November 7, 2000

BRICS, University of Aarhus
Frank D. Valencia

An Introduction to Temporal Logics
(D. Scott, Advice on Modal Logic, 1970)

and to investigate their properties... What is essential is to single out important concepts

There is no weight to the claim that the original system must therefore be replaced by the new one. In a first-order way, take no notice of such arguments.

because it can be translated into some simpler language... One often hears that modal logic is pointless...
Branching Time (several possible futures): $\mathcal{B}$ induces a tree-like orders on $\mathcal{M}$.

Linear Time (one unique future): $\mathcal{L}$ induces a linear orders on $\mathcal{M}$.

Possible world semantics $(\mathcal{M}, \mathcal{P})$.

TL as Modal Logic: Linear and Branching Time.
$\diamondsuit$ means "next $A$", or "tomorrow $A$"

$\Box$ means "sometime $A$", or "eventually $A$"

$\lozenge$ means "always $A$", or "henceforth $A$"

$d$ is a proposition (called state formula)

$$\forall \diamondsuit \forall \Box \forall B \land \forall A \neg \rightarrow d =: \ldots 'B', 'A$$

Syntax

Temporal Language
is "Sometimes equivalent to "Not never" i.e. " 

\[ \Diamond \equiv \forall \Diamond \neg \Box \neg A \]

\[ (G \text{Causal} \equiv \neg \text{GetCausal}) \]

\[ \text{Fairness} \equiv \text{Success} \]

Examples
\[ \{ \alpha \in \mathcal{A} \mid \alpha(0) = s, \text{ i.e. \text{ "tree rooted at \textbf{s}"}} \} = s' \mathcal{A} \]

Denote by \( s' \mathcal{A} \) the suffix closure of \( s ' \).

If \( \alpha = s_0, s_1, s_2, \ldots \) then \( a^n \alpha \) for \( n > 0 \).

Auxiliary definitions:

\( \mathcal{S} \) is a set of infinite sequences of states.

\( \mathcal{S}' \) is a set of states (truth-valued functions on the set of propositions).

A model is a structure \( (\mathcal{S}', \mathcal{A}, \mathcal{W}) \) where...
\[\forall T \models \varphi : 0 \geq \diamond \forall \varphi \iff \forall \square T \models \varphi\]
\[\forall T \models \varphi : 0 \geq \diamond \exists \varphi \iff \forall \Diamond T \models \varphi\]

\[\forall T \not\models \varphi \iff \forall \neg T \models \varphi\]
\[\forall T \models \varphi \land \forall T \models \chi \iff \forall \chi \land \varphi T \models \varphi\]
\[\text{true} = (d)(0)\varphi \iff \forall T \not\models d T \models \varphi\]

Where

\[\forall T \models \varphi \iff \forall \models (S, \exists)(\exists) \text{ for all sequences } a \in \exists, a \models T, A\]

A Linear TL Semantics
Examples:

\[\Diamond T \equiv A \overset{\triangleright}{\land} A\]

"Sometimes" is "Not ever", i.e., \[\Diamond T \equiv \Box \Box T \land A\]
\[ \forall b \models (\langle \rangle a : 0 < \langle \rangle a) \quad \text{for every} \quad a \in S, \text{there is} \quad s' \in \text{states} \quad s' \models b\]

\[ \forall b \models (\langle \rangle a : 0 < \langle \rangle a) \quad \text{for all} \quad a \in S, \text{and} \quad b \models s \quad \text{true} = (d)s \]

Where

\( M = \langle S, \Delta, \models \rangle \) models \( A \) if for all states \( s \in S' \), \( s' \models b \),

A Branching TL Semantics

November 7, 2000
Thus, "sometimes is sometimes Not Never".

"Sometimes is not "Not never" i.e., $A \not\equiv B \boxminus A$.

$\Box A \equiv B \Box A$
Example. For any $S$, $A \rightarrow B$ for any $V$ and $\forall V$.

$(\exists S, V) = W$ for all $W$ such that $W$ for all $W$.

$S \equiv (\exists V, W) \equiv V$

$S$-equivalence ($\equiv$) •

$W$ for all $W$. $B \equiv V$

Strong-equivalence ($\equiv$) •

$B \rightarrow W$ for all $V$. $X \rightarrow W$

$W \equiv (\exists V, W)$ •

Comparing Expressiveness
Theorem (Lamport 1980). For any $S$, and non-trivial $p$, if $p$ is not $\square\Diamond$ equivalent to any $BT$ formula.

Expressiveness

November 7, 2000
(\forall \text{true}_{\top}) \cap (\forall \text{true}_{\top}) \subseteq \forall \text{true}_{\top} \cap \forall \text{true}_{\top} \equiv \forall \Diamond \bullet

Notice that:

\forall T = \exists \forall \alpha \neq \top \text{ and } T = \forall \text{true}_{\top} \iff \forall \alpha : \forall \gamma \exists \forall \beta \forall \gamma \geq \beta \text{ for all } \gamma \text{ for some } \beta, \forall \forall \beta (\forall \beta \cap \forall \beta) \neq \forall \beta \cap \forall \beta \equiv \forall \Diamond \bullet

New operator \LaTeX{}'s Expressiveness.
Inference Rules are

\[
\begin{align*}
\text{Modus Ponens and Instantiation:} \\
&\frac{d}{\square d} \\
&\frac{d \text{ valid}}{\square d}
\end{align*}
\]

\[
\begin{align*}
\forall \Diamond & \equiv (B, \forall) \land ((B, \forall \forall) \land (B, \forall) \land (B \land \forall)) \equiv (B, \forall) \\
\forall \Diamond & \equiv \forall \forall \land (\forall \forall \equiv \forall \forall) \\
\forall \square & \equiv \forall \forall \land (\forall \forall \equiv \forall \forall) \\
\forall \Diamond & \equiv \forall \square \\
\forall & \equiv \forall \square
\end{align*}
\]

Temporal Axioms

Complete deductive system (by Maibaum and Pnueli, 1982) for strong validity:

Proving LT strong validity

November 7, 2000
is strongly valid.

\[ A \models \Box \]

Theorem.

Providing \( \mathcal{L}_S \)-validity (Lamport 1980) for any \( S \) - valid \( A \) there is a \( S \) - valid \( p \) such that
\[ M \models \phi \iff \exists \nu \models M \]

Theorem. (Manaa and Pnueli, 1992). For any \( M \), \( \nu \) s.t. \( \nu \models M \) exists.

For some models \( M \), one can define \( \nu \) s.t.

Proving \( \nu \models M \)-validity (The Semantic Approach).
\[ T \models \varphi \quad \text{(a) try find proof of} \quad B \models \varphi \quad \text{(b) try model checking} \]

Answer: (a) try find proof of \[ B \models \varphi \]

\[ (\forall z \in \text{counter}) \quad T \models \langle \text{do counter} := \text{counter} + 1 \rangle \]

Verification: \[ T \models \varphi \quad \text{e.g.} \]

Specification: Formulate explicitly program properties \[ (\forall z \in \text{counter}) \]

Programs are represented as restricted models \[ \mathcal{M} \]

Application to C#: (Specification and Verification)
Conclusion

The TLL provides a convenient language for specifying and proving programs proper-

November 7, 2000

Concurrent Constraint Programming