Benefits of Interval Temporal Logic for Specification of Concurrent Systems

Ben Moszkowski

Software Technology Research Laboratory
De Montfort University
Leicester
Great Britain

email: benm@dmu.ac.uk

http://www.tech.dmu.ac.uk/~benm
Introduction

- **Intervals** and **discrete linear state sequences** offer a compellingly natural and flexible way to model computational processes involving hardware or software.

  Finite or infinite state sequence: \[ \cdots \cdots \cdots \]

- **Interval Temporal Logic (ITL)** is an established formalism (over 25 years old) for reasoning about such phenomena.
- It includes operators for **sequentially** combining formulas.
- For example, if \( A \) and \( B \) are formulas, so are following
  \[ A; B \quad ("\text{chop}") \quad A^* \quad ("\text{chop-star}"). \]
- **ITL** can express various **imperative programming** constructs (e.g., **while-loops**) and has **executable subsets**.
- The **Duration Calculus (DC)** is a **real-time** extension of ITL for hybrid systems.
ITL, Point-Based Temporal Logic and Time Reversal

- For several decades, widely held conventional wisdom: **Point-based** linear-time temporal logic (e.g., PTL) is much superior to **ITL** for safety and liveness **concurrency** properties.
- ITL’s computational intractability seen as limit to **tool support**.
- We offer evidence suggesting that a **reexamination** is in order.
- Our presentation uses Peterson’s mutual exclusion algorithm.
- **Interval** properties can be more natural and at higher level than **point-based** ones. Use transformations with **time reversal** and from **infinite time to finite time**.
- For **tool support**, transform to lower-level, point-based formulas.

Thus point-based linear-time temporal logic primarily serves as a **subordinate** formalism.
Syntax of Propositional ITL

In what follows, \( p \) is any propositional variable and both \( A \) and \( B \) themselves denote formulas in Propositional ITL (PITL):

- \( \text{true} \)
- \( p \)
- \( \neg A \)
- \( A \lor B \)
- \( \text{skip} \)
- \( A; B \)
- \( A^* \).

“;” is called chop.

“\( * \)” is called chop-star.
Intervals of Time

Discrete linear time is represented by **intervals** (i.e., sequences of states).

An **interval** $\sigma$ has a finite, nonzero number of states $\sigma_0, \sigma_1, \ldots$.

Can naturally extend to also permit infinite (i.e., $\omega$) states.

Each state $\sigma_i$ maps each variable $p, q, \ldots$ to *true* or *false*.

The value of $p$ in the state $\sigma_i$ is denoted $\sigma_i(p)$.

Hence, propositional variables $p, q, \ldots$ are **local** to states.
Semantics of PITL for Finite Time

Let $\sigma \models A$ denote that the interval $\sigma$ satisfies the PITL formula $A$. Below is excerpt from semantics of basic PITL constructs:

- $\sigma \models p$ iff $\sigma_0(p) = true$. (Use $p$’s value in $\sigma$’s initial state $\sigma_0$)
- $\sigma \models \neg A$ iff $\sigma \not\models A$.
- $\sigma \models A \lor B$ iff $\sigma \models A$ or $\sigma \models B$.

Pictorial summary of finite-time semantics of interval constructs:

Each pair of adjacent subintervals share a state.

Can also have “chomp”, a version of chop with unit gap: $A; skip; B$. 
Some Sample Formulas

(Assume Finite Time)

\begin{align*}
P & \quad p: \begin{array}{c} t \ t \ t \ f \ f \end{array} \\
\land \ skip & \quad \land \skip; \begin{array}{c} p: \begin{array}{c} t \ f \ f \end{array} \end{array} \\
(\bigcirc \ p) & \quad (\bigcirc \ \begin{array}{c} p: \begin{array}{c} f \ t \ t \ f \ f \end{array} \end{array} \\
true; \neg p & \quad true; \begin{array}{c} \neg p: \begin{array}{c} f \ t \ t \ f \ f \end{array} \end{array} \\
(\bigotimes \neg p) & \quad (\bigotimes \ \neg p: \begin{array}{c} f \ t \ t \ f \ f \end{array} \\
\neg (true; \neg p) & \quad \neg (true; \begin{array}{c} \neg p: \begin{array}{c} t \ t \ t \ t \ t \ t \ t \ t \end{array} \end{array} \\
(\Box p) & \quad (\Box \ \begin{array}{c} p: \begin{array}{c} t \ t \ t \ t \ t \ t \ t \ t \end{array} \end{array} \end{align*}
Notes

- $A, A', B, C, \ldots$ denote arbitrary formulas.

- $w, w', \ldots$ denote state formulas (no temporal operators).

- If $\sigma \models A$ for some $\sigma$, then $A$ is satisfiable.

- If $\sigma \models A$ for all $\sigma$, then $A$ is valid. Denote as $\models A$.

- Can extend PITL to include infinite time and $A^\omega$ (chop-omega).
Some Derivable ITL Operators

Define \( \text{false} \), \( A \land B \), \( A \supset B \) (implies), \( A \equiv B \) (equivalence), \ldots.

Propositional Temporal Logic (PTL) with finite \& infinite time:

\[
\begin{align*}
\Box A & \quad \equiv \quad \text{skip}; A \quad (\text{“next”}) \\
more & \quad \equiv \quad \Box \text{true} \quad (\geq 2 \text{ states}) \\
\text{empty} & \quad \equiv \quad \neg more \quad (\text{just 1 state}) \\
\text{finite} & \quad \equiv \quad \Diamond \text{empty} \quad (\text{finite}) \\
\text{inf} & \quad \equiv \quad \neg \text{finite} \quad (\text{infinite}) \\
\Diamond A & \quad \equiv \quad \text{finite}; A \quad (\text{“eventually”}) \\
\square A & \quad \equiv \quad \neg \Diamond \neg A \quad (\text{“henceforth”}) \\
\text{fin } A & \quad \equiv \quad \square (\text{empty } \supset A) \quad (\text{final state}) \\
A \leftarrow B & \quad \equiv \quad \text{finite } \supset ( (\text{fin } A) \equiv B) \quad (\text{temporal assignment})
\end{align*}
\]

Interval-oriented operators:

\[
\begin{align*}
\Diamond A & \quad \equiv \quad (A \land \text{finite}); \text{true} \quad (\text{Some finite prefix}) \\
\square A & \quad \equiv \quad \neg \Diamond \neg A \quad (\text{ALL finite prefixes}) \\
A^\omega & \quad \equiv \quad (A \land \text{finite})^* \land \text{inf} \quad (\text{Chop-omega})
\end{align*}
\]
Sequential Compositionality with Temporal Fixpoints

- ITL-based **assumptions** and **commitments** for a system $Sys$:

  $$w \land As \land Sys \supset Co \land fin w'$$

- Sequential composition of two formulas $Sys$ and $Sys'$:

  $$\frac{\models w \land As \land Sys \supset Co \land fin w'}{\models w' \land As \land Sys' \supset Co \land fin w''}$$

  $$\frac{\models w' \land As \land Sys' \supset Co \land fin w''}{\models w \land As \land (Sys; Sys') \supset Co \land fin w''}$$

- Zero or more iterations of a formula $Sys$:

  $$\frac{\models w \land As \land Sys \supset Co \land fin w}{\models w \land As \land Sys^* \supset Co \land fin w}$$

See Moszkowski '94 (also '96, '98)
(Shares some features with Jones’ rely/guarantee conditions)
Commitments as Fixpoints

• Formalization:
  \[ \models \ Co \equiv Co^* \]

• Intuition:
  \( Co \) is true on an interval
  \text{iff} \( Co \) is true on each of a sequence of subintervals.

• Examples:
  – “\( p \)'s values in the initial and final states are equal” \( (p \leftarrow p) \)
    \[
    \begin{array}{cccccccc}
    \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
p: f & t & t & f & f & t & t & f \\
    \end{array}
    \]
    \[
    (p \leftarrow p) \quad (p \leftarrow p) \quad (p \leftarrow p)
    \]

  – Any formula \( A^* \) \quad Example: “even length” \( (skip; skip)^* \)

  – Any formula expressible as \( \Box (more \supset \lozenge B) \).
    Examples: \( \Box (more \supset w) \), \( \Box ((more \land w) \supset \lozenge w') \)
Peterson’s Mutual Exclusion Algorithm

Process $P_0$:

do forever
  
  (. . . Noncritical section . . .)

  $flag_0 := 1$;
  $turn := 0$;

  $await (flag_1 = 0 \lor turn = 1)$;

  $cs_0 := 1$;

  (. . . Critical section . . .)

  $cs_0 := 0$;

  $flag_0 := 0$;

  (. . . Noncritical section . . .)

Overall program has two processes $P_0$ and $P_1$:

$flag_0 = flag_1 = cs_0 = cs_1 = 0 \land turn = 0 \land (P_0 \parallel P_1)$

Safety Property

Can express behaviour of the program in PITL.
Recall that $\□ A$ means: “$A$ is true in all finite prefix subintervals”.
Safety property for mutual exclusion expressible as

$$\□(more \supset \Diamond B).$$

Includes formulas of form $\□((more \land fin w) \supset \Diamond C)$.
Safety property for individual process in Peterson’s algorithm:

$$\□((more \land fin(cs_0 = 1)) \supset \Diamond(test;\newline\quad stability;\newline\quad test;\newline\quad stability)).$$

“Star-free” operand of $\□$ reducible to point-based temporal logic.
Can use $\□$ instead of past-time constructs.
Time Reversal and Chop-Star Fixpoints

Let $\sigma^r$ denote **temporal reverse** of finite interval $\sigma$:  $\sigma\vert_\sigma \ldots \sigma_0$. Let $A^r$ be like $A$ in **reverse**:  $\sigma^r \models A$ iff $\sigma \models A^r$ for finite intervals. Sample finite-time semantic equivalences:

$$(A \lor B)^r \equiv A^r \lor B^r \quad (A;B)^r \equiv B^r;A^r \quad more^r \equiv more \quad (\Box A)^r \equiv \square(A^r).$$

For finite time:  $\models finite \supset A$ iff $\models finite \supset A^r$.

Time reversal of $\Box(more \supset \diamondsuit B)$ with finite time:

$$\Box(more \supset \diamondsuit B)^r \equiv \square(more \supset \diamondsuit B^r).$$

Already have that $\square(more \supset \diamondsuit B^r)$ is chop-star fixpoint:

$$\models \square(more \supset \diamondsuit B^r) \equiv \left(\square(more \supset \diamondsuit B^r)\right)^*.$$  

Using time reversal of this:

$$\models finite \supset \left(\square(more \supset \diamondsuit B^r) \equiv \left(\square(more \supset \diamondsuit B^r)\right)^*\right)^r.$$  

Simplify:  $\models finite \supset \Box(more \supset \diamondsuit B) \equiv \left(\Box(more \supset \diamondsuit B)\right)^*$.  

\(\rightarrow\) Can extend proof to infinite time.
Reduction to Conventional Temporal Logic

- Want to show: \( \mathcal{F}(\text{more } \supset \Diamond B) \land \mathcal{F}(\text{more } \supset \Diamond B') \supset \square w \).

Example: In Peterson’s algorithm, let \( w \) be \( cs_0 = 0 \lor cs_1 = 0 \).

Re-express \( \square w \) as \( \mathcal{F}\text{ fin } w \).

Can test validity for finite time. Readily extends to infinite time.

Or test by reducing to finite time with point-based temporal logic:

\[
(\text{more } \supset \Diamond B) \land (\text{more } \supset \Diamond B') \supset \text{ fin } w
\]

- Temporal reverse of \( \mathcal{F}(\text{more } \supset \Diamond B) \) reducible to PTL formula.
  Simplifies testing \( w \land As \land Sys \supset Co \land \text{ fin } w \).
  \( Sys \) is like a regular expression and easy to reverse.

- In following, for any \( \mathcal{F} \) formula only need to check for finite time:

\[
\models w \land \text{ finite } \land Sys^* \supset \mathcal{F} A \land \text{ fin } w \quad \Rightarrow \quad \models w \land Sys^o \supset \mathcal{F} A \land \text{ fin } w.
\]

Helps show \( \mathcal{F}(\text{more } \supset \Diamond B) \) is chop-star fixpoint for infinite time.
Some Related Observations by Others

• Textbook using Duration Calculus (ITL variant for real time):

  *Real-Time Systems:*

  *Formal Specification and and Automatic Verification*


Regarding **point-based logics**: “complicated reasoning”

Regarding **timed process algebras**: “difficult to calculate with”

• KIV interactive theorem prover at Univ. of Augsburg, Germany. Uses ITL as **frontend** and as **backend** for UML, Statecharts, etc. FACS journal paper, 2009 (lock-free algs., linearizability, Re/Gu):

  “The (program’s) line numbers . . . are not used in KIV.”

  “An additional translation to a special normal form (as e.g. in TLA) using explicit program counters is not necessary.”
Conclusions

We have presented some ideas about reasoning in ITL:

• **Temporal fixpoints** for sequential compositionality
• **Prefix subintervals** (instead of past-time constructs).
• **Time reversal** for reduction to point-based temporal logic.
• Reductions from **infinite time** to **finite time**.

View **point-based temporal logic** as **lower level** and **subordinate**. Approach is intriguing (and **axiomatisable**) but needs further study. Appears related to Dijkstra’s “**Gotos considered harmful**” thesis. Somewhat analogous to Vardi’s “**Final Showdown**” article relating **linear-time** and **branch-time** temporal logics for model checking.

For more about ITL: **ITL webpages** – maintained by Antonio Cau:

http://www.tech.dmu.ac.uk/STRL/ITL/

Using search engine: “**Interval Temporal Logic**”