> an"b
```

```
DR6: ENIMP
      |- a -> b
    ------------------
      |- en"a -> en"b

DR7: ENI
      |- a
    ----------
      |- en"a

DR8: CIA (computational induction rule)
      |- a -> an"a
    --------------
      |- a -> ah"a

DR9: CIE (computational induction rule)
      |- a -> en"a
    --------------
      |- a -> eh"a

DR10: BIA (backward induction rule)
      |- an"a -> a
    --------------
      |- av"a -> a

DR11: BIE (backward induction rule)
      |- en"a -> a
    --------------
      |- ev"a -> a

DR12: NPA (next to present rule)
      |- (an"a <-> an"b) -> (a <-> b)
      |- a -> av"(a & b)
      |- b -> av"(a & b)
    ----------------------------------
      |- a <-> b

DR13: NPE (next to present rule)
      |- (en"a <-> en"b) -> (a <-> b)
      |- a -> av"(a & b)
      |- b -> av"(a & b)
    ----------------------------------
      |- a <-> b

DR14: WNPA (weak next-to-present rule)
      |- (an"a -> an"b) -> (a -> b)
      |- a -> av"(a & b)
      |- b -> av"(a & b)
    ----------------------------------
      |- a -> b

DR15: WNPE (weak next-to-present rule)
      |- (en"a -> en"b) -> (a -> b)
      |- a -> av"(a & b)
      |- b -> av"(a & b)
    ----------------------------------
      |- a -> b
```

```
DR16: TSUBST <->
  Let c' be the result of replacing an occurrence
  of a subformula a1 in c by a2.  Then
     |- a1 <-> a2
    --------------
     |- c <-> c'
```

The backward induction rules, BIA and BIE, may be used in proving specification correctness theorems in FDM extended with the temporal operators. The initial condition correctness theorem of the example given below in extended FDM, for instance, uses BIE.

We now list certain useful theorems of *Ina Jo* extended with temporal logic. Complete proofs are contained in the final section.

From *The Logic of Nexttime* [BP80]:

```
T1:    |- ah"a -> a
T2:    |- ah"a -> av"a
T3:    |- an"(a -> b) -> (en"a -> en"b)
T4:    |- ah"(a -> b) -> (av"a -> av"b)
T5:    |- eh"a -> en"a
T6:    |- an"a -> en"a
T7:    |- ah"(a & b) <-> (ah"a & ah"b)
T8:    |- eh"(a & b) <-> (eh"a & eh"b)
T9:    |- an"(a & b) <-> (an"a & an"b)
T10:   |- en"(a & b) -> (en"a & en"b)
T11:   |- an"a & en"b -> en"(a & b)
T12:   |- ah"a & eh"b -> eh"(a & b)
T13:   |- ah"a <-> a & an"ah"a
T14:   |- eh"a <-> a & en"eh"a
T15:   |- ah"a <-> ah"ah"a
T16:   |- eh"a <-> eh"eh"a
T17:   |- eh"(a -> an"a) -> (a -> eh"a)
T18:   |- av"ah"a -> ah"av"a
T19:   |- eh"((a | eh"b) & (eh"a | b)) <-> (eh"a | eh"b)
T20:   |- an"ah"a <-> ah"an"a
T21:   |- en"eh"a -> eh"en"a
```

Some *Lewis S4* [HC68] theorem schemata:

```
T22:   |- a -> av"a
T23:   |- a -> ev"a
T24:   |- av"a <-> av"av"a
T25:   |- ev"a <-> ev"ev"a
```

Distribution laws for *henceforth* and *eventually*:

```
T26:   |- ah"(a -> b) -> (ev"a -> ev"b)
T27:   |- av"(a | b) <-> (av"a | av"b)
T28:   |- ev"(a | b) <-> (ev"a | ev"b)
T29:   |- av"(a & b) -> (av"a & av"b)
T30:   |- ev"(a & b) -> (ev"a & ev"b)
T31:   |- (ah"a | ah"b) -> ah"(a | b)
T32:   |- (eh"a | eh"b) -> eh"(a | b)
T33:   |- (ah"a & ev"b) -> ev"(a & b)
T34:   |- (eh"a & av"b) -> ev"(a & b)
```

Some theorem schemata concerning *next-time*:

```
T35:   |- an"a -> av"a
T36:   |- en"a -> ev"a
T37:   |- (an"a | an"b) -> an"(a | b)
T38:   |- en"(a | b) <-> (en"a | en"b)
```

```
T39:  |- an"(a <-> b) -> (an"a <-> an"b)
T40:  |- an"(a <-> b) -> (en"a <-> en"b)
```

Some theorem schemata concerning *next-time* and *eventually*:

```
T41:  |- av"an"a -> an"av"a
T42:  |- ev"en"a <-> en"ev"a
T43:  |- av"an"a -> av"a
T44:  |- ev"en"a -> ev"a
T45:  |- av"a <-> (a | an"av"a)
T46:  |- ev"a <-> (a | en"ev"a)
T47:  |- (a & av"~a) -> ev"(a & an"~a)
T48:  |- (a & ev"~a) -> ev"(a & en"~a)
```

Theorem schemata for *until*:

```
T49:  |- a & (~a au" b) -> b
T50:  |- ~b & (a au" b) -> a
T51:  |- b -> (a au" b)
T52:  |- a & an"(a au" b) -> (a au" b)
T53:  |- av"(a au" b) <-> (av"a au" av"b)
T54:  |- ev"(a eu" b) <-> (ev"a eu" ev"b)
T55:  |- (a au" b) -> (a eu" b)
T56:  |- an"a au" an"b -> an"(a au" b)
```

Theorem schemata for *precedes*:

```
T57:  |- a & ~b -> a ap" b
T58:  |- (a ap" b) -> ~b
T59:  |- (a|b|c)&(a ap" b)&(b ap" c) -> (a ap" c)
T60:  |- (a ap" b) -> (b -> c)
```

Theorem schemata for *before*:

```
T61:  |- a -> (a ab" b)
T62:  |- (a|b|c) & (a ab" b) & (b ab" c) -> (a ab" c)
T63:  |- (a -> d) & (a ab" b) & (b ab" c) -> (c -> d)
```

5. AN EXAMPLE OF A LIVENESS PROPERTY IN EXTENDED FDM

The following specification written in *Ina Jo* extended with temporal operators contains a liveness property (expressed as a criterion with the non-deterministic new-value operator, ev").

```
specification LIVE
    level tls
        variable
            x : integer;
        initial
            x>0 & ah"( an"(n"x = x-1) )
        criterion
          /**  if x>0 then eventually x=0  **/
            x>0 -> ev"( x=0 )
        transform decrement external
          /**  x is decremented in every next state  **/
            effect
                an"(n"x = x-1)
    end tls
```

The correctness theorems to be generated by the *Ina Jo* processor for such a specification under extended FDM would include the following:

```
        THEOREM FROM LEVEL TLS FOR: INITIAL CONDITIONS

   |- ev"  ( ah"(an"(n"x=x-1)) & x>0 -> (x>0 -> ev"(x=0)) )
       -> ( ah"(an"(n"x=x-1)) & x>0 -> (x>0 -> ev"(x=0)) )

proof
  1.  | | en"  ( ah"(an"(n"x=x-1)) & x>0 -> (x>0 -> ev"(x=0)) )      assume
  2.  | | | ~( ah"(an"(n"x=x-1)) & x>0 -> (x>0 -> ev"(x=0)) )       assume
  3.  | | | ah"(an"(n"x=x-1)) & x>0                                  2: simp
  4.  | | | ~(x>0 -> ev"(x=0))                                       2: simp
  5.  | | | x>0                                                      3: simp
  6.  | | | ~ev"(x=0)                                                4: simp
  7.  | | | en"(~(ah"(an"(n"x=x-1)) & x>0 ) | (x>0 -> ev"(x=0)))     1: FOPL
  8.  | | | en"((~ah"(an"(n"x=x-1)) | ~(x>0) ) | (x>0 -> ev"(x=0))) 7: FOPL
  9.  | | | en"~ah"(an"(n"x=x-1)) | en"~(x>0) | en"(x>0 -> ev"(x=0))  8,T38: subst <->
  10. | | | an"ah"(an"(n"x=x-1))                                     3,A3: simp, FOPL
  11. | | | ~en"~ah"(an"(n"x=x-1))                                   10,N1: FOPL
  12. | | | en"~(x>0) | en"(x>0 -> ev"(x=0))                        9,11: ds
  13. | | | an"(x>=0)                                                5,3: simp, arith
  14. | | | en"(x=0) -> ev"(x=0)                                     T36
  15. | | | ~en"(x=0)                                                6,14: FOPL
  16. | | | an"~(x=0)                                                15,N1: DNI, subst <->, DNE
  17. | | | an"(x>=0 & ~(x=0))                                       13,16,T9: FOPL
  18. | | | an"(x>0)                                                 17: arith
  19. | | | ~en"~(x>0)                                               18,N1: DNI, subst <->, DNE
  20. | | | en"(x>0 -> ev"(x=0))                                     12,19: ds
  21. | | | en"~(x>0) | en"ev"(x=0)                                  20,T38: FOPL
  22. | | | en"ev"(x=0)                                              19,21: ds
  23. | | | x=0 | en"ev"(x=0)                                        22: addition
  24. | | | ev"(x=0)                                                 23,T46: subst <->
  25. | | ah"(an"(n"x=x-1)) & x>0 -> (x>0 -> ev"(x=0))               2-24: IP
  26. | en"  ( ah"(an"(n"x=x-1)) & x>0 -> (x>0 -> ev"(x=0)) )        1-25: CP
          -> ( ah"(an"(n"x=x-1)) & x>0 -> (x>0 -> ev"(x=0)) )
  27. | ev"  ( ah"(an"(n"x=x-1)) & x>0 -> (x>0 -> ev"(x=0)) )        26: BIE
          -> ( ah"(an"(n"x=x-1)) & x>0 -> (x>0 -> ev"(x=0)) ) )
qed
```

which is entailed by the following analogue of what FDM would currently require:

```
    |- ( ah"(an"(n"x=x-1)) & x>0 ) -> ( x>0 -> ev"(x=0) )
```

since the proposed theorem has the analogue to the current one as consequent. It follows that the proposed theorem form:

```
    |- ev"( INITIAL -> CRITERION )
         -> ( INITIAL -> CRITERION )
```

preserves the theoremhood of initial condition correctness theorems under current FDM.

It is noteworthy that the the initial condition of specification LIVE contains the subformula

```
    ah"( an"(n"x=x-1) )
```

which is an instance of the following schema for that particular specification:

ah"(r_0 & c_0 | ... | r_n & c_n)

the rules and theorems for temporal operators to the proof task. Note that the non-deterministic new-value operator, en", is used to express the new value of the criterion for the reason that the state machine underlying an *Ina Jo* specification is assumed to be non-deterministic in both current and extended FDM.

```
THEOREM FROM LEVEL TLS FOR: DECREMENT

|- an"(n"x=x-1) & (x>0 -> ev"(x=0)) -> en"(x>0 -> ev"(x=0))

 proof
   1.  | | ~( an"(n"x=x-1)&(x>0->ev"(x=0)) -> en"(x>0->ev"(x=0)) )    assume
   2.  | | an"(n"x=x-1) & (x>0 -> ev"(x=0)) & ~en"(x>0 -> ev"(x=0))   1: FOPL
   3.  | | an"(n"x=x-1)                                                2: simp
   4.  | | x>0 -> ev"(x=0)                                             2: simp
   5.  | | an"(x>0 & ~ev"(x=0))                                        2,N1: simp, FOPL
   6.  | | an"(x>0)                                                    5,T9: subst <->, simp
   7.  | | an"~ev"(x=0)                                                5,T9: subst <->, simp
   8.  | | ~en"ev"(x=0)                                                7,N1: FOPL, subst <->
   9.  | | x>1                                                         3,6: arith
   10. | | ev"(x=0)                                                    4,9: arith, mp
   11. | | x=0 | en"ev"(x=0)                                           10,T46: subst <->
   12. | | en"ev"(x=0)                                                 9,11: arith, ds
   13. | an"(n"x=x-1) & (x>0 -> ev"(x=0)) -> en"(x>0 -> ev"(x=0))     1-12: CP
 qed
```

It should be noted that FDM has hitherto provided no support for the expression of *liveness* properties such as the *eventuality* requirement we imposed on states satisfying the antecedent condition of our criterion. Other examples illustrating the applicability of the *until* (au" and eu"), *precedes* (ap" and ep") and *before* (ab" and eb") operators can be constructed with little difficulty. Proving that such properties are preserved under arbitrary state transitions, however, does require more sophistication of both the ITP software and the FDM user.

6. SUMMARY OF CHANGES PROPOSED FOR FDM WITH TEMPORAL LOGIC

In this section, we summarize the changes required to extend FDM with temporal logic in the interest of specifying concurrent system requirements. We explain each of the changes proposed. Note that none of the changes we propose is independent of any of the others except those involving the generalization of the n" operator's syntax and application.

6.1 Syntax, Processing and Theorem Proving with Temporal Operators

To add the capability to both the *Ina Jo* processor and the ITP of processing assertions involving the twelve temporal operators would call for extensive revision and addition of CWIC source code.

6.1.1 Explanation

The *Ina Jo* processor must be extended to recognize, manipulate and generate theorems for *Ina Jo* specifications involving the use of the temporal operators. The ITP must be extended to apply the primitive axioms and rules of inference presented in section 5.3. Moreover, some method must be introduced to allow for the proof and application of derived rules and theorems.

The last point highlights the need for an extensible ITP enhanced so that the FDM user may introduce axiom schemata and rules of inference in a more general form. Under the ITP's current design, we have estimated the amount of CWIC code needed to encompass the 21 axiom schemata and three primitive rules of inference proposed.

6.2 Generalizing the Syntax of the New-Value Operator

The modifications to *Ina Jo*'s grammar set out in section 5.1 include the generalization of the n" operator from the class of *Ina Jo* terms to the class of *Ina Jo* assertions.

6.2.1 Explanation

Axioms N3 and N4 of our proposed extensions to the deductive system underlying FDM are certified by our semantical extensions for n" (section 5.2) and preserve the intended meaning of n" under current FDM. Note that because of axioms N3 and N4, simplification laws which distribute n" inward across all non-temporal logical opators are justified. Thus, occurrences of n" can always be distributed inward so as to prefix only atomic Boolean elements (variables or constants) or other occurrences of n".

Under current FDM, expressions such as n"n"x are flagged as syntax errors since nested applications of n" are not allowed. The *Ina Jo* processor and the ITP would, therefore, have to be modified to allow for such expressions under our proposed temporal logic extensions.

6.3 Generalizing the Application of New-Value Operators

Our example of a specification written in *Ina Jo* extended with temporal operators used a subformula of the *initial condition* containing an occurrence of the deterministic new-value operator, an". We also suggested how allowing occurrences of an" and en" within *invariants* would allow FDM users to make explicit those assumptions they wish to make about the structure of time underlying their specifications. Occurrences of these operators should be admitted in such cases.

6.3.1 Explanation

In general, the current restrictions against occurrences of the new-value operators in declarations providing explicit assumptions about a specification's state machine and its operations (as against its set of requirements) are not essential to the methodology of FDM.

6.4 Requiring Conjoint Initial Condition and Criterion Consistency Proofs

This extension to FDM would not modify the intended methodology underlying FDM. There are indeed cases under current FDM in which the inconsistency of a criterion would not be detected, viz., those in which both the initial condition and the criterion were separately inconsistent. For, in such cases, both the initial condition correctness theorem and the transform correctness theorems would prove by virtue of having inconsistent antecedents. As unlikely as such a case might be, it is not covered at present.

6.4.1 Explanation

The purpose of the initial condition correctness theorem is to guarantee that a specification's criterion is true in the initial state underlying its semantic model, i.e., are satisfied by the *Ina Jo* state machine intended by a specification.

However, there are weaker formulae of the system of temporal logic we propose to add to FDM, the proofs of which in conjunction with a separate proof of criterion consistency also provide this guarantee (see the discussion of our example, the initial condition proven for it, and the argument that criterion consistency is the only additional requirement needed to justify initial condition correctness).

6.5 Adding the Proposed Form of Initial Condition Correctness Theorem

The proposed form of the initial condition correctness conjecture to be generated by the *Ina Jo* processor under FDM extended with temporal logic is:

```
|- ev"(INITIAL -> CRITERION)
   -> (INITIAL -> CRITERION)
```

This form of initial condition correctness theorem is weaker than that currently generated but, as we discussed above, is sufficiently strong to capture the notion of initial condition correctness under FDM as it is at present.

6.5.1 Explanation

The following proof schema summarizes how the current form of initial condition correctness is preserved under FDM extended with temporal logic:

```
1. |-  INITIAL -> CRITERION                                assume
2. |- (INITIAL -> CRITERION)                               FOPL tautology
        -> ( ev"(INITIAL -> CRITERION)
             -> (INITIAL -> CRITERION) )
3. |- ev"(INITIAL -> CRITERION)                            1,2: modus ponens
      -> (INITIAL -> CRITERION)
```

So FDM's current notion of initial condition correctness entails the proposed notion of initial condition correctness.

The following proof schema summarizes how the consistency of a specification's criterion with its initial condition under our proposed temporal logic extensions may be used to derive initial condition correctness in the sense of FDM at present:

```
1. | ev"(INITIAL & CRITERION)                             by initial-criterion consistency
2. | (INITIAL & CRITERION) -> (INITIAL -> CRITERION)      FOPL tautology
3. | ev"(INITIAL & CRITERION)                             2: EVIMP
     -> ev"(INITIAL -> CRITERION)
4. | ev"(INITIAL -> CRITERION)                             1,3: modus ponens
5. |- ev"(INITIAL -> CRITERION)                            proposed IC theorem
     -> (INITIAL -> CRITERION)
6. | INITIAL -> CRITERION                                  1,5: modus ponens
```

7. ANNOTATED PROOFS

In this section, we give complete proofs of the derived theorems and rules of inference presented above.

Proofs are given in a natural deduction style using the following notations:

```
subst <->       =df. substitutivity of material equivalents.
tsubst <->      =df. substitutivity of temporal equivalents.
FOPL            =df. tautology or simple consequence of first-order predicate logic.
contraposition  =df. from a->b to infer ~b->~a
mp              =df. modus ponens, from a and a->b to infer b
ds              =df. disjunctive syllogism, from ~a and a|b to infer b
add             =df. addition, from a to infer a|b
syll            =df. hypothetical syllogism, from a->b and b->c to infer a->c
dni             =df. double-negation introduction
dne             =df. double-negation elimination
simp            =df. conjunction elimination
ip              =df. indirect proof
cp              =df. conditional proof
```

7.1 DERIVED RULES

```
DR0: AHIMP
      |- a -> b
    -----------------
      |- ah"a -> ah"b
  proof
    1.  | |- a -> b              assume
    2.  | |- ah"(a -> b)         1: NEC
    3.  | |- ah"a -> ah"b        2,A2: mp
  qed

DR1: EHIMP
      |- a -> b
    -----------------
      |- eh"a -> eh"b
  proof
    1.  | |- a -> b              assume
    2.  | |- ah"(a -> b)         1: NEC
    3.  | |- eh"a -> eh"b        2,E2: mp
  qed

DR2: AVIMP
      |- a -> b
    -----------------
      |- av"a -> av"b
  proof
    1.  | |- a -> b              assume
    2.  | |- ~b -> ~a            1: contraposition
    3.  | |- ah"~b -> ah"~a      2: AHIMP
    4.  | |- ~ah"~a -> ~ah"~b    3: contraposition
    5.  | |- av"a -> av"b        4,A1: subst <->
  qed

DR3: EVIMP
      |- a -> b
    -----------------
      |- ev"a -> ev"b
  proof
```

```
    1.  | |- a -> b               assume
    2.  | |- ~b -> ~a             1: contraposition
    3.  | |- eh"~b -> eh"~a       2: EHIMP
    4.  | |- ~eh"~a -> ~eh"~b     3: contraposition
    5.  | |- ev"a -> ev"b         4,E1: subst <->
  qed

DR4: ANI
      |- a
    ---------
      |- an"a
  proof
    1.  | |- a                    assume
    2.  | |- ah"a                 1: NEC
    3.  | |- an"a                 2,A3: mp
  qed

DR5: ANIMP
      |- a -> b
    -----------------
      |- an"a -> an"b
  proof
    1.  | |- a -> b               assume
    2.  | |- an"(a -> b)          1: ANI
    3.  | |- an"a -> an"b         2,N2: mp
  qed

DR6: ENIMP
      |- a -> b
    -----------------
      |- en"a -> en"b
  proof
    1.  | |- a -> b               assume
    2.  | |- ~b -> ~a             1: contraposition
    3.  | |- an"~b -> an"~a       2: ANIMP
    4.  | |- ~an"~a -> ~an"~b     3: contraposition
    5.  | |- en"a -> en"b         4,A1: subst <->
  qed

DR7: ENI
      |- a
    ---------
      |- en"a
  proof
    1.  | |- a                    assume
    2.  | |- ah"a                 1: NEC
    3.  | |- eh"a                 2,E4: mp
    4.  | |- eh"a -> en"eh"a      E3: FOPL
    5.  | |- en"eh"a -> en"a      E3: FOPL, ENIMP
    6.  | |- en"a                 3,4,5: FOPL
  qed

DR8: CIA (computational induction rule)
      |- a -> an"a
    --------------
      |- a -> ah"a
  proof
    1.  | |- a -> an"a            assume
    2.  | |- ah"(a -> an"a)       1: NEC
    3.  | |- a -> ah"a            2,A4: mp
  qed
```

```
DR9: CIE (computational induction rule)
     |- a -> en"a
    --------------
     |- a -> eh"a
  proof
    1.  | |- a -> en"a          assume
    2.  | |- ah"(a -> en"a)     1: NEC
    3.  | |- a -> eh"a          2,E5: mp
  qed

DR10: BIA (backward induction rule)
     |- an"a -> a
    --------------
     |- av"a -> a
  proof
    1.  | |- an"a -> a          assume
    2.  | |- ~a -> ~an"a        1: contraposition
    3.  | |- ~a -> en"~a        2,N1: dni, subst <->
    4.  | |- ~a -> eh"~a        3: CIE
    5.  | |- ~a -> ~av"a        4,A1: dni, subst <->
    6.  | |- av"a -> a          5: contraposition
  qed

DR11: BIE (backward induction rule)
     |- en"a -> a
    --------------
     |- ev"a -> a
  proof
    1.  | |- en"a -> a          assume
    2.  | |- ~a -> ~en"a        1: contraposition
    3.  | |- ~a -> an"~a        2,N1: dni, subst <->
    4.  | |- ~a -> ah"~a        3: CIA
    5.  | |- ~a -> ~ev"a        4,E1: dni, subst <->
    6.  | |- ev"a -> a          5: contraposition
  qed

DR12: NPA (next to present rule)
     |- (an"a <-> an"b) -> (a <-> b)
     |- a -> av"(a & b)
     |- b -> av"(a & b)
    ----------------------------------
     |- a <-> b
  proof
    1.  | |- a -> av"(a & b)                     assume
    2.  | |- b -> av"(a & b)                     assume
    3.  | |- (a | b) -> av"(a & b)               1,2: FOPL
    4.  | |- (a & b) -> (a <-> b)                FOPL
    5.  | |- av"(a & b) -> av"(a <-> b)          4: AVIMP
    6.  | |- (an"a <-> an"b) -> (a <-> b)        assume
    7.  | |- an"(a <-> b) -> (an"a <-> an"b)     N2: FOPL
    8.  | |- an"(a <-> b) -> (a <-> b)           6,7: FOPL
    9.  | |- av"(a <-> b) -> (a <-> b)           8: BIA
    10. | |- (a | b) -> (a <-> b)                3,5,9: FOPL
    11. | |- ~(a | b) | (a & b | ~a & ~b)       10: FOPL
    12. | |- ~a & ~b | (a & b | ~a & ~b)        11: FOPL
    13. | |- a & b | ~a & ~b                     12: FOPL
    14. | |- a <-> b                            13: FOPL
  qed

DR13: NPE (next to present rule)
     |- (en"a <-> en"b) -> (a <-> b)
```

```
     |- a -> av"(a & b)
     |- b -> av"(a & b)
   --------------------------------
     |- a <-> b
  proof is similar to proof of DR12 using T3 in place of N2.

DR14: WNPA (weak next-to-present rule)
     |- (an"a -> an"b) -> (a -> b)
     |- a -> av"(a & b)
     |- b -> av"(a & b)
   --------------------------------
     |- a -> b
  proof
    1.   | |- a -> av"(a & b)                    assume
    2.   | |- b -> av"(a & b)                    assume
    3.   | |- (a | b) -> av"(a & b)              1,2: FOPL
    4.   | |- (a & b) -> (a -> b)                FOPL
    5.   | |- av"(a & b) -> av"(a -> b)          4: AVIMP
    6.   | |- (an"a -> an"b) -> (a -> b)         assume
    7.   | |- an"(a -> b) -> (an"a -> an"b)      N2
    8.   | |- an"(a -> b) -> (a -> b)            6,7: FOPL
    9.   | |- av"(a -> b) -> (a -> b)            8: BIA
    10.  | |- (a | b) -> (a -> b)                3,5,9: FOPL
    11.  | |- ~(a | b) | (~a | b)                10: FOPL
    12.  | |- ~a & ~b | ~a | b                   11: FOPL
    13.  | |- ~a | b                             12: FOPL
    14.  | |- a -> b                             13: FOPL
  qed

DR15: WNPE (weak next-to-present rule)
     |- (en"a -> en"b) -> (a -> b)
     |- a -> av"(a & b)
     |- b -> av"(a & b)
   --------------------------------
     |- a -> b
  proof is similar to proof of DR15 using T3 in place of N2.

DR16: TSUBST <->
  Let c' be the result of replacing an occurrence
  of a subformula a1 in c by a2.  Then
     |- a1 <-> a2
   --------------
     |- c <-> c'
  proof:
    By induction on the structure of c.  Then for each:
      case:  c' of the form ~b, we have:
        1. | |- b <-> b'                         induction hypothesis
        2. | |- ~b <-> ~b'                       FOPL
        3. | |- c <-> c'                         2: df.
      case:  c' of the form b1 | b2, we have:
        1. | |- b1 <-> b1'                       induction hypothesis
        2. | |- b2 <-> b2'                       induction hypothesis
        3. | |- (b1 | b2) <-> (b1' | b2')        FOPL
        4. | |- c <-> c'                         3: df.
      cases:  c' of any of the FOPL forms b1 & b2, b1 -> b2,
        a"x:TYPE ( b ), etc. are similar.
      case:  c' of the form ah"b, we have:
        1. | |- b <-> b'                         induction hypothesis
        2. | |- ah"b <-> ah"b'                   1: AHIMP, FOPL, subst <->
        3. | |- c <-> c'                         2: df.
      cases:  c' of any of the the forms eh"b, av"b, ev"b, an"b and en"b,
```

```
    we proceed similarly using EHIMP, AVIMP, EVIMP, ANIMP and ENIMP,
    respectively, for AHIMP.
  case:  c' of the form b1 au" b2, we have:
    1. | |- b1 <-> b1'                           induction hypothesis
    2. | |- b2 <-> b2'                           induction hypothesis
    3. | |- b1 au" b2 <-> (b2 | (b1 & an"(b1 au" b2)))
                                                 A5
    4. | |- b1' au" b2' <-> (b2' | (b1' & an"(b1' au" b2')))
                                                 A5
    5. | |- b1' au" b2' <-> (b2 | (b1 & an"(b1' au" b2')))
                                                 1,2,4: subst <->
    6. | |- (an"(b1 au" b2) <-> an"(b1' au" b2'))
                -> ((b1 au" b2) <-> (b1' au" b2'))
                                                 3,5: subst <->
    7. | |- b1 au" b2 -> av"b2                   A6
    8. | |- b2 -> ((b1 au" b2) & (b1' au" b2'))
                                                 3,5: addition, mp, subst <->
    9. | |- av"b2 -> av"((b1 au" b2) & (b1' au" b2'))
                                                 8: AVIMP
    10.| |- b1 au" b2 -> av"((b1 au" b2) & (b1' au" b2'))
                                                 7,9: FOPL
    11.| |- b1' au" b2' -> av"b2'                A6
    12.| |- av"b2 <-> av"b2'                     2: AVIMP, FOPL
    13.| |- b1' au" b2' -> av"b2                 11,12: subst <->
    14.| |- av"b2 -> av"((b1 au" b2) & (b1' au" b2'))
                                                 8: AVIMP
    15.| |- b1' au" b2' -> av"((b1 au" b2) & (b1' au" b2'))
                                                 13,14: FOPL
    16.| |- (b1 au" b2) <-> (b1' au" b2')        6,10,15: NPA
    17.| |- c <-> c'                             3: df.
  case:  c' of the form b1 eu" b2 is similar, using E6 for A5, E7 for A6,
    and NPE for NPA.
 These are all the cases of c'.
qed
```

7.2 THEOREM SCHEMATA

```
T1:   |- ah"a -> a
  proof
    1.  | |- ah"a -> eh"a                  E4
    2.  | |- eh"a -> a                     E3
    3.  | |- ah"a -> a                     1,2: FOPL
  qed

T2:   |- ah"a -> av"a
  proof
    1.  | |- ah"a -> a                     T1
    2.  | |- eh"~a -> ~a                   E3
    3.  | |- ah"a & eh"~a -> a & ~a        1,2: FOPL
    4.  | |- ah"a -> ~eh"~a                3: FOPL
    5.  | |- ah"a -> av"a                  4,A1: FOPL, subst <->
  qed

T3:   |- an"(a -> b) -> (en"a -> en"b)
  proof
    1.  | |- (a -> b) -> (~b -> ~a)            FOPL: contraposition
    2.  | |- an"(a -> b) -> an"(~b -> ~a)      A3,N2: FOPL
    3.  | |- an"(a -> b) -> (an"~b -> an"~a)   2,N2: FOPL
    4.  | |- an"(a -> b) -> (~an"~a -> ~an"~b) 3: FOPL
    5.  | |- an"(a -> b) -> (en"a -> en"b)     4,N1: subst <->
  qed
```

```
T4:   |- ah"(a -> b) -> (av"a -> av"b)
  proof
    1.  | |- ah"((a -> b) -> (~b -> ~a))              FOPL: NEC
    2.  | |- ah"(a -> b) -> ah"(~b -> ~a)             1,A2: mp
    3.  | |- ah"(~b -> ~a) -> (eh"~b -> eh"~a)        E2
    4.  | |- (eh"~b -> eh"~a) -> (~eh"~a -> ~eh"~b)   contraposition
    5.  | |- ah"(a -> b) -> (~eh"~a -> ~eh"~b)        2,3,4: FOPL
    6.  | |- ah"(a -> b) -> (av"a -> av"b)            5,A1: subst <->
  qed

T5:   |- eh"a -> en"a
  proof
    1.  | |- eh"a -> a                     E3: simp
    1.  | |- eh"a -> en"eh"a               E3: simp
    2.  | |- en"eh"a -> en"a               1: ENIMP
    4.  | |- eh"a -> en"a                  3,E3: FOPL
  qed

T6:   |- an"a -> en"a
  proof
    1.  | |- en"(~a | a)                   FOPL, ENI
    2.  | |- en"~a | en"a                  1,T38: subst <->
    3.  | |- ~en"~a -> en"a                2: FOPL, subst <->
    4.  | |- an"a -> en"a                  3,N1: DNI, subst <->, DNE
  qed

T7:   |- ah"(a & b) <-> (ah"a & ah"b)
  proof
    1.  | |- (a & b) -> a                                  FOPL
    2.  | |- ah"(a & b) -> ah"a                            1: AHIMP
    3.  | |- (a & b) -> b                                  FOPL
    4.  | |- ah"(a & b) -> ah"b                            3: AHIMP
    5.  | |- ah"(a & b) -> (ah"a & ah"b)                   2,4: FOPL
    6.  | |- a -> (b -> (a & b))                           FOPL
    7.  | |- ah"a -> ah"(b -> (a & b))                     6: AHIMP
    8.  | |- ah"(b -> (a & b)) -> (ah"b -> ah"(a & b))     A2
    9.  | |- ah"a -> (ah"b -> ah"(a & b))                  7,8: FOPL
    10. | |- (ah"a & ah"b) -> ah"(a & b)                   9: FOPL
    11. | |- ah"(a & b) <-> (ah"a & ah"b)                  5,10: FOPL
  qed

T8:   |- eh"(a & b) <-> (eh"a & eh"b)
  proof is similar to proof of T7 using E2 in place of A2, and EHIMP in place of AHIMP

T9:   |- an"(a & b) <-> (an"a & an"b)
  proof is similar to proof of T7 using N2 in place of A2

T10:  |- en"(a & b) -> (en"a & en"b)
  proof is similar to proof of T7 lines 1-5, using ENIMP in place of AHIMP

T11:  |- an"a & en"b -> en"(a & b)
  proof
    1.  | |- a -> (b -> (a & b))                           FOPL
    2.  | |- an"a -> an"(b -> (a & b))                     1: ANIMP
    3.  | |- an"a -> (en"b -> en"(a & b))                  2,T3: FOPL
    4.  | |- an"a & en"b -> en"(a & b)                     3: FOPL
  qed

T12:  |- ah"a & eh"b -> eh"(a & b)
  proof is similar to proof of T11, using AHIMP and E2 in place of ANIMP and T3.

T13:  |- ah"a <-> a & an"ah"a
  proof
```

```
    1.  | |- ah"a -> a & an"ah"a                     A3,T1: simp, FOPL
    2.  | |- an"ah"a -> an"(a & an"ah"a)             1: ANIMP
    3.  | |- a & an"ah"a -> an"(a & an"ah"a)         2: FOPL
    4.  | |- a & ah"ah"a -> ah"(a & an"ah"a)         3: CIA
    5.  | |- ah"(a & an"ah"a) -> ah"a                T7: simp
    6.  | |- a & an"ah"a -> ah"a                     4,5: FOPL
    7.  | |- ah"a <-> a & an"ah"a                    1,6: FOPL
  qed

T14:  |- eh"a <-> a & en"eh"a
  proof
    1.  | |- eh"a -> a & eh"eh"a                     E3
    2.  | |- en"eh"a -> en"(a & eh"eh"a)             1: ENIMP
    3.  | |- a & en"eh"a -> en"(a & en"eh"a)         2: FOPL
    4.  | |- ah"(a & en"eh"a -> en"(a & en"eh"a))    3: NEC
    5.  | |- a & en"eh"a -> eh"(a & en"eh"a)         4,E5: FOPL
    6.  | |- a & en"eh"a -> eh"a                     5,T8: FOPL
    7.  | |- eh"a <-> a & en"eh"a                    1,6: FOPL
  qed

T15:  |- ah"a <-> ah"ah"a
  proof
    1.  | |- ah"a -> an"ah"a              A3: FOPL
    2.  | |- ah"(ah"a -> an"ah"a)         1: NEC
    3.  | |- ah"a -> ah"ah"a              2,A4: mp
    4.  | |- ah"ah"a -> eh"ah"a           E4
    5.  | |- eh"ah"a -> ah"a              E3: FOPL
    6.  | |- ah"ah"a -> ah"a              4,5: FOPL
    7.  | |- ah"a <-> ah"ah"a             3,6: FOPL
  qed

T16:  |- eh"a <-> eh"eh"a
  proof
    1.  | |- eh"a -> en"eh"a              E3: FOPL
    2.  | |- ah"(eh"a -> en"eh"a)         1: NEC
    3.  | |- eh"a -> eh"eh"a              2,E5: mp
    4.  | |- eh"eh"a -> eh"a              E3: FOPL
    5.  | |- eh"a <-> eh"eh"a             3,4: FOPL
  qed

T17:  |- eh"(a -> an"a) -> (a -> eh"a)
  proof
    1.  | |- eh"(a -> an"a) -> (a -> an"a) & en"eh"(a -> an"a)   E3
    2.  | |- a & eh"(a -> an"a) -> an"a & en"eh"(a -> an"a)      1: FOPL, simp
    3.  | |- a & eh"(a -> an"a) -> en"(a & eh"(a -> an"a))       2,T11: FOPL
    4.  | |- a & eh"(a -> an"a) -> eh"(a & eh"(a -> an"a))       3,E5: NEC, mp
    5.  | |- a & eh"(a -> an"a) -> eh"a                          4,T8: FOPL
    6.  | |- eh"(a -> an"a) -> (a -> eh"a)                       5: FOPL
  qed

T18:  |- av"ah"a -> ah"av"a
  proof
    1.  | |- ah"ah"a -> av"ah"a                          T2
    2.  | |- ah"a -> av"ah"a                             1,T15: subst <->
    3.  | |- an"ah"a -> an"av"ah"a                       2: ANIMP
    4.  | |- ah"a -> an"av"ah"a                          3,A3: FOPL
    5.  | |- ~eh"~~ev"~a <-> (~ev"~a | an"~eh"~~ev"~a)   T14,N1: FOPL
    6.  | |- av"ah"a -> (ah"a | an"av"ah"a)              5: FOPL
    7.  | |- av"ah"a -> an"av"ah"a                       6,4: FOPL
    8.  | |- av"ah"a -> ah"av"ah"a                       7,A4: NEC, mp
    9.  | |- ah"a -> a                                   T1
```

```
    10. | |- ah"av"ah"a -> ah"av"a                      9: AVIMP, AHIMP
    11. | |- av"ah"a -> ah"av"a                         8,10: syll
  qed

T19:  |- eh"((a | eh"b) & (eh"a | b)) <-> (eh"a | eh"b)
  proof
    1.  | |- eh"((eh"a|b) & (a|eh"b)) & ~eh"a & ~eh"b              E3: FOPL
               -> (a|eh"b) & (eh"a|b) & ~eh"a & ~eh"b
    2.  | |- eh"((eh"a|b) & (a|eh"b)) & ~eh"a & ~eh"b -> a & b     1: FOPL
    3.  | |- eh"((eh"a|b) & (a|eh"b)) & ~eh"a & ~eh"b              2,T14: subst <->, FOPL
               -> a & b & (~a | ~en"eh"a) & (~b | ~en"eh"b)
    4.  | |- eh"((eh"a|b) & (a|eh"b)) & ~eh"a & ~eh"b              3: FOPL
               -> ~en"eh"a & ~en"eh"b
    5.  | |- eh"((eh"a|b) & (a|eh"b)) & ~eh"a & ~eh"b              4,E3: FOPL
               -> en"eh"((eh"a|b) & (a|eh"b)) & ~en"eh"a & ~en"eh"b
    6.  | |- eh"((eh"a|b) & (a|eh"b)) & ~eh"a & ~eh"b              5,T11,N1: subst <->
               -> en"(eh"((eh"a|b) & (a|eh"b)) & ~eh"a & ~eh"b)
    7.  | |- eh"((eh"a|b) & (a|eh"b)) & ~eh"a & ~eh"b              6: CIE
               -> eh"(eh"((eh"a|b) & (a|eh"b)) & ~eh"a & ~eh"b)
    8.  | |- eh"((eh"a|b) & (a|eh"b)) & ~eh"a & ~eh"b              7,E3: FOPL
               -> eh"((eh"a|b) & (a|eh"b) & ~eh"a & ~eh"b)
    9.  | |- eh"((eh"a|b) & (a|eh"b)) & ~eh"a & ~eh"b              8,E3: FOPL
               -> eh"(b & a)
    10. | |- eh"((eh"a|b) & (a|eh"b)) & ~eh"a & ~eh"b              9,T8: FOPL
               -> eh"a & eh"b
    11. | |- eh"((eh"a|b) & (a|eh"b)) & ~eh"a                      10: FOPL
               -> (~eh"b -> eh"a & eh"b)
    12. | |- eh"((eh"a|b) & (a|eh"b)) & ~eh"a                      11: FOPL
               -> (eh"b | eh"a & eh"b)
    13. | |- eh"((eh"a|b) & (a|eh"b)) & ~eh"a -> eh"b              12: FOPL
    14. | |- eh"((eh"a|b) & (a|eh"b)) -> (~eh"a -> eh"b )          13: FOPL
    15. | |- eh"((eh"a|b) & (a|eh"b)) -> (eh"a | eh"b)             14: FOPL
    16. | |- eh"a -> eh"a | b                                      FOPL
    17. | |- eh"a -> a | eh"b                                      E3: simp, add
    18. | |- eh"a -> (eh"a | b) & (a | eh"b)                       16,17: FOPL
    19. | |- eh"eh"a -> eh"((eh"a | b) & (a | eh"b))               18: EHIMP
    20. | |- eh"a -> eh"((eh"a | b) & (a | eh"b))                  19,T16: subst <->
    21. | |- eh"b -> eh"((eh"a | b) & (a | eh"b))                  16-20: symmetry for 'b'
    22. | |- (eh"a | eh"b) -> eh"((eh"a | b) & (a | eh"b))         20,21: FOPL
    23. | |- eh"((eh"a | b) & (a | eh"b)) <-> (eh"a | eh"b)        15,22: FOPL
  qed

T20:  |- an"ah"a <-> ah"an"a
  proof
    1.  | |- ah"ah"a -> ah"an"a                     A3: FOPL, AHIMP
    2.  | |- ah"a -> ah"an"a                        1,T15: subst <->
    3.  | |- an"ah"a -> an"ah"an"a                  2: ANIMP
    4.  | |- an"ah"a -> an"a                        T1: ANIMP
    5.  | |- an"ah"a -> an"a & an"ah"an"a           3,4: FOPL
    6.  | |- an"ah"a -> ah"an"a                     5,T13: subst <->
    7.  | |- ah"an"a -> an"a & an"ah"an"a           T13
    8.  | |- ah"an"a -> an"(a & ah"an"a)            7,T9: subst <->, FOPL
    9.  | |- a & ah"an"a -> an"(a & ah"an"a)        8: FOPL
    10. | |- a & ah"an"a -> ah"(a & ah"an"a)        9: CIA
    11. | |- a & ah"an"a -> ah"a                    10,T7: FOPL
    12. | |- an"a & an"ah"an"a -> an"ah"a           11,T9: ANIMP, tsubst <->
    13. | |- ah"an"a -> an"ah"a                     7,12: FOPL
    14. | |- an"ah"a <-> ah"an"a                    6,13: FOPL
  qed
```

```
T21:  |- en"eh"a -> eh"en"a
  proof is similar to proof of T20 lines 1-6 using E3, T14, T16 and ENIMP in place
    of A3 (T1), T13, T15 and ANIMP.  note that the converse is not provable.

T22:    |- a -> av"a
  proof
    1.  | |- eh"~a -> ~a                    E3: FOPL
    2.  | |- a -> ~eh"~a                    1: contraposition
    3.  | |- a -> av"a                      2,A1: subst <->
  qed

T23:    |- a -> ev"a
  proof
    1.  | |- ah"~a -> ~a                    E4,E3: FOPL
    2.  | |- a -> ~ah"~a                    1: contraposition
    3.  | |- a -> ev"a                      2,E1: subst <->
  qed

T24:    |- av"a <-> av"av"a
  proof
    1.  | |- eh"~a <-> eh"eh"~a             T16: FOPL
    2.  | |- ~eh"~a <-> ~eh"eh"~a           1: subst <->
    3.  | |- av"a <-> ~eh"~~eh"~a           2,A1: tsubst <->, DNI
    4.  | |- av"a <-> av"av"a               3,A1: subst <->
  qed

T25:    |- ev"a <-> ev"ev"a
  proof
    1.  | |- ah"~a <-> ah"ah"~a             T15: FOPL
    2.  | |- ~ah"~a <-> ~ah"ah"~a           1: subst <->
    3.  | |- ev"a <-> ~ah"~~ah"~a           2,E1: tsubst <->, DNI
    4.  | |- ev"a <-> ev"ev"a               3,E1: subst <->
  qed

T26:    |- ah"(a -> b) -> (ev"a -> ev"b)
  proof
    1.  | |- (a -> b) -> (~b -> ~a)                          FOPL
    2.  | |- ah"(a -> b) -> ah"(~b -> ~a)                    1: AHIMP
    3.  | |- ah"(a -> b) -> (ah"~b -> ah"~a)                 2,A2: FOPL
    4.  | |- (ah"~b -> ah"~a) -> (~ah"~a -> ~ah"~b)          contraposition
    5.  | |- ah"(a -> b) -> (~ah"~a -> ~ah"~b)               3,4: syll.
    6.  | |- ah"(a -> b) -> (ev"a -> ev"b)                   5,E1: subst <->
  qed

T27:    |- av"(a | b) <-> (av"a | av"b)
  proof
    1.  | |- eh"(~a & ~b) <-> (eh"~a & eh"~b)                T8
    2.  | |- eh"~(a | b) <-> ~(~eh"~a | ~eh"~b)              1: tsubst <->
    3.  | |- ~av"(a | b) <-> ~(av"a | av"b)                  2,E1: dni, subst <->
    4.  | |- av"(a | b) <-> (av"a | av"b)                    3: FOPL
  qed

T28:    |- ev"(a | b) <-> (ev"a | ev"b)
  proof is similar to proof of T27 using T7 and A1 in place
  of T8 and E1, respectively.

T29:    |- av"(a & b) -> (av"a & av"b)
  proof
    1.  | |- av"(a & b) -> av"a                              FOPL, AVIMP
    2.  | |- av"(a & b) -> av"b                              FOPL, AVIMP
    3.  | |- av"(a & b) -> (av"a & av"b)                     1,2: FOPL
  qed
```

```
T30:   |- ev"(a & b) -> (ev"a & ev"b)
   proof is similar to proof of T29 using EVIMP for AVIMP.

T31:   |- (ah"a | ah"b) -> ah"(a | b)
  proof
    1.  | |- ah"a -> ah"(a | b)                       FOPL: AHIMP
    2.  | |- ah"b -> ah"(a | b)                       FOPL: AHIMP
    3.  | |- (ah"a | ah"b) -> ah"(a | b)              1,2: FOPL
  qed

T32:   |- (eh"a | eh"b) -> eh"(a | b)
  proof is similar to proof of T31 using EHIMP for AHIMP.

T33:   |- (ah"a & ev"b) -> ev"(a & b)
  proof
    1.  | |- ah"(a -> ~b) -> (ah"a -> ah"~b)          A2
    2.  | |- ah"~(a & b) -> ~(ah"a & ~ah"~b)          1: tsubst <->
    3.  | |- ~ev"(a & b) -> ~(ah"a & ev"b)            2,E1: subst <->
    4.  | |- (ah"a & ev"b) -> ev"(a & b)              3: contraposition
  qed

T34:   |- (eh"a & av"b) -> ev"(a & b)
  proof is similar to proof of T33 using E2 and A1 for A2 and E1.

T35:  |- an"a -> av"a
  proof
    1.  | |- eh"~a -> en"eh"~a          E3: FOPL
    2.  | |- en"eh"~a -> en"~a          E3: FOPL, ENIMP
    3.  | |- eh"~a -> en"~a             1,2: syll.
    4.  | |- ~en"~a -> ~eh"~a           3: contraposition
    5.  | |- an"a -> av"a               4,N1,A1: subst <->, dne, tsubst <->
  qed

T36:  |- en"a -> ev"a
  proof is similar to proof of T35 using A3, E1 and ANIMP in place
    of E3, A1 and ENIMP

T37:  |- (an"a | an"b) -> an"(a | b)
  proof
    1.  | |- an"a -> an"(a | b)                  FOPL: ANIMP
    2.  | |- an"b -> an"(a | b)                  FOPL: ANIMP
    3.  | |- (an"a | an"b) -> an"(a | b)         1,2: FOPL
  qed

T38:  |- en"(a | b) <-> (en"a | en"b)
  proof
    1.  | |- ~an"~~(~a & ~b) <-> (~an"~a | ~an"~b)      T9: FOPL, subst <->
    2.  | |- en"(a | b) <-> (en"a | en"b)               1,N1: subst <->
  qed

T39:  |- an"(a <-> b) -> (an"a <-> an"b)
  proof
    1.  | |- an"(a <-> b) <-> an"((a -> b) & (b -> a))  FOPL: ANIMP
    2.  | |- an"((a -> b) & (b -> a))                   T9
               <-> (an"(a -> b) & an"(b -> a))
    3.  | |- (an"(a -> b) & an"(b -> a))                2,T9,N2: FOPL
               -> (an"a -> an"b) & (an"b -> an"a)
    4.  | |- an"(a <-> b) -> (an"a <-> an"b)            1,2,3: FOPL
  qed

T40:  |- an"(a <-> b) -> (en"a <-> en"b)
  proof is similar to the proof of T39 using T3 in place of N2
```

```
T41:  |- av"an"a -> an"av"a
  proof
    1.  | |- ~eh"~~en"~a -> ~en"~~eh"~a            T21: FOPL
    2.  | |- av"an"a -> an"av"a                    1,N1: FOPL, tsubst <->
  qed

T42:  |- ev"en"a <-> en"ev"a
  proof is similar to proof of T41 using T20 in place of T21

T43:  |- av"an"a -> av"a
  proof
    1.  | |- av"an"a -> av"av"a         T35: AVIMP
    2.  | |- av"an"a -> av"a            1,T24: tsubst <->
  qed

T44:  |- ev"en"a -> ev"a
  proof
    1.  | |- ev"en"a -> ev"ev"a         T36: EVIMP
    2.  | |- ev"en"a -> ev"a            1,T25: tsubst <->
  qed

T45:  |- av"a <-> (a | an"av"a)
  proof
    1.  | |- ~eh"~a <-> ~(~a & en"eh"~a)           T14: FOPL
    2.  | |- av"a <-> (a | an"av"a)                1,N1: FOPL, tsubst <->
  qed

T46:  |- ev"a <-> (a | en"ev"a)
  proof is similar to proof of T45 using T13 in place of T14

T47:  |- (a & av"~a) -> ev"(a & an"~a)
  proof
    1.  | |- ah"(a -> en"a) -> (a -> eh"a)            E5
    2.  | |- ~ev"~(a -> ~an"~a) -> (a -> ~av"~a)      1,E1,N1: FOPL, tsubst <->
    3.  | |- ~ev"(a & an"~a) -> ~(a & av"~a)          2: FOPL
    4.  | |- (a & av"~a) -> ev"(a & an"~a)            3: contraposition
  qed

T48:  |- (a & ev"~a) -> ev"(a & en"~a)
  proof is similar to proof of T47 using A4 and A1 in place of E5 and E1

T49:  |- a & (~a au" b) -> b
  proof
    1.  | | a & (~a au" b) & ~b        assume
    2.  | | a                          1: simp
    3.  | | ~a au" b                   1: simp
    4.  | | ~b                         1: simp
    5.  | | b | ~a & an"(~a au" b)     3,A5: subst <->
    6.  | | ~a & an"(~a au" b)         4,5: ds
    7.  | | ~a                         6: simp
    8.  | a & (~a au" b) -> b          1-7: ind. proof
  qed

T50:  |- ~b & (a au" b) -> a
  proof
    1.  | | ~b & (a au" b) & ~a        assume
    2.  | | ~b                         1: simp
    3.  | | a au" b                    1: simp
    4.  | | ~a                         1: simp
    5.  | | b                          3,4,T49: FOPL
    6.  | ~b & (a au" b) -> a          1-5: ind. proof
  qed
```

```
T51:  |- b -> (a au" b)
  proof
    1.  | | b & ~(a au" b)                  assume
    2.  | | b                               1: simp
    3.  | | b | a & an"(a au" b)            2: add
    4.  | | ~(b | a & an"(a au" b))         1,A5: simp, subst <->
    5.  | b -> (a au" b)                    1-4: ip
  qed
```

```
T52:  |- a & an"(a au" b) -> (a au" b)
  proof
    1.  | | a & an"(a au" b) & ~(a au" b)        assume
    2.  | | a & an"(a au" b)                     1: simp
    3.  | | b | a & an"(a au" b)                 2: add
    4.  | | ~(b | a & an"(a au" b))              4,A5: subst <->
    5.  | a & an"(a au" b) -> (a au" b)          1-4: ip
  qed
```

```
T53:  |- av"(a au" b) <-> (av"a au" av"b)
  proof
    1.  | | av"(a au" b)                          assume
    2.  | | av"av"b                               A6,1: AVIMP, mp
    3.  | | av"b                                  2,T24: subst <->
    4.  | | (av"a au" av"b)                       3,T51: mp
    5.  | av"(a au" b) -> (av"a au" av"b)         1-4: CP
    6.  | | (av"a au" av"b)                       assume
    7.  | | av"av"b                               A6,6: mp
    8.  | | av"b                                  7,T24: subst <->
    9.  | | av"(a au" b)                          T51,8: AVIMP, mp
    10. | (av"a au" av"b) -> av"(a au" b)         6-9: CP
    11. | av"(a au" b) <-> (av"a au" av"b)        5,10: FOPL
  qed
```

```
T54:  |- ev"(a eu" b) <-> (ev"a eu" ev"b)
  proof
    1.  | | ev"(a au" b)                           assume
    2.  | | ev"av"b                                E7,1: EVIMP, mp
    3.  | | av"b -> ev"b                           E3,A1,E1: FOPL
    4.  | | ev"av"b -> ev"ev"b                     3: EVIMP
    5.  | | ev"b                                   2,4,T25: mp, subst <->
    6.  | | ev"b | ev"a & en"(ev"a eu" ev"b)       5: addition
    7.  | | (ev"a au" ev"b)                        6,E6: subst <->
    8.  | ev"(a au" b) -> (ev"a au" ev"b)          1-7: CP
    9.  | | (ev"a au" ev"b)                        assume
    10. | | av"ev"b                                A6,9: mp
    11. | | av"ev"b -> ev"ev"b                     E3,A1,E1: FOPL
    12. | | ev"b                                   10,11,T25: mp, subst <->
    13. | | ev"(a au" b)                           T51,12: EVIMP, mp
    14. | (ev"a au" ev"b) -> ev"(a au" b)          9-13: CP
    15. | ev"(a au" b) <-> (ev"a au" ev"b)         8,14: FOPL
  qed
```

```
T55:   |- (a au" b) -> (a eu" b)
  proof
    1.  | | | an"(a au" b) -> an"(a eu" b)     assume
    2.  | | | | (a au" b) & ~(a eu" b)         assume
    3.  | | | | b | a & an"(a au" b)          2,A5: simp, subst <->
    4.  | | | | b | a & an"(a eu" b)          3,1: FOPL
    5.  | | | | b | a & en"(a eu" b)          4,T6: FOPL
    6.  | | | | a eu" b                        5,E6: subst <->
    7.  | | | (a au" b) -> (a eu" b)          2-6: cp
```

```
   8.  | |- (an"(a au" b) -> an"(a eu" b))          1-8: cp
              -> ((a au" b) -> (a eu" b))
   9.  | |- (a au" b) -> av"b                        A6
  10.  | |- (a eu" b) -> av"b                        E7
  11.  | |- b -> ( b | a & an"(a au" b) )            FOPL
              & ( b | a & en"(a eu" b) )
  12.  | |- av"b                                     11: AVIMP
             -> av"((b|a&an"(a au" b)) & (b|a&en"(a eu" b)))
  13.  | |- (a au" b)                                9,12,A5,E6: FOPL
             -> av"((a au" b) & (a eu" b))
  14.  | |- (a eu" b)                                10,12,A5,E6: FOPL
             -> av"((a au" b) & (a eu" b))
  15.  | |- (a au" b) -> (a eu" b)                   8,13,14: WNPA
 qed

T56:   |- an"a au" an"b -> an"(a au" b)
 proof
   1.  | |- an"a au" an"b                                 A5: FOPL
             -> an"b | an"a & an"(an"a au" an"b)
   2.  | |- an"(a au" b)                                  A5: ANIMP
             <-> an"(b | a & an"(a au" b))
   3.  | |- an"b | an"a & an"an"(a au" b)                 T9,T37: tsubst <->, FOPL
             -> an"(b | a & an"(a au" b))
   4.  | |- an"b | an"a & an"an"(a au" b)                 2,3: subst <->
             -> an"(a au" b)
   5.  | |- (an"(an"a au" an"b) -> an"an"(a au" b))       1: FOPL
             -> (an"a au" an"b) -> an"b | an"a & an"an"(a au" b)
   6.  | |- (an"(an"a au" an"b) -> an"an"(a au" b))       4,5: syll, FOPL
             -> (an"a au" an"b) -> an"(a au" b)
   7.  | |- an"b -> an"(a au" b)                          T51: ANIMP
   8.  | |- an"b -> (an"a au" an"b)                       T51
   9.  | |- an"b -> (an"(a au" b) & (an"a au" an"b))      7,8: FOPL
  10.  | |- av"an"b                                       9: AVIMP
             -> av"(an"(a au" b) & (an"a au" an"b))
  11.  | |- an"a au" an"b -> av"an"b                      A6
  12.  | |- an"a au" an"b                                 10,11: syll
             -> av"(an"(a au" b) & (an"a au" an"b))
  13.  | |- a au" b -> av"b                               A6
  14.  | |- an"(a au" b) -> an"av"b                       13: ANIMP
  15.  | |- an"(a au" b) -> av"an"b                       14,T41: subst <->
  16.  | |- an"(a au" b)                                  10,15: syll
             -> av"(an"(a au" b) & (an"a au" an"b))
  17.  | |- an"(a au" b) -> (an"a au" an"b)               5,12,16: WNPE
 qed

T57:  |- a & ~b -> a ap" b
 proof
   1.  | | a & ~b & ~(a ap" b)                   assume
   2.  | | ~a au" b                              1: simp, AP, subst <->, dne
   3.  | | b                                     1,2,T49: simp
   4.  | | ~b                                    1: simp
   5.  | a & ~b -> a ap" b                       1-4: ip
 qed

T58:  |- (a ap" b) -> ~b
 proof
   1.  | | (a ap" b) & b                         assume
   2.  | | ~(~a au" b)                           1: simp, AP, subst <->
   3.  | | b | ~a & an"(~a au" b)                1: simp, addition
   4.  | | ~(b | ~a & an"(~a au" b))             2,A5: subst <->
   5.  | (a ap" b) -> ~b                         1-4: ip
```

```
  qed

T59:   |- (a|b|c)&(a ap" b)&(b ap" c) -> (a ap" c)
  proof
    1.  | | (a|b|c) & (a ap" b) & (b ap" c) & ~(a ap" c)   assume
    2.  | | ~(~a au" b) & ~(~b au" c) & (~a au" c)          1: AP, FOPL
    3.  | | ~(b | a & an"(~a au" b))                        2,A5: simp, FOPL
    4.  | | ~(c | b & an"(~b au" c))                        2,A5: simp, FOPL
    5.  | | (c | ~a & an"(~a au" c))                        2,A5: simp, FOPL
    6.  | | ~b                                              3: FOPL
    7.  | | a | ~an"(~a au" b)                              3: FOPL
    8.  | | ~c                                              4: FOPL
    9.  | | b | ~an"(~b au" c)                              4: FOPL
    10. | | ~a & an"(~a au" c)                              5,8: ds
    11. | | ~a                                              10: simp
    12. | | b | c                                           1,11: simp, ds
    13. | | c                                               6,12: ds
    14. | (a|b|c) & (a ap" b) & (b ap" c) -> (a ap" c)      1-13: ip, FOPL
  qed

T60:   |- (a ap" b) -> (b -> c)
  proof
    1.  | | (a ap" b)                                       assume
    2.  | | | b & ~c                                        assume
    3.  | | | b                                             2: simp
    4.  | | | ~b                                            1,T58: FOPL
    5.  | | b -> c                                          2-4: ip, FOPL
    6.  | (a ap" b) -> (b -> c)                             1-5: cp
  qed

T61:   |- a -> (a ab" b)
  proof
    1.  | | a                                               assume
    2.  | | (~b au" a)                                      1,T51: mp
    3.  | | av"b -> (~b au" a)                              2: FOPL
    4.  | | a ab" b                                         3,AB: subst <->
    5.  | a -> (a ab" b)                                    1-4: cp
  qed

T62:   |- (a|b|c) & (a ab" b) & (b ab" c) -> (a ab" c)
  proof
    1.  | | (a|b|c) & (a ab" b) & (b ab" c) & ~(a ab" c)   assume
    2.  | | a | b | c                                       1,AB: simp
    3.  | | av"b -> (~b au" a)                              1,AB: simp
    4.  | | av"c -> (~c au" b)                              1,AB: simp
    5.  | | av"c & ~(~c au" a)                              1,AB: simp, FOPL
    6.  | | av"c                                            5: simp
    7.  | | ~(a | ~c & an"(~c au" a))                       5,A5: simp, FOPL
    8.  | | ~a & (c | ~an"(~c au" a))                       7: FOPL
    9.  | | ~a                                              8: simp
    10. | | b | c                                           2,9: ds
    11. | | ~c au" b                                        4,6: mp
    12. | | b | ~c & an"(~c au" b)                          11,A5: subst <->
    13. | | b | ~c                                          12: FOPL
    14. | | b                                               10,13: FOPL
    15. | | av"b                                            14,T22: mp
    16. | | ~b au" a                                        3,15: mp
    17. | | a | ~b & an"(~b au" a)                          16,A5: subst <->
    18. | | ~b & an"(~b au" a)                              9,17: ds
    19. | | ~b                                              18: simp
    20. | (a|b|c) & (a ab" b) & (b ab" c) -> (a ab" c)      1-19: ind proof, FOPL
```

```
  qed

T63:   |- (a -> d) & (a ab" b) & (b ab" c) -> (c -> d)
  proof
    1.  | | (a -> d) & (a ab" b) & (b ab" c)              assume
    2.  | | | c                                           assume
    3.  | | | av"c                                        2,T22: simp, mp
    4.  | | | av"c -> (~c au" b)                          1,AB: simp, subst <->
    5.  | | | b | ~c & an"(~c au" b)                      3,4,A5: mp, subst <->
    6.  | | | c | ~an"(~c au" b)                          5: add
    7.  | | | ~(~c & an"(~c au" b))                       6: FOPL
    8.  | | | b                                           5,7: ds
    9.  | | | av"b                                        8,T22: mp
    10. | | | av"b -> (~b au" a)                          1,AB: simp, subst <->
    11. | | | a | ~b & an"(~b au" a)                      9,10,A5: mp, subst <->
    12. | | | b | ~an"(~b au" a)                          8: add
    13. | | | ~(~b & an"(~b au" a))                       12: FOPL
    14. | | | a                                           11,13: ds
    15. | | | d                                           1,14: simp, mp
    16. | | c -> d                                        2-15: cp
    17. | (a -> d) & (a ab" b) & (b ab" c) -> (c -> d)    1-16: cp
  qed
```

8. BIBLIOGRAPHY

[AFFIRM81] "AFFIRM Reference Manual", D.H. Thompson and R.W. Erickson, Eds., USC Information Sciences Institute, 1981.

[BP80] Ben-Ari, Mordechai and Amir Pnueli, "The Logic of Nexttime", Technical Report 80-13, Tel Aviv University, Tel Aviv, Israel, July 1980.

[HC68] Hughes, G.E., and M.J. Cresswell, *An Introduction to Modal Logic*, London, Methuen (1968).

[Ing78] Ingalls, D.H.H., "The Smalltalk-76 Programming System Design and Implementation, *Proceedings of the Fifth ACM Symposium on Principles of Programming Languages*, January, 1978.

[Kr63] Kripke, S.A. "Semantical Considerations on Modal Logics". *Acta Philosophica Fennica*, (1963) *Modal and Many-valued Logics*, pp. 83-94.

[LZ74] Liskov, B.H. and Zilles, S.N., "Programming with Abstract Data Types", *SIGPLAN Notices 9:5* (1974).

[Man81] Manna, Zohar "Verification of Sequential Programs: Temporal Axiomatization", Technical Report No. STAN-CS-81-877, Department of Computer Science, Stanford University, Stanford, CA, September 1981.

[MP81a] Manna, Zohar and Amir Pnueli, "Verification of Concurrent Programs: Part I: The Temporal Framework", Technical Report No. STAN-CS-81-836, Department of Computer Science, Stanford University, Stanford, CA, July 1981.

[MP81b] Manna, Zohar and Amir Pnueli, "Verification of Concurrent Programs: Part II: Temporal Proof Principles", Technical Report No. STAN-CS-81-843, Department of Computer Science, Stanford University, Stanford, CA, September 1981.

[Mye78] Myers, G.H., *Composite/Structured Design*, New York, van Nostrand Reinhold (1978).

[Par72] Parnas, D.L., "On the Criteria to be Used in Decomposing Systems into Modules", *CACM* 15 (1972).

[Par76] Parnas, D.L. "Some Hypotheses About the 'Uses' Hierarchy for Operating Systems", Technical Report, Technische Hochschule Darmstadt, Darmstadt, West Germany, March 1976.

[SA85] Scheid, J. and S. Anderson *The Ina Jo Specification Language Reference Manual*, TM-(L)-6021/001/01, System Development Corporation, Santa Monica, March 1985.

[Win83] Wing, J.M., "A Two-Tiered Approach to Specifying Programs", Ph.D Thesis, MITLaboratory for Computer Science, 1983.

[Wir77] Wirth, N., "Modula: A Language for Modular Multiprogramming", *Software — Practice and Experience* 7,3-35 (1977).

System Development
Corporation
A Burroughs Company

On Adding Concurrency to the Formal Development Methodology (FDM)

SP-4360

March 1986