The Temporal Logic of Actions I

Wolfgang Schreiner
Research Institute for Symbolic Computation (RISC-Linz)
Johannes Kepler University, A-4040 Linz, Austria

Wolfgang.Schreiner@risc.uni-linz.ac.at
http://www.risc.uni-linz.ac.at/people/schreine

Introduction

- Concurrent algorithm typically described by a program.
  - *Correctness* of algorithm means program satisfies desired property.
- TLA = Temporal Logic of Actions.
  - Lamport, 1994.
  - Both algorithm and property are specified by formulas in single logic.
  - Correctness of algorithm means algorithm *implies* property.
- Reasonable to abandon programing language?
  - Mostly reasoning about concurrent *algorithms*.
  - Concurrent *programs* are much to complicated.
  - 1 page algorithm = 5000 lines of C code.
  - Goal to detect *algorithmic* errors.

Talking about concurrent algorithms!

Logic Versus Programming

- Aren't programs simpler than logic formulas?
  - Everyday mathematics simpler than programs.
  - Assignment versus equality!
  - *Program* versus *mathematical* functions.
- Methods for reasoning about programs based on toy languages.
  - Simpler than real programming languages.
  - More complicated than simple logic.
- Resemblance often misleasing:
  - $\{ x = 0 \}$ y:=x+1 $\{ y = x + 1 \}$
  - x:=0; y:=x+1; write(y,x+1)
  - May be different in certain contexts (aliasing)!

Real languages contain difficult concepts because they must yield reasonably efficient programs for complex computers.

Goals of a Programming Logic

Reasoning about concurrent algorithms.

- Simpler alternative to programming languages.
  - No point in trading language for a complicated logic.
- Expressive to describe real algorithms.
  - Formulas must not be too long and complicated to understand.
- TLA formulas:
  - Familiar mathematical operators ($\wedge$).
  - ' (prime), $\Box$, $\exists$.
- Combination of two logics:
  - A logic of actions.
  - A standard temporal logic.

The Logic of Actions

Values, variables and state.

- Collection **Val** of *values*.
  - Algorithms manipulate data.
  - Numbers, strings, sets.
- Infinite set **Var** of *variable* names.
  - Algorithms assign values to variables.
- A *state* assigns values to variables.
  - $s \in \textbf{St} = \textbf{Var} \to \textbf{Val}$.
  - $s[[x]] := s(x)$.
  - $[[x]] \in St \to Val$.
  - Semantic meaning $[[x]]$ of syntactic object x.

State Functions and Predicates

- *State function*
  - Expression built from variables and constant symbols.
  - $s[[x^2 + y - 3]] = (s[[x]])^2$+$s[[y]]$-3.
  - $s[[f]] := f(\forall$ 'v': $s[[v]]/v)$
  - Correspond to program expressions (and subexpressions of assertions).
- *State predicate*
  - Boolean expression.
  - $x^2 = y - 3$
  - $s[[P]] \in \{$**true**, **false**$\}$.
  - s *satisfies* P iff $s[[P]] =$ **true**.
  - Correspond to assertions (and boolean-valued program expressions).

Actions

- *Action* = boolean-valued expression.
  - Variables, primed variables, constant symbols.
  - $x' + y = y$, $x - 1 \in z'$.
- Relation between *old* state and *new* state.
  - Unprimed variables refer to old state.
  - Primed variables refer to new state.
  - Representation of atomic operation of concurrent program.
- Formalization of A
  - [[A]] $\in$ **St** $\rightarrow$ **S** $\rightarrow$ **Bool**
  - s[[A]]$t \in$ **Bool**.
  - Old state s, new state t.
  - s[[A]]$t \equiv$ A($\forall$ 'v: $s[[v]]/v$, $t[[v]]/v'$).
  - $s[[y = x' + 1]]t = (s[[y]] = t[[x]]+1)$.
  - s, t is an A *step* iff s[[A]]t = **true**.

Predicates as Actions

- $s[[P]]$ is boolean for any s.
- View P as action without primed variables.
  - $s[[P]]t = s[[P]]$ for any s, t.
  - s, t is a P *step* iff s satisfies P.
- Replacement of unprimed variables:
  - State function or predicate F.
  - $F' := F(\forall$ 'v': $v'/v)$.
  - $s[[P']]t = t[[P]]$

Validity and Provability

- Action A is *valid* ($\models$ A)
  - Every step is an A step.
  - $\models$ A $\equiv \forall s, t \in$ **St**: s[[A]]t
  - $\models P \equiv \forall s \in$ **St**: $s[[P]]$
  - True regardless of what values are substituted for primed and unprimed variables.
  - $(x' + y \in \textbf{Nat}) \Rightarrow (2(x' + y) \geq x' + y)$
- Formula F is provable ($\vdash F$)
  - Formal derivation by rules of logic.
- *Soundness* of the logic.
  - Every provable formula is valid.
  - $\vdash F \Rightarrow \models F$.

Rigid Variables and Quantifiers

- Program described using parameter *n*.
  - Mathematician: *variable* (symbol does not represent known value).
  - Programmer: *constant* (value of *n* does not change).
- Two kinds of *variables*:
  - *Rigid variables* (unknown constant).
  - *(Flexible) variables* (program variable).
- *Constant expressions*:
  - Built from rigid variables and constant symbols.
  - Extend state functions and actions to contain constant expressions.
- *Quantification* over rigid variables
  - $s[[\exists\ m \in \textbf{Nat}:\ m x' = n + x]] \equiv$
    $\exists\ m \in \textbf{Nat}:\ m(t[[x]]) = n + s[[x]]$
  - A is valid if $s[[A]]t$ equals **true** for all states s, t and all possible values of its free rigid variables.

The Enabled Predicate

- *Enabled* A
  - True for s iff it is possible to take an A step starting in s.
  - s[[*Enabled* A]] $\equiv \exists\, t \in$ **St**: s[[A]]t
- Syntactic definition
  - v_i all (flexible) variables in A.
  - *Enabled* A $\equiv$
    $\exists\, c_1, \ldots, c_n$: A$(c_1/v_1', \ldots, c_n/v_n')$.
  - *Enabled*$(y = (x')^2 + n) =$
    $\exists\, c$: $y = c^2 + n$
- If A represents actomic operation, *Enabled* A is true for those states in which it is possible to perform the operation.

Simple Temporal Logic

Execution of algorithm

- Sequence of steps.
- Each step produces new state changing the values of variables.
- Execution is sequence of states.
- Semantic meaning of algorithm is collection of all possible executions.

Temporal logic allows reasoning about sequences of states.

Temporal Formulas

- *Always* ($\Box$)
  - Elementary formulas E_1, E_2
  - $\neg E_1 \land \Box(\neg E_2)$
  - $\Box(E_1 \Rightarrow \Box(E_1 \lor E_2))$
- Semantics based on *behaviors*
  - Infinite sequences of states.
  - Behavior $\sigma = \langle s_0, s_1, \ldots \rangle$
  - $\sigma[[F]] \in$ **Bool**.
  - σ *satisfies* F iff $\sigma[[F]] =$ **true**.
- Meaning of temporal formulas:
  - $\langle s_0, s_1, \ldots \rangle[[F]] \equiv s_0[[F]]$, if F elementary.
  - $\sigma[[F \land G]] \equiv \sigma[[F]] \land \sigma[[G]]$
  - $\sigma[[\neg F]] \equiv \neg\sigma[[F]]$
  - $\langle s_0, s_1, \ldots \rangle[[\Box F]] \equiv$
    $\forall\ n \in$ **Nat**: $\langle s_n, s_{n+1}, \ldots \rangle[[F]]$

Some Useful Temporal Formulas

- *Eventually* ($\Diamond$)
  - F is eventually true.
  - $\Diamond F \equiv \neg\Box\neg F$.
  - $\langle s_0, s_1, \ldots\rangle[[\Diamond F]] \equiv$
    $\exists\, n \in \textbf{Nat}\text{: } \langle s_n, s_{n+1}, \ldots\rangle[[F]]$
- *Infinitely Often* ($\Box\Diamond$)
  - $\langle s_0, s_1, \ldots\rangle[[\Box\Diamond F]] \equiv$
    $\forall n \in \textbf{Nat}\text{: } \exists\, m \in \textbf{Nat}\text{:}$
    $\langle s_{n+m}, s_{n+m+1}, \ldots\rangle[[F]]$
- *Eventually Always* ($\Diamond\Box$)
  - $\langle s_0, s_1, \ldots\rangle[[\Diamond\Box F]] \equiv$
    $\exists n \in \textbf{Nat}\text{: } \forall\, m \in \textbf{Nat}\text{:}$
    $\langle s_{n+m}, s_{n+m+1}, \ldots\rangle[[F]]$
- *Leads to* ($\leadsto$)
  - $F \leadsto G \equiv \Box(F \Rightarrow \Diamond G)$
  - Any time F is true, G is true then or at some later time.

Validity and Provability

- *Validity* of F ($\models F$)
  - $\models F \equiv \forall \sigma \in \mathbf{St}^{\infty}$: $\sigma[[F]]$
  - $^{\infty}$ set of all possible behaviors.
- Representation of algorithm:
  - Temporal formula F:
  - $\sigma[[F]] =$ **true** iff σ represents a possible execution of the algorithm.
- Property G of algorithm:
  - $\models F \Rightarrow G$.
  - Algorithm represented by F satisfies property G.
- Rules will be introduced for proving temporal formulas.
  - Soundness: $\vdash F \Rightarrow \models F$.

The Raw Logic

Raw Temporal Logic of Actions (RTLA)

- Elementary temporal formulas are actions.
- Action A is true on behavior σ:
  - $\langle s_0, s_1, \ldots \rangle[[\mathsf{A}]] \equiv s_0[[\mathsf{A}]]s_1$
  - First pair s_0, s_1 of behaviors is an A step.
- Temporal operator:
  - $\langle s_0, s_1, \ldots \rangle[[\Box \mathsf{A}]]$
    $\equiv \forall n \in \textbf{Nat}\colon \langle s_n, s_{n+1}, \ldots \rangle[[\mathsf{A}]]$
    $\equiv \forall n \in \textbf{Nat}\colon s_n[[\mathsf{A}]]s_{n+1}$.
- Predicates:
  - $\langle s_0, s_1, \ldots \rangle[[P]] \equiv s_0[[P]]$
  - $\langle s_0, s_1, \ldots \rangle[[\Box P]] \equiv \forall n \in \textbf{Nat}\colon s_n[[P]]$

TLA formulas will be subset of RTLA formulas.

Describing Programs with RTLA

- Program in guarded command language.
  - **var natural** x, $y = 0$
    do
    $\langle$**true** $\rightarrow x := x + 1\rangle$
    []
    $\langle$**true** $\rightarrow y := y + 1\rangle$
    od
- Formula Φ
  - $\mathit{Init}_\Phi \equiv (x = 0) \wedge (y = 0)$
  - $\mathsf{M}_1 \equiv (x' = x + 1) \wedge (y' = y)$
  - $\mathsf{M}_2 \equiv (y' = y + 1) \wedge (x' = x)$
  - $\mathsf{M} \equiv \mathsf{M}_1 \vee \mathsf{M}_2$
  - $\Phi \equiv \mathit{Init}_\Phi \wedge \Box\mathsf{M}$
- $\sigma[[\Phi]] =$ **true** iff σ represents possible execution of program.

Describing Programs with RTLA

- Init_Φ asserts initial condition.
- Action M_1 asserts effect of first guarded command.
- Action M_2 asserts effect of second guarded command.
- Action M asserts effect of non-deterministic composition.
- Formular Φ represents whole program:
  - Init_Φ is true in first state.
  - Every step is an M step.

Each equivalent formula is a valid representation of the program.

TLA

- Formula Φ is too simple.
  - Should allow *stuttering steps*
  - Leave both x and y unchanged.
- Example: clock specification.
  - Clock C_1 with hours h and minutes m.
  - Clock C_2 with hours h, minutes m, seconds s.
  - C_2 should statisfy specification of C_1.
  - But C_2 has 59 steps where h and m do not change!
  - Such stuttering steps should be ignored.
- Modification of Φ:
  - $\Phi \equiv \mathit{Init}_\Phi \wedge \Box(\mathsf{M} \vee ((x' = x) \wedge (y' = y)))$
  - $\Phi \equiv \mathit{Init}_\Phi \wedge \Box\mathsf{M}_{\langle x,y \rangle}$
  - $[\mathsf{A}]_f := \mathsf{A} \vee (f' = f)$
- TLA is subset of RTLA
  - Elementary formulas of form $\Box[\mathsf{A}]_f$

Adding Liveness

- Modified Φ also not acceptable:
  - x, y might be never changed!
  - Φ only expresses *safety* property.
  - Program must not execute other than described steps.
- *Liveness* properties:
  - Something *does* eventually happen.
  - Program must eventually perform described steps.
- Dijkstra semantics:
  - Infinitely many steps increase x *or* y.
  - $\Phi \equiv \mathit{Init}_\Phi \land \Box \mathsf{M}_{\langle x,y \rangle} \land \Box\Diamond \mathsf{M}$.
- Add *fairness* requirement:
  - Infinitely many steps increase x *and* y.
  - $\Phi \equiv \mathit{Init}_\Phi \land \Box \mathsf{M}_{\langle x,y \rangle} \land \Box\Diamond \mathsf{M}_1 \land \Box\Diamond \mathsf{M}_2$.

Problem: Both are not TLA formulas!

Liveness as TLA Formulas

- $\Box[\mathsf{A}]_f$ is TLA formula.
  - $\neg\Box[\neg\mathsf{A}]_f$
    $\equiv\Diamond\neg[\neg\mathsf{A}]_f$
    $\equiv\Diamond\neg(\neg\mathsf{A} \vee f' = f)$
    $\equiv\Diamond(\mathsf{A} \wedge f' \neq f)$
  - $\langle\mathsf{A}\rangle_f \equiv \mathsf{A} \wedge f' \neq f$
  - $\langle\mathsf{A}\rangle_f$ is TLA formula.
- Reformulation of Φ:
  - $\Phi \equiv \mathit{Init}_\Phi \wedge \Box\mathsf{M}_{\langle x,y\rangle}$
    $\wedge\ \Box\Diamond\langle\mathsf{M}_1\rangle_{\langle x,y\rangle} \wedge \Box\Diamond\langle\mathsf{M}_2\rangle_{\langle x,y\rangle}$

Fairness

- Arbitrary liveness properties dangerous:
  - Used to express fairness requirements.
  - May unexpectedly add *safety* properties.
  - Add: $\Box\Diamond(x = 0)$.
  - Consequence: x never changes!
  - Solution: express liveness by fairness.
- Fairness:
  - If operation possible, program must eventually execute it.
- *Weak* fairness:
  - Operation must be executed if it remains possible to do so *for long enough time*.
  - ($\Diamond$ executed) $\vee$ ($\Diamond$ impossible)
- *Strong* fairness:
  - Operation must be executed if it is *often enough* possible to do so.
  - ($\Diamond$ executed) $\vee$ ($\Diamond\Box$ impossible)

Fairness

- Fairness at all times:
  - $\Box((\Diamond \text{ executed}) \vee (\Diamond \text{ impossible}))$
  - $\Box((\Diamond \text{ executed}) \vee (\Diamond\Box \text{ impossible}))$
- Equivalent to:
  - $(\Box\Diamond \text{ executed}) \vee (\Box\Diamond \text{ impossible})$
  - $(\Box\Diamond \text{ executed}) \vee (\Diamond\Box \text{ impossible}))$
- Formalization:
  - $\mathit{executed} \equiv \langle A \rangle_f$.
  - $\mathit{impossible} \equiv \neg \mathit{Enabled}\ \langle A \rangle_f$.
- Fairness conditions:
  - $WF_f(A) \equiv (\Box\Diamond\langle A \rangle_f) \vee (\Box\Diamond\ \neg \mathit{Enabled}\ \langle A \rangle_f)$
  - $SF_f(A) \equiv (\Box\Diamond\langle A \rangle_f) \vee (\Diamond\Box\ \neg \mathit{Enabled}\ \langle A \rangle_f)$
  - $SF_f(A) \Rightarrow WF_f(A)$

Rewriting the Fairness Requirement

- *Machine-closed*
  - Pair $(\mathit{Init} \land \Box[\mathsf{N}]_f, F)$ is *machine-closed* $\equiv \mathit{Init} \land \Box[\mathsf{N}]_f \land F$ does not add additional safety properties.
  - If F is conjunciton of conditions $\mathsf{WF}_f(\mathsf{A})$ and/or $\mathsf{SF}_f(\mathsf{A})$, where each $\langle \mathsf{A} \rangle_f$ implies N, then $\mathit{Init} \land \Box[\mathsf{N}]_v \land F$ is machine-closed.
- Fairness requirements:
  - Rewrite $\Box\Diamond\langle \mathsf{M}_1 \rangle_{\langle x,y \rangle} \land \Box\Diamond\langle \mathsf{M}_2 \rangle_{\langle x,y \rangle}$ as fairness conditons.
  - $\mathit{Enabled}\ \langle \mathsf{M}_1 \rangle_{\langle x,y \rangle} = \mathit{Enabled}\ \langle \mathsf{M}_2 \rangle_{\langle x,y \rangle} = \mathbf{true}$
  - $\mathsf{WF}_{\langle x,y \rangle}(\mathsf{M}_1) = \Box\Diamond\langle \mathsf{M}_1 \rangle_{\langle x,y \rangle}$
    $\mathsf{WF}_{\langle x,y \rangle}(\mathsf{M}_2) = \Box\Diamond\langle \mathsf{M}_2 \rangle_{\langle x,y \rangle}$
  - $\Phi \equiv \mathit{Init}_\Phi \land \Box\mathsf{M}_{\langle x,y \rangle}$
    $\land \mathsf{WF}_{\langle x,y \rangle}(\mathsf{M}_1) \land \mathsf{WF}_{\langle x,y \rangle}(\mathsf{M}_2)$

Examining Formula Φ

- Each TLA formula representing program:
  - *Init* $\land \Box[\mathrm{N}]_f \land$ F
  - *Init* specifies initial variable values.
  - N is the program's *next-state relation* that represents the execution of the inidividual atomic operations.
  - f is the n tuple of all flexible variables.
  - F is a conjunciton of formulas of the form $\mathrm{WF}_f(\mathrm{A})$ and/or $\mathrm{SF}_f(\mathrm{A})$ wher A represents a subset of the program's atomic operations.
- Parallel composition:
  - Two programs represented by Φ and Ψ.
  - No variables in common
  - $\Phi \land \Psi$ describes parallel composition of both programs!

Syntax of Simple TLA

TLA logic without quantification.

- $\langle \mathit{formula} \rangle \equiv \langle \mathit{predicate} \rangle$
  $\| \; \Box[\langle \mathit{action} \rangle]_{\langle \mathit{state\ function} \rangle} \; \| \; \neg\langle \mathit{formula} \rangle$
  $\| \; \langle \mathit{formula} \rangle \wedge \langle \mathit{formula} \rangle \; \| \; \Box\langle \mathit{formula} \rangle$
- $\langle \mathit{action} \rangle \equiv$ boolean-valued expression
  of constant symbols,
  variables, and primed variables
- $\langle \mathit{predicate} \rangle \equiv$ *action* with no primed variables
  $\|$ *Enabled* $\langle \mathit{action} \rangle$
- $\langle \mathit{state\ function} \rangle \equiv$ non-boolean expression
  containing constant symobls and variables

Semantics of Simple TLA

- $s[[f]] \equiv f(\forall\ 'v': s[[v]]/v)$
- $s[[\mathsf{A}]]t \equiv \mathsf{A}(\forall\ 'v': s[[v]]/v,\ t[[v]]/v')$
- $\langle s_0,\ s_1,\ \ldots\rangle[[\mathsf{A}]] \equiv s_0[[\mathsf{A}]]s_1$
- $\models \mathsf{A} \equiv \forall\ s, t \in \textbf{St}:\ s[[\mathsf{A}]]t$
- $s[[\textit{Enabled}\ \mathsf{A}]] \equiv \exists t \in \textbf{St}:\ s[[\mathsf{A}]]t$
- $\langle s_0,\ s_1,\ \ldots\rangle[[\Box F]] \equiv$
  $\forall\ n \in \textbf{Nat}:\ \langle s_n,\ s_{n+1},\ \ldots\rangle[[F]]$
- $\sigma[[F \wedge G]] \equiv \sigma[[F]] \wedge \sigma[[G]]$
- $\sigma[[\neg F]] \equiv \neg\sigma[[F]]$
- $\models F \equiv \forall\ \sigma \in \textbf{St}^\infty:\ \sigma[[\mathsf{A}]]t$

Additional Notation

- $p' \equiv p(\forall 'v': v'/v)$
- $[\mathsf{A}]_f \equiv \mathsf{A} \vee (f' = f)$
- $\langle \mathsf{A} \rangle_f \equiv \mathsf{A} \wedge (f' \neq f)$
- *Unchanged* $f \equiv f' = f$
- $\Diamond F \equiv \neg \Box \neg F$
- $F \mapsto G \equiv \Box(F \Rightarrow \Diamond G)$
- $\mathsf{WF}_f(\mathsf{A}) \equiv (\Box\Diamond\langle \mathsf{A} \rangle_f) \vee (\Box\Diamond \neg \textit{Enabled } \langle \mathsf{A} \rangle_f)$
- $\mathsf{SF}_f(\mathsf{A}) \equiv (\Box\Diamond\langle \mathsf{A} \rangle_f) \vee (\Diamond\Box \neg \textit{Enabled } \langle \mathsf{A} \rangle_f)$

The Rules of Simple Temporal Logic

- STL1. $\dfrac{F \text{ provable by propositional logic}}{\Box F}$
- STL2. $\vdash \Box F \Rightarrow F$
- STL3. $\vdash \Box\Box F \equiv \Box F$
- STL4. $\dfrac{F \Rightarrow G}{\Box F \Rightarrow \Box G}$
- STL5. $\vdash \Box(F \land G) \equiv (\Box F) \land (\Box G)$
- STL6. $\vdash (\Diamond\Box F) \land (\Diamond\Box G) \equiv \Diamond\Box(F \land G)$
- LATTICE.
  $$\dfrac{\begin{array}{l} F \land (c \in S) \Rightarrow \\ (H_c \leadsto (G \lor \exists d \in S: (c > d) \land H_d)) \end{array}}{F \Rightarrow ((\exists c \in S: H_c) \leadsto G)}$$
  $>$ a well-founded partial order on set S

The Basic Rules of TLA

- TLA1. $$\frac{P \land (f = f') \Rightarrow P'}{\Box P \equiv P \land \Box[P \Rightarrow P']_f}$$
- TLA2. $$\frac{P \land \langle \mathsf{A} \rangle_f \Rightarrow Q \land [\mathsf{B}]_g}{\Box P \land \Box\langle \mathsf{A} \rangle_f \Rightarrow \Box Q \land \Box[\mathsf{B}]_g}$$

Additional Rules

- INV1. $$\frac{I \land [\mathsf{N}]_f \Rightarrow I'}{I \land \Box[\mathsf{N}]_f \Rightarrow \Box I}$$
- INV2. $\vdash \Box I \Rightarrow (\Box[\mathsf{N}]_f \equiv \Box[\mathsf{N} \land I \land I']_f)$
- WF1. $$\frac{\begin{array}{l} P \land [\mathsf{N}]_f \Rightarrow (P' \lor Q') \\ P \land \langle \mathsf{N} \land \mathsf{A} \rangle_f \Rightarrow Q' \\ P \Rightarrow \textit{Enabled}\ \langle \mathsf{A} \rangle_f \end{array}}{\Box[\mathsf{N}]_f \land \mathsf{WF}_f(\mathsf{A}) \Rightarrow (P \mapsto Q)}$$

Additional Rules

- WF2.

$$\frac{\begin{array}{l}\langle \mathsf{N} \wedge \mathsf{B}\rangle_f \Rightarrow \langle \overline{\mathsf{M}}\rangle_{\overline{g}} \\ P \wedge P' \wedge \langle \mathsf{N} \wedge \mathsf{A}\rangle_f \wedge \overline{\mathit{Enabled}\ \langle \mathsf{M}\rangle_g} \Rightarrow \mathsf{B} \\ P \wedge \overline{\mathit{Enabled}\ \langle \mathsf{M}\rangle_g} \Rightarrow \mathit{Enabled}\ \langle \mathsf{A}\rangle_f \\ \Box[\mathsf{N} \wedge \neg \mathsf{B}]_f \wedge \mathsf{WF}_f(\mathsf{A}) \wedge \Box F \\ \quad \wedge \Diamond\Box\overline{\mathit{Enabled}\ \langle \mathsf{M}\rangle_g} \Rightarrow \Diamond\Box P\end{array}}{\Box[\mathsf{N}]_f \wedge \mathsf{WF}_f(\mathsf{A}) \wedge \Box F \Rightarrow \overline{\mathsf{WF}_g(\mathsf{M})}}$$

- SF1.

$$\frac{\begin{array}{l}P \wedge [\mathsf{N}]_f \Rightarrow (P' \vee Q') \\ P \wedge \langle \mathsf{N} \wedge \mathsf{A}\rangle_f \Rightarrow Q' \\ \Box P \wedge \Box[\mathsf{N}]_f \wedge \Box F \Rightarrow \Diamond \mathit{Enabled}\ \langle \mathsf{A}\rangle_f\end{array}}{\Box[\mathsf{N}]_f \wedge \mathsf{SF}_f(\mathsf{A}) \wedge \Box F \Rightarrow (P \leadsto Q)}$$

- SF2.

$$\frac{\begin{array}{l}\langle \mathsf{N} \wedge \mathsf{B}\rangle_f \Rightarrow \langle \overline{\mathsf{M}}\rangle_{\overline{g}} \\ P \wedge P' \wedge \langle \mathsf{N} \wedge \mathsf{A}\rangle_f \Rightarrow \mathsf{B} \\ P \wedge \overline{\mathit{Enabled}\ \langle \mathsf{M}\rangle_g} \Rightarrow \mathit{Enabled}\ \langle \mathsf{A}\rangle_f \\ \Box[\mathsf{N} \wedge \neg \mathsf{B}]_f \wedge \mathsf{SF}_f(\mathsf{A}) \wedge \Box F \\ \quad \wedge \Box\Diamond\overline{\mathit{Enabled}\ \langle \mathsf{M}\rangle_g} \Rightarrow \Diamond\Box P\end{array}}{\Box[\mathsf{N}]_f \wedge \mathsf{SF}_f(\mathsf{A}) \wedge \Box F \Rightarrow \overline{\mathsf{SF}_g(\mathsf{M})}}$$