Conjunction Capers

A TLA$^+$ Truffle

Image: Amazon

Ron Pressler, October 2020

“Just as natural pearls grow from grains of sand that have irritated oysters, these programming pearls have grown from real problems that have irritated programmers. The programs are fun, and they teach important programming techniques and fundamental design principles.”

Jon Bentley, Programming Pearls, Communications of the ACM

TLA+ Truffle

In the spirit of Programming Pearls and Graphics Gems

- an instructive example of program calculation or proof, or
- a nifty presentation of an old or new data structure, or
- an interesting application or programming technique.

“polished, elegant, instructive, entertaining”

Richard Bird, *How to Write a Functional Pearl*

The Technique

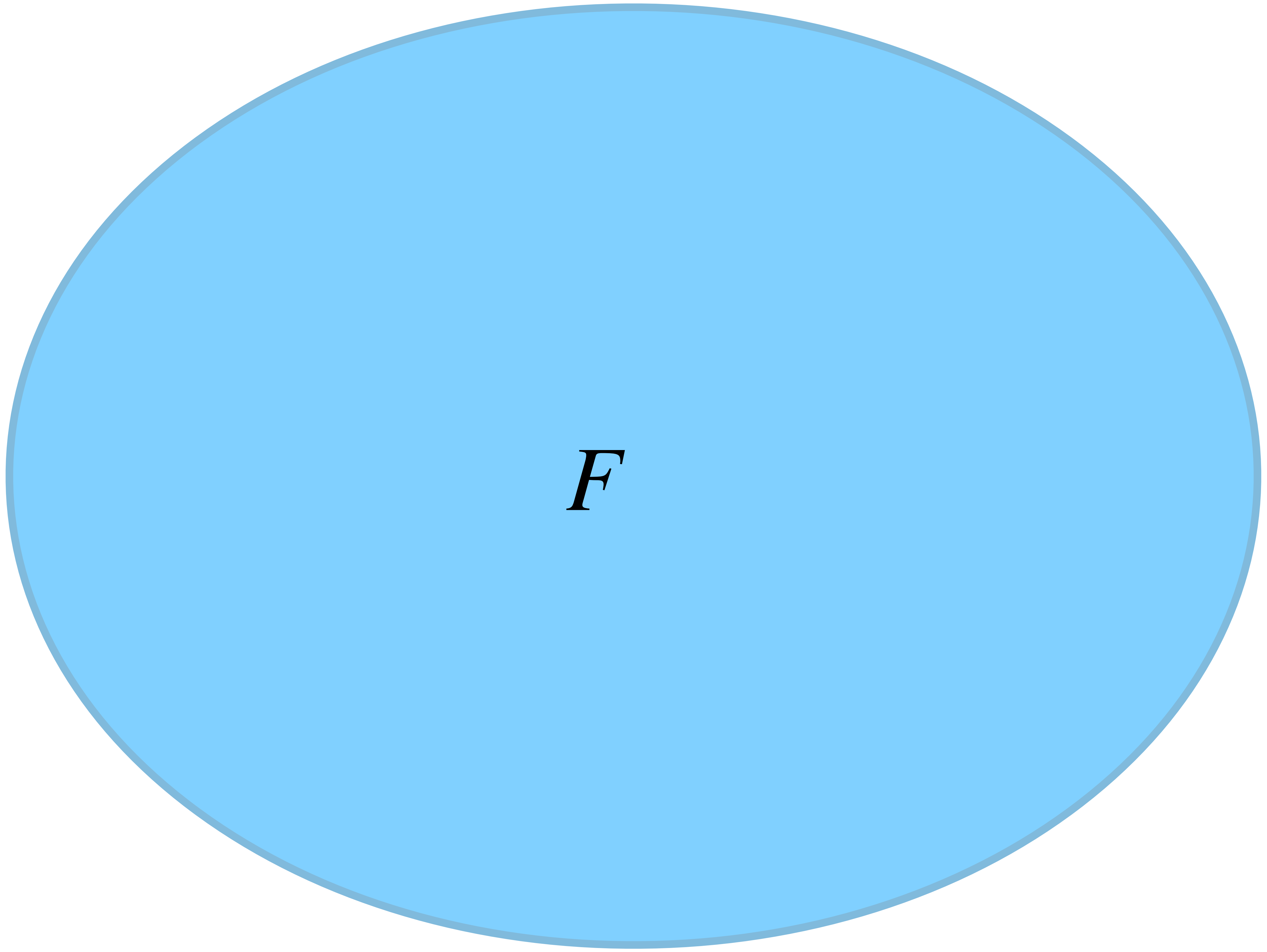
Image: Amazon

The Canonical TLA Formula

A State-Machine Specification

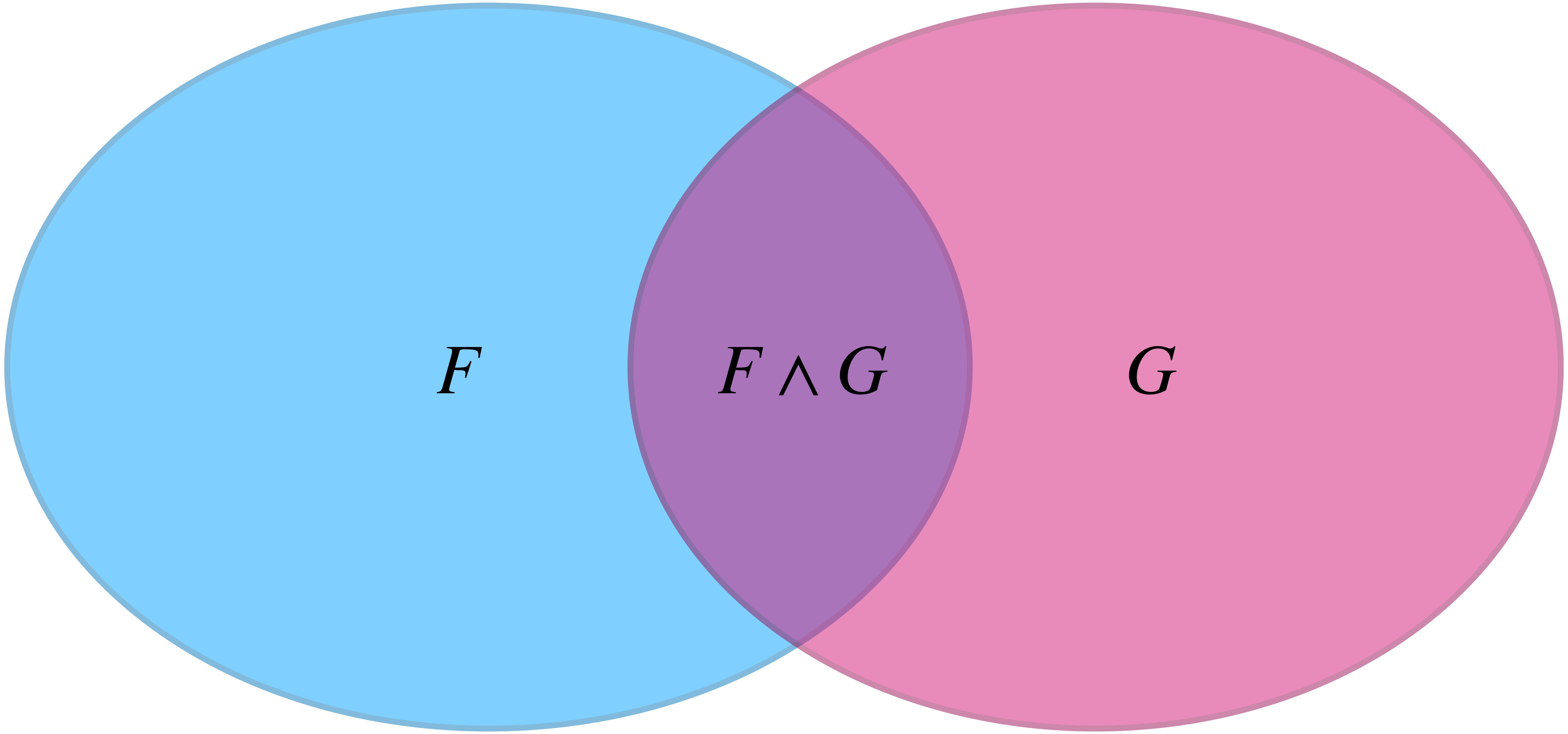
$$Init \land \Box [Next]_{vars} \land Fairness$$

The Meaning of a TLA Formula



The Meaning of a TLA Formula

Conjunction is Composition



The Meaning of a TLA Formula

Conjunction

$$\frac{F \Rightarrow A \qquad G \Rightarrow B}{F \land G \Rightarrow A} \qquad \frac{F \Rightarrow A \qquad G \Rightarrow B}{F \land G \Rightarrow B}$$

Conjunction of Canonical TLA Formulas

$$Init \land \Box[Next]_{vars} \land Fairness$$

$$\land$$

$$Init \land \Box[Next]_{vars} \land Fairness$$

Conjunction of Canonical TLA Formulas

$\land$ $Init$	$\land$ $Init$
$\land$ $\Box[Next]_{vars}$	$\land$ $\Box[Next]_{vars}$
$\land$ $Fairness$	$\land$ $Fairness$

Conjunction of Canonical TLA Formulas

$\land \; Init \qquad \land \; Init$

$\land \; \Box[Next]_{vars} \; \land \; \Box[Next]_{vars}$

$\land \; Fairness \qquad \land \; Fairness$

$$\equiv \; Init \land Init \land \Box[???]_{???} \land Fairness \land Fairness$$

Conjunction of Canonical TLA Formulas

(1) $$(p \lor q) \land (r \lor s) \equiv \begin{array}{l} \lor\, p \land r \\ \lor\, p \land s \\ \lor\, q \land r \\ \lor\, q \land s \end{array}$$

(2) $$\Box A \land \Box B \equiv \Box(A \land B)$$

(3) $$\begin{aligned} \Box[A]_{vars} &\equiv \Box(A \lor \text{UNCHANGED } vars) \\ &\equiv \Box(A \lor vars' = vars) \end{aligned}$$

Conjunction of Canonical TLA Formulas

$$\Box [Next]_{vars} \land \Box [Next]_{vars}$$

$$\begin{aligned} \equiv \Box(\, &\lor Next \land Next \\ &\lor Next \land \text{UNCHANGED } vars \\ &\lor Next \land \text{UNCHANGED } vars \\ &\lor \text{UNCHANGED } vars \land \text{UNCHANGED } vars) \end{aligned}$$

Conjunction of Canonical TLA Formulas

$$\Box[Next]_{vars} \land \Box[Next]_{vars}$$

$$\equiv \quad \Box[\begin{array}{l} \lor\ Next \land Next \\ \lor\ Next \land \text{UNCHANGED } vars \\ \lor\ Next \land \text{UNCHANGED } vars \end{array}]_{\langle vars, vars \rangle}$$

Conjunction of Canonical TLA+ Formulas

$$Compose(NextA, varsA, NextB, varsB) \triangleq$$

$$\begin{aligned}
&\lor NextA \land NextB \\
&\lor NextA \land \text{UNCHANGED } varsB \\
&\lor NextB \land \text{UNCHANGED } varsA
\end{aligned}$$

Conjunction of Canonical TLA+ Formulas

$$Compose(NextA,\ varsA,\ NextB,\ varsB) \triangleq$$

$$\begin{aligned} &\lor NextA \land NextB \\ &\lor NextA \land \text{UNCHANGED } varsB \\ &\lor NextB \land \text{UNCHANGED } varsA \end{aligned}$$

$x' = A$ $\qquad$ $x' \in A$

Conjunction of Canonical TLA+ Formulas

$$Compose(NextA, varsA, NextB, varsB) \triangleq$$

$$\begin{aligned}
&\lor NextA \land NextB \\
&\lor \text{UNCHANGED } varsB \land NextA \\
&\lor \text{UNCHANGED } varsA \land NextB
\end{aligned}$$

Conjunction of Canonical TLA⁺ Formulas

$$Spec \triangleq \begin{array}{l} \land\ Init1 \land \Box[Next1]_{v1} \land Fairness1 \\ \land\ Init2 \land \Box[Next2]_{v2} \land Fairness2 \end{array}$$

$$Spec \triangleq \begin{array}{l} \land\ Init1 \land Init2 \\ \land\ \Box[Compose(Next1,\ v1,\ Next2,\ v2)]_{\langle v1,\ v2 \rangle} \\ \land\ Fairness1 \land Fairness2 \end{array}$$

Conjunction of Canonical TLA+ Formulas

$$
\begin{aligned}
Next12 &\triangleq Compose(Next1,\ v1,\ Next2,\ v2)\\
Next123 &\triangleq Compose(Next12,\ \langle v1,\ v2\rangle,\ Next3,\ v3)\\
Next1234 &\triangleq Compose(Next123,\ \langle v1,\ v2,\ v3\rangle,\ Next4,\ v4)\\
Next12345 &\triangleq Compose(Next1234,\ \langle v1,\ v2,\ v3,\ v4\rangle,\ Next5,\ v5)\\
Next123456 &\triangleq Compose(Next12345,\ \langle v1,\ v2,\ v3,\ v4,\ v5\rangle,\ Next6,\ v6)\\
Next1234567 &\triangleq Compose(Next123456,\ \langle v1,\ v2,\ v3,\ v4,\ v5,\ v6\rangle,\ Next7,\ v7)
\end{aligned}
$$

$$
\begin{aligned}
vars &\triangleq \langle v1,\ v2,\ v3,\ v4,\ v5,\ v6,\ v7\rangle\\
Init &\triangleq Init1 \land Init2 \land Init3 \land Init4 \land Init5 \land Init6 \land Init7\\
Next &\triangleq Next1234567
\end{aligned}
$$

$$
Spec \triangleq Init \land \Box[Next]_{vars}
$$

Example I

Image: Amazon

Behavioral Programming

David Harel, Assaf Marron, Gera Weiss, CACM, 2012

DOI:10.1145/2209249.2209270

A novel paradigm for programming reactive systems centered on naturally specified modular behavior.

BY DAVID HAREL, ASSAF MARRON, AND GERA WEISS

Behavioral Programming

SPELLING OUT THE requirements for a software system under development is not an easy task, and translating captured requirements into correct operational software can be even harder. Many technologies (languages, modeling tools, programming paradigms) and methodologies (agile, test-driven, model-driven) were designed, among other things, to help address these challenges. One widely accepted practice is to formalize requirements in the form of use cases and scenarios. Our work extends this approach into using scenarios

To illustrate the naturalness of constructing systems by composing behaviors, consider how children may be taught, step-by-step, to play strategy games (See Gordon et al.[14]). For example, in teaching the game of Tic-Tac-Toe, we first describe rules of the game, such as:

`EnforceTurns`: To play, one player marks a square in a 3 by 3 grid with `X`, then the other player marks a square with `O`, then `X` plays again, and so on;

`SquareTaken`: Once a square is marked, it cannot be marked again;

`DetectXWin/DetectOWin`: When a player places three of his or her marks in a horizontal, vertical, or diagonal line, the player wins;

Now we may already start playing. Later, the child may infer, or the teacher may suggest, some tactics:

`AddThirdO`: After placing two `O`s in a line, the `O` player should try to mark the third square (to win the game);

`PreventThirdX`: After the `X` player marks two squares in a line, the `O` player should try to mark the third square (to foil the attack); and

`DefaultOMoves`: When other tactics are not applicable, player `O` should prefer the center square, then the cor-

Behavioral Programming

Tic-Tac-Toe

1. Board: At each step, an X or an O is marked on the board

$$
\begin{array}{l}
\textsc{variable}\ board \\
v1 \triangleq \langle board \rangle
\end{array}
$$

$$
\begin{array}{lcl}
N & \triangleq & 3 \\
Empty & \triangleq & \text{“-”} \\
Player & \triangleq & \{\text{“X”}, \text{“O”}\} \\
Mark & \triangleq & Player \\
Square & \triangleq & \{Empty\} \cup Mark
\end{array}
$$

$$BoardType \triangleq board \in [(1 \,..\, N) \times (1 \,..\, N) \rightarrow Square]$$

$$BoardFull \triangleq \forall\, i, j \in 1 \,..\, N : board[i, j] \neq Empty$$

$$Init1 \triangleq board = [i, j \in 1 \,..\, N \mapsto Empty]$$

$$Next1 \triangleq \exists\, i, j \in 1 \,..\, N, mark \in Mark : \begin{array}{l} \land\, board[i, j] = Empty \\ \land\, board' = [board \ \textsc{except}\ ![i, j] = mark] \end{array}$$

$$Board \triangleq Init1 \land \Box[Next1]_{v1}$$

$$TicTacToe1 \triangleq Board$$

Properties we can state at this point:

$$\textsc{theorem}\ TicTacToe1 \Rightarrow \Box BoardType$$

$$
\begin{array}{l}
OnceSetAlwaysSet \triangleq \\
\quad \forall\, i, j \in 1 \,..\, N : \Box(\exists\, mark \in Mark : board[i, j] = mark \Rightarrow \Box(board[i, j] = mark)) \\
\textsc{theorem}\ TicTacToe1 \Rightarrow OnceSetAlwaysSet
\end{array}
$$

Behavioral Programming

Tic-Tac-Toe

2. *EnforceTurns*: X and O play in alternating turns

$$\text{VARIABLE } current, \quad turn$$

Necessary for some properties we may wish to state

$$v2 \triangleq \langle v1, turn, current \rangle$$

$$Other(player) \triangleq \text{IF } player = \text{"X" THEN "O" ELSE "X"}$$

$$Opponent \triangleq Other(current)$$

$$TurnType \triangleq \land\, current \in Player \quad \land\, turn \in Nat$$

$$Init2 \triangleq \land\, turn = 0 \quad \land\, current = \text{"X"}$$

X starts

$$Next2 \triangleq \land\, turn' = turn + 1 \quad \land\, current' = Opponent \quad \land\, \exists\, i, j \in 1 \,..\, N : \land\, board[i, j] = Empty \;\land\, board'[i, j] = current$$

$$EnforceTurns \triangleq Init2 \land \Box[Next2]_{v2}$$

$$TicTacToe2 \triangleq TicTacToe1 \land EnforceTurns$$

Properties we can state at this point:

$$\text{THEOREM } EnforceTurns \Rightarrow TurnType$$

$$Alternating \triangleq \Box[current' \neq current]_{v2}$$

$$\text{THEOREM } EnforceTurns \Rightarrow Alternating$$

Behavioral Programming

Tic-Tac-Toe

Statistics

State space progress (click column header for graph)

Time	Diameter	States Found	Distinct States
00:00:25	10	21,630	5,478
00:00:05	5	580	412
00:00:02	0	1	1

3. *DetectWin*: Detect win or draw and end game

VARIABLE win

$v3 \triangleq \langle v2, win \rangle$

$Result \triangleq Player \cup \{\text{"Draw"}\}$

$WinType \triangleq win \in \{Empty\} \cup Result$

$GameEnd \triangleq win \in Result$

$$\begin{aligned} Line \triangleq\ & \{[i \in 1 \,..\, N \mapsto \langle i, y \rangle] : y \in 1 \,..\, N\} && \text{horizontal} \\ \cup\ & \{[i \in 1 \,..\, N \mapsto \langle x, i \rangle] : x \in 1 \,..\, N\} && \text{vertical} \\ \cup\ & \{[i \in 1 \,..\, N \mapsto \langle i, i \rangle]\} \cup \{[i \in 1 \,..\, N \mapsto \langle i, N - i + 1 \rangle]\} && \text{diagonal} \end{aligned}$$

$f \circ g \triangleq [x \in \text{DOMAIN } g \mapsto f[g[x]]]$

$BoardLine(line) \triangleq board \circ line$

$Won(player) \triangleq \exists\, line \in Line : BoardLine(line) = [i \in 1 \,..\, N \mapsto player]$

$NoWin \triangleq \neg \exists\, player \in Player : Won(player)'$

$StopGame \triangleq board' = board$ UNCHANGED *board* – fails *TLC*

$Init3 \triangleq win = Empty$

$$\begin{aligned} Next3 \triangleq\ & \lor \land win = Empty \\ & \quad \land \lor \exists\, player \in Player : Won(player)' \land win' = player \\ & \quad\quad \lor NoWin \land BoardFull' \land win' = \text{"Draw"} \\ & \quad\quad \lor NoWin \land \neg BoardFull' \land \text{UNCHANGED } win \\ & \lor \land win \in Player \\ & \quad \land \text{UNCHANGED } win \\ & \quad \land StopGame \end{aligned}$$

$DetectWin \triangleq Init3 \land \Box[Next3]_{v3}$

$TicTacToe3 \triangleq TicTacToe2 \land DetectWin$

Properties we can state at this point:

THEOREM $DetectWin \Rightarrow WinType$

$GameEndsWhenPlayerWins \triangleq \Box(win \in Player \Rightarrow \Box[board' = board]\_v3)$

$GameEndsWhenPlayerWins \triangleq \Box[(win \in Player \Rightarrow \text{UNCHANGED } board)]_{v3}$

THEOREM $TicTacToe3 \Rightarrow GameEndsWhenPlayerWins$

$AtLeast5TurnsToWin \triangleq win \neq Empty \Rightarrow turn \geq 2 * N - 1$

THEOREM $TicTacToe3 \Rightarrow \Box(AtLeast5TurnsToWin)$

$GameEndsWhenBoardFull \triangleq BoardFull \Rightarrow GameEnd$

THEOREM $TicTacToe3 \Rightarrow \Box(GameEndsWhenBoardFull)$

Behavioral Programming

Tic-Tac-Toe

4. *AddThirdToWin*: Add third mark to win

So far, we've specified the rules of the game. Now we start adding tactic rules. This one says that if a player has two marks in a line they should place the third to win.

But we run into a problem: the tactics may be contradictory, and prioritization is required. b-threads can be prioritized, and we could simulate that mechanism with with maps of boolean functions, but that would be overly clever, especially in a simple specification such as this. Instead, we'll order the rules by their priority, and explicitly model priorities. This means that new rules would need to be inserted in the sequence of rules into their right position.

$$Count(mark, line) \triangleq Cardinality(\{i \in 1 \,..\, N : BoardLine(line)[i] = mark\})$$

$$CanWin(player) \triangleq \exists\, line \in Line : \begin{array}{l} \land\, Count(player, line) = N - 1 \\ \land\, Count(Empty, line) = 1 \end{array}$$

$$MarkLast(line) \triangleq \exists\, i \in 1 \,..\, N : \begin{array}{l} \land\, BoardLine(line)[i] = Empty \\ \land\, board'[line[i]] = current \end{array}$$

$$\begin{array}{lcl} v4 & \triangleq & v3 \\ Init4 & \triangleq & \text{TRUE} \\ Next4 & \triangleq & CanWin(current) \Rightarrow \\ & & \quad \exists\, line \in Line : Count(current, line) = N - 1 \land MarkLast(line) \end{array}$$

$$Priority1 \triangleq CanWin(current)$$

$$AddThirdToWin \triangleq Init4 \land \Box[Next4]_{v4}$$

$$TicTacToe4 \triangleq TicTacToe3 \land AddThirdToWin$$

Behavioral Programming

Tic-Tac-Toe

5. *BlockOpponentFromWinning*: Block the other player if they're about to win

$$
\begin{aligned}
v5 &\triangleq v4 \\
Init5 &\triangleq \text{TRUE} \\
Next5 &\triangleq CanWin(Opponent) \land \neg Priority1 \Rightarrow \\
&\qquad \exists\, line \in Line : Count(Opponent, line) = N - 1 \land MarkLast(line)
\end{aligned}
$$

$$Priority2 \triangleq Priority1 \lor CanWin(Opponent)$$

$$BlockOpponentFromWinning \triangleq Init5 \land \Box[Next5]_{v5}$$

$$TicTacToe5 \triangleq TicTacToe4 \land BlockOpponentFromWinning$$

Behavioral Programming

Tic-Tac-Toe

6. *MarkCenterIfAvailable*: Prefer center square

$$CenterSquare \triangleq \langle (N+1) \div 2,\ (N+1) \div 2 \rangle$$
$$CenterFree \triangleq board[CenterSquare] = Empty$$

$$v6 \triangleq v5$$
$$Init6 \triangleq \text{TRUE}$$
$$Next6 \triangleq (CenterFree \land \neg Priority2) \Rightarrow board'[CenterSquare] = current$$

$$Priority3 \triangleq Priority2 \lor CenterFree$$

$$MarkCenterIfAvailable \triangleq Init6 \land \Box[Next6]_{v6}$$

$$TicTacToe6 \triangleq TicTacToe4 \land MarkCenterIfAvailable$$

Properties we can state at this point:

$$FirstMarksSquare \triangleq turn = 1 \Rightarrow board[CenterSquare] \neq Empty$$
$$\text{THEOREM } TicTacToe6 \Rightarrow \Box(FirstMarksSquare)$$

Behavioral Programming

Tic-Tac-Toe

7. *MarkCornerIfAvailable*: Prefer corner square

$$CornerSquares \triangleq \{1, N\} \times \{1, N\}$$
$$CornerFree \triangleq \exists\, corner \in CornerSquares : board[corner] = Empty$$

$$v7 \triangleq v6$$
$$Init7 \triangleq \text{TRUE}$$
$$Next7 \triangleq (CornerFree \land \neg Priority3) \Rightarrow \exists\, corner \in CornerSquares : \land\ board[corner] = Empty \quad \land\ board'[corner] = current$$

$$Priority4 \triangleq Priority3 \lor CornerFree$$

$$MarkCornerIfAvailable \triangleq Init7 \land \Box[Next7]_{v7}$$

$$TicTacToe7 \triangleq TicTacToe6 \land MarkCornerIfAvailable$$

Properties we can state at this point:

$$SecondMarksCorner \triangleq turn = 2 \Rightarrow \exists\, corner \in CornerSquares : board[corner] \neq Empty$$
$$\text{THEOREM } TicTacToe7 \Rightarrow \Box(SecondMarksCorner)$$

The tactics are sufficient to always force a draw

$$AlwaysDraw \triangleq (win \notin Player)$$
$$\text{THEOREM } TicTacToe7 \Rightarrow \Box AlwaysDraw$$

Behavioral Programming

Tic-Tac-Toe

$$
\begin{aligned}
Next12 &\triangleq Compose(Next1,\ v1,\ Next2,\ v2)\\
Next123 &\triangleq Compose(Next12,\ \langle v1,\ v2\rangle,\ Next3,\ v3)\\
Next1234 &\triangleq Compose(Next123,\ \langle v1,\ v2,\ v3\rangle,\ Next4,\ v4)\\
Next12345 &\triangleq Compose(Next1234,\ \langle v1,\ v2,\ v3,\ v4\rangle,\ Next5,\ v5)\\
Next123456 &\triangleq Compose(Next12345,\ \langle v1,\ v2,\ v3,\ v4,\ v5\rangle,\ Next6,\ v6)\\
Next1234567 &\triangleq Compose(Next123456,\ \langle v1,\ v2,\ v3,\ v4,\ v5,\ v6\rangle,\ Next7,\ v7)
\end{aligned}
$$

$$
\begin{aligned}
vars &\triangleq \langle v1,\ v2,\ v3,\ v4,\ v5,\ v6,\ v7\rangle\\
Init &\triangleq Init1 \land Init2 \land Init3 \land Init4 \land Init5 \land Init6 \land Init7\\
Next &\triangleq Next1234567
\end{aligned}
$$

$$TicTacToe0 \triangleq Init \land \Box[Next]_{vars} \land \mathrm{WF}_{vars}(Next)$$

$$Terminates \triangleq win \neq Empty$$

$$\textsc{theorem}\ \ TicTacToe0 \Rightarrow \Diamond Terminates$$

Statistics

State space progress (click column header for graph)

Time	Diameter	States Found	Distinct States
00:00:06	10	378	62
00:00:05	8	188	44
00:00:02	0	1	1

Behavioral Programming

Resources

https://pron.github.io/files/TicTacToe.pdf

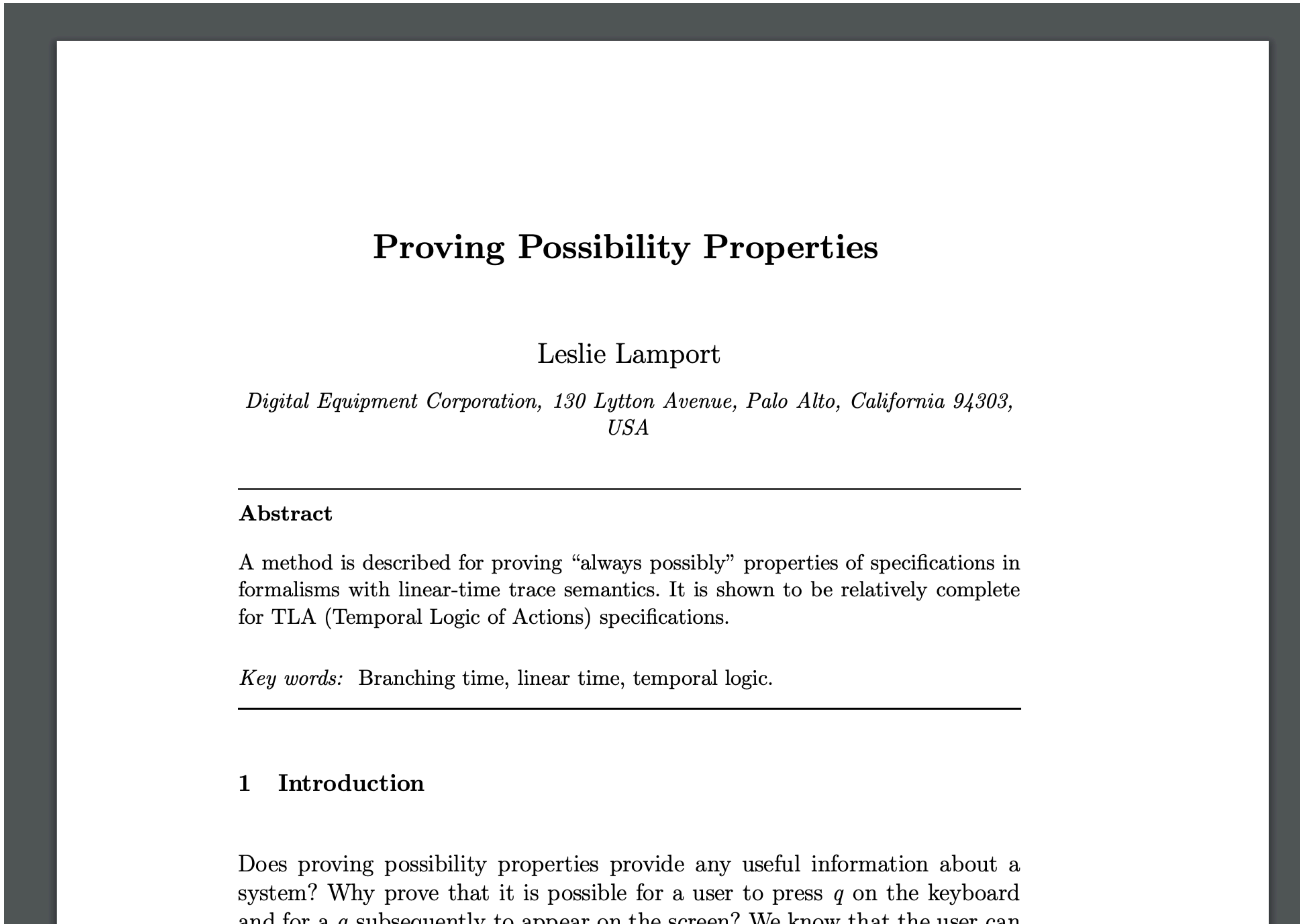
- Behavioral Programming Home Page: http://www.wisdom.weizmann.ac.il/~bprogram/
- Rethinking Software Systems: A friendly introduction to Behavioral Programming: https://youtu.be/PW8VdWA0UcA
- Bridging Specifications and Code: Behavioral Programming with React: https://vimeo.com/298554103

Example II

Image: Amazon

Proving Possibility Properties

Leslie Lamport, Theoretical Computer Science, 1998



Proving Possibility Properties

Leslie Lamport

Digital Equipment Corporation, 130 Lytton Avenue, Palo Alto, California 94303, USA

Abstract

A method is described for proving "always possibly" properties of specifications in formalisms with linear-time trace semantics. It is shown to be relatively complete for TLA (Temporal Logic of Actions) specifications.

Key words: Branching time, linear time, temporal logic.

1 Introduction

Does proving possibility properties provide any useful information about a system? Why prove that it is possible for a user to press q on the keyboard and for a q subsequently to appear on the screen? We know that the user can

Proving Possibility Properties

Although possibility properties may tell us nothing about a system, we do not reason about a system; we reason about a mathematical model of a system. A possibility property can provide a sanity check on our model. Proving that it is always possible for a $press(q)$ action to occur tells us something useful about the model. In general, we want to prove that a model allows the occurrence of actions representing events that the system cannot prevent.

Proving Possibility Properties

Although possibility properties may tell us nothing about a system, we do not reason about a system; we reason about a mathematical model of a system. A possibility property can provide a sanity check on our model. Proving that it is always possible for a *press*(q) action to occur tells us something useful about the model. In general, we want to prove that a model allows the occurrence of actions representing events that the system cannot prevent.

To prove $\mathbf{P}_{\Pi}(P)$, we find an action M and a conjunction G of fairness properties such that

$$Init \land \Box[N]_v \land \Diamond\Box[M]_v \land G \Rightarrow \Box\Diamond P \tag{4}$$

Checking Possibility Properties

$$Spec \triangleq Init \land \Box[Next]_{vars} \land Fairness$$

$$\textsc{proposition}\ \ Init \land \Box[Next]_{vars} \land \Diamond\Box[M]_{vars} \land G \Rightarrow \Box\Diamond P$$

Checking Possibility Properties

$$Init \land \Box[Next]_{vars} \land \Diamond\Box[M]_{vars} \land G$$

Checking Possibility Properties

$$Init \land \Box[Next]_{vars} \land \Diamond\Box[M]_{vars} \land G$$

$\text{VARIABLE } t$

$$\begin{aligned}&\land\ Init \land \Box[Next]_{vars}\\&\land\ t = \text{FALSE} \land \Box[t \Rightarrow (M \land \text{UNCHANGED } t)]_{\langle vars,\, t\rangle} \land \text{WF}_{\langle vars,\, t\rangle}(t' = \text{TRUE}) \land G\end{aligned}$$

$$\Diamond\Box[M]_{vars} \equiv \Box[t \Rightarrow (M \land \text{UNCHANGED } t)]_{\langle vars,\, t\rangle} \land \text{WF}_{\langle vars,\, t\rangle}(t' = \text{TRUE})$$

Checking Possibility Properties

$$\text{VARIABLE } t$$

$$Trigger \triangleq t = \text{FALSE} \land t' = \text{TRUE}$$

$$PossibilitySpec \triangleq$$

$$\begin{aligned}
&\land Init \land t = \text{FALSE} \\
&\land \Box[\lor \land Next \\
&\qquad\quad \land t \Rightarrow M \\
&\qquad\quad \land \text{UNCHANGED } t \\
&\qquad \lor Trigger \land \text{UNCHANGED } vars]_{\langle vars,\, t\rangle} \\
&\land G \land \text{WF}_{\langle vars,\, t\rangle}(Trigger)
\end{aligned}$$

$$\text{PROPOSITION } PossibilitySpec \Rightarrow \Box\Diamond P$$

Example III

Image: Amazon

Model-Based Trace-Checking

Yvonne Howard et al., UK Software Testing and Research, 2003

Model-Based Trace-Checking

Yvonne Howard, Stefan Gruner, Andrew M Gravell, Carla Ferreira, Juan Carlos Augusto
DSSE, Department of Electronics and Computer Science, University of Southampton
Southampton, SO17 1BJ
Email: ymh@ecs.soton.ac.uk

Abstract

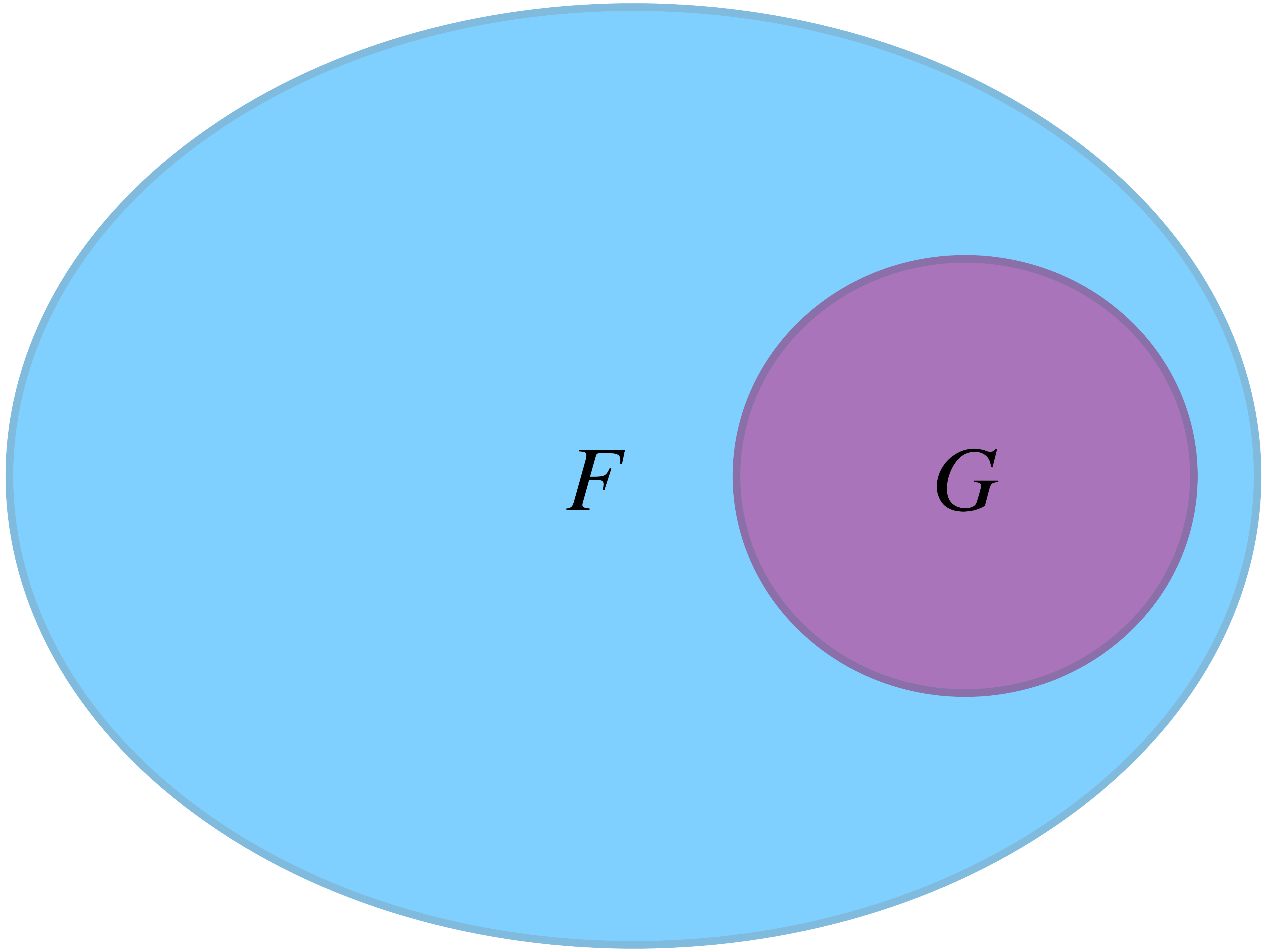
Trace analysis can be a useful way to discover problems in a program under test. Rather than writing a special purpose trace analysis tool, this paper proposes that traces can usefully be analysed by checking them against a formal model using a standard model-checker or else an animator for executable specifications. These techniques are illustrated using a Travel Agent case study implemented in J2EE. We added trace beans to this code that write trace information to a database. The traces are then extracted and converted into a form suitable for analysis by Spin, a popular model-checker, and Pro-B, a model-checker and animator for the B notation. This illustrates the technique, and also the fact that such a system can have a variety of models, in different notations, that capture different features. These experiments have demonstrated that model-based trace-checking is feasible. Future work is focussed on scaling up the approach to larger systems by increasing the level of automation.

1 Introduction

From the tester's perspective, tracing is perhaps considered a last resort. When a program or system crashes, it may be necessary to analyse a trace recorded in a log file which can be tedious. A trace viewer [Helmbold90] can help with this task, but human intuition is still

Conformance in TLA

Implication is refinement/implementation



$$G \Rightarrow F$$

Model-Based Trace-Checking in TLA⁺

MODULE *System*

VARIABLES $x,\ y,\ z,\ tickTock$

$vars \triangleq \langle x,\ y,\ z,\ tickTock \rangle$

$$TypeOK \triangleq \land\ x \in Nat \\ \land\ y \in Nat \\ \land\ z \in Nat \\ \land\ tickTock \in \{\text{"tick"},\ \text{"tock"}\}$$

$$Init \triangleq \land\ x \in 0\,..\,9 \\ \land\ y \in 0\,..\,9 \\ \land\ z = 0 \\ \land\ tickTock = \text{"tick"}$$

$$Next \triangleq \lor \land\ tickTock = \text{"tick"} \\ \land\ tickTock' = \text{"tock"} \\ \land\ z' = x + y \\ \land\ \text{UNCHANGED}\ \langle x,\ y \rangle \\ \lor \land\ tickTock = \text{"tock"} \\ \land\ tickTock' = \text{"tick"} \\ \land\ x' \in 0\,..\,9 \\ \land\ y' \in 0\,..\,9 \\ \land\ \text{UNCHANGED}\ z$$

$Safety \triangleq Init \land \Box[Next]_{vars}$ Just the safety part of the spec

$Spec \triangleq Safety \land \text{WF}_{vars}(Next)$

Statistics

State space progress (click column header for graph)

Time	Diameter	States Found	Distinct States
00:00:03	3	12,000	2,000
00:00:02	0	100	100

Model-Based Trace-Checking in TLA$^+$

Tuples are: $\langle x,\ y,\ z,\ tickTock \rangle$

$$
\begin{aligned}
Trace \triangleq\ & \langle \langle 1, 0, 0, \text{“tick”} \rangle, \langle 1, 0, 1, \text{“tock”} \rangle, \langle 1, 1, 1, \text{“tick”} \rangle, \langle 1, 1, 2, \text{“tock”} \rangle, \\
& \langle 0, 3, 2, \text{“tick”} \rangle, \langle 0, 3, 3, \text{“tock”} \rangle, \langle 2, 2, 3, \text{“tick”} \rangle, \langle 2, 2, 4, \text{“tock”} \rangle, \\
& \langle 3, 2, 4, \text{“tick”} \rangle, \langle 3, 2, 5, \text{“tock”} \rangle, \langle 2, 4, 5, \text{“tick”} \rangle, \langle 2, 4, 6, \text{“tock”} \rangle, \\
& \langle 5, 2, 6, \text{“tick”} \rangle, \langle 5, 2, 7, \text{“tock”} \rangle, \langle 4, 4, 7, \text{“tick”} \rangle, \langle 4, 4, 8, \text{“tock”} \rangle, \\
& \langle 2, 7, 8, \text{“tick”} \rangle, \langle 2, 7, 9, \text{“tock”} \rangle, \langle 6, 4, 9, \text{“tick”} \rangle, \langle 6, 4, 10, \text{“tock”} \rangle \rangle
\end{aligned}
$$

Model-Based Trace-Checking in TLA+

$$\text{VARIABLES } x,\ y,\ z,\ tickTock$$
$$Model \triangleq \text{INSTANCE } System$$

$$\text{VARIABLE } i \quad \text{the trace index}$$
$$Read \triangleq Model!vars = Trace[i]$$
$$Init \triangleq i = 1 \land Read$$
$$Next \triangleq i < Len(Trace) \land i' = i + 1 \land Read'$$
$$TraceBehavior \triangleq Init \land \Box[Next]_{\langle Model!vars,\ i \rangle}$$

Model-Based Trace-Checking in TLA⁺

$\text{VARIABLES } x,\ y,\ z,\ tickTock$
$Model \triangleq \text{INSTANCE } System$
$vars \triangleq \langle x,\ y,\ z,\ tickTock \rangle$ If we write $Model!vars$, TLC complains.

$\text{VARIABLE } i$ the trace index

"Reading" a record is just $vars = Trace[i]$, but unfortunately TLC isn't happy with that, so:
$Read \triangleq \text{LET } Rec \triangleq Trace[i] \text{ IN } x = Rec[1] \land y = Rec[2] \land z = Rec[3] \land tickTock = Rec[4]$

Unfortunately, TLC also isn't happy with just $Read'$ – which is equivalent to:
$ReadNext \triangleq \text{LET } Rec \triangleq Trace[i'] \text{IN } x' = Rec[1] \land y' = Rec[2] \land z' = Rec[3] \land tickTock' = Rec[4]$

$Init \triangleq i = 1 \land Read$
$Next \triangleq i < Len(Trace) \land i' = i + 1 \land ReadNext$

$TraceBehavior \triangleq Init \land \Box[Next]_{\langle vars,\ i \rangle}$

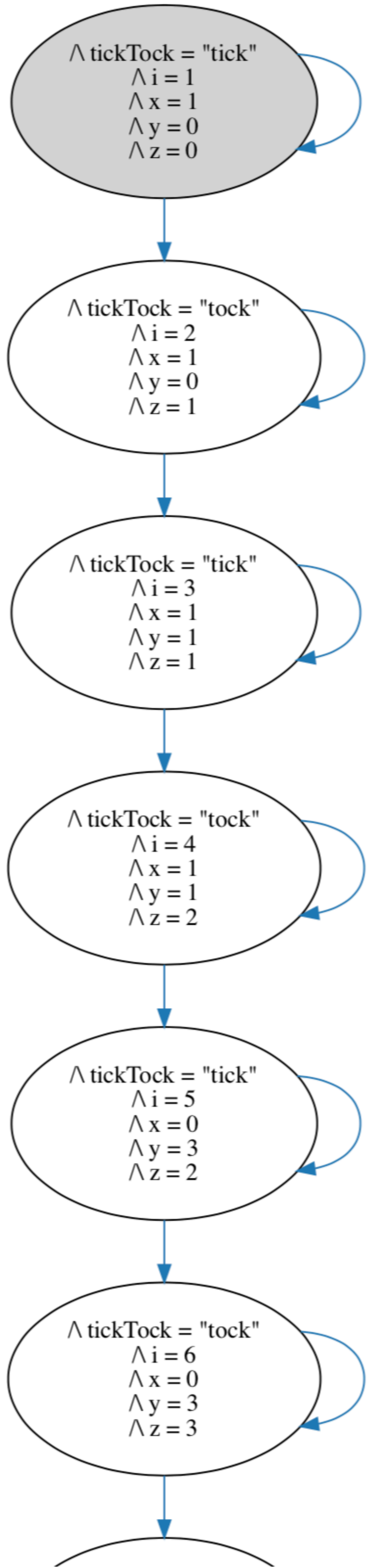
Model-Based Trace-Checking in TLA+

$$TraceBehavior \Rightarrow Model!Safety$$

Statistics

State space progress (click column header for graph)

Time	Diameter	States Found	Distinct States
00:00:02	20	40	20
00:00:02	0	1	1



Model-Based Trace-Checking in TLA+

$$
\begin{aligned}
Traces \triangleq [&log1 \mapsto \\
&\langle\langle 1, 0, 0, \text{“tick”}\rangle, \langle 1, 0, 1, \text{“tock”}\rangle, \langle 1, 1, 1, \text{“tick”}\rangle, \langle 1, 1, 2, \text{“tock”}\rangle, \\
&\ \langle 0, 3, 2, \text{“tick”}\rangle, \langle 0, 3, 3, \text{“tock”}\rangle, \langle 2, 2, 3, \text{“tick”}\rangle, \langle 2, 2, 4, \text{“tock”}\rangle, \\
&\ \langle 3, 2, 4, \text{“tick”}\rangle, \langle 3, 2, 5, \text{“tock”}\rangle, \langle 2, 4, 5, \text{“tick”}\rangle, \langle 2, 4, 6, \text{“tock”}\rangle\rangle, \\
&log2 \mapsto \\
&\langle\langle 5, 2, 0, \text{“tick”}\rangle, \langle 5, 2, 7, \text{“tock”}\rangle, \langle 4, 4, 7, \text{“tick”}\rangle, \langle 4, 4, 8, \text{“tock”}\rangle, \\
&\ \langle 2, 7, 8, \text{“tick”}\rangle, \langle 2, 7, 9, \text{“tock”}\rangle, \langle 6, 4, 9, \text{“tick”}\rangle, \langle 6, 4, 10, \text{“tock”}\rangle\rangle, \\
&log3 \mapsto \\
&\langle\langle 3, 4, 0, \text{“tick”}\rangle, \langle 3, 4, 7, \text{“tock”}\rangle, \langle 0, 9, 7, \text{“tick”}\rangle, \langle 0, 9, 9, \text{“tock”}\rangle, \\
&\ \langle 2, 2, 9, \text{“tick”}\rangle, \langle 2, 2, 4, \text{“tock”}\rangle, \langle 2, 6, 4, \text{“tick”}\rangle, \langle 2, 6, 8, \text{“tock”}\rangle\rangle]
\end{aligned}
$$

Model-Based Trace-Checking in TLA+

$\text{VARIABLES } x, y, z, tickTock$
$Model \triangleq \text{INSTANCE } System$
$vars \triangleq \langle x, y, z, tickTock \rangle$

$\text{VARIABLE } log,$ the *log* file
$\quad i$ the trace index

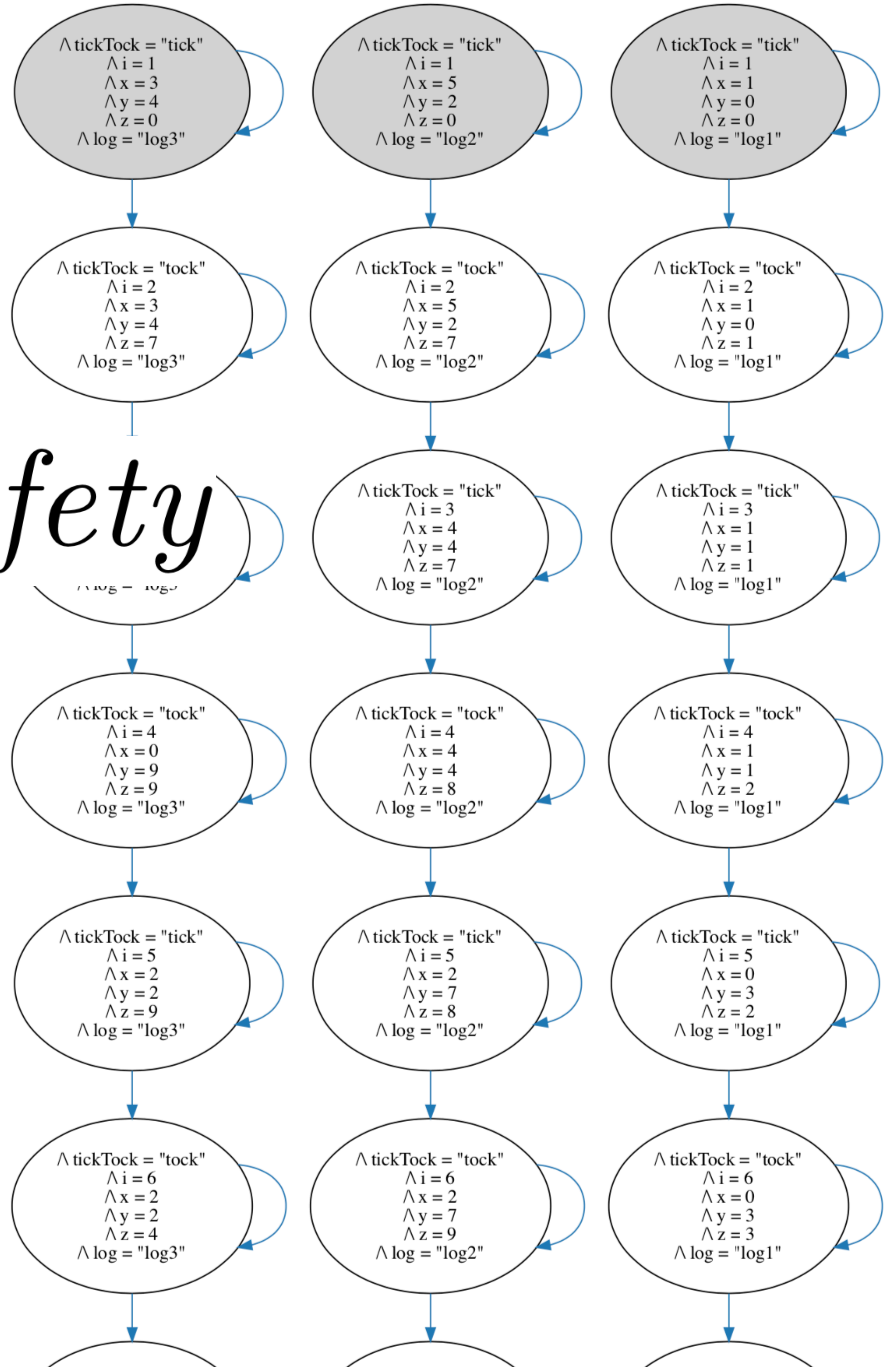
$Trace \triangleq Traces[log]$
$Read \triangleq \text{LET } Rec \triangleq Trace[i] \text{ IN } x = Rec[1] \land y = Rec[2] \land z = Rec[3] \land tickTock = Rec[4]$
$ReadNext \triangleq \text{LET } Rec \triangleq Trace[i'] \text{ IN } x' = Rec[1] \land y' = Rec[2] \land z' = Rec[3] \land tickTock' = Rec[4]$

$Init \triangleq log \in \text{DOMAIN } Traces \land i = 1 \land Read$
$Next \triangleq \land i < Len(Trace) \land i' = i + 1 \land ReadNext$
$\quad \land \text{UNCHANGED } log$ Each trace follows a single log

$TraceBehavior \triangleq Init \land \Box[Next]_{\langle log, i, vars \rangle}$

Model-Based Trace-Checking in TLA+

$$TraceBehavior \Rightarrow Model!Safety$$



Model-Based Trace-Checking in TLA+

$$Trace \triangleq \langle 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 \rangle$$

Model-Based Trace-Checking in TLA⁺

$$Trace \triangleq \langle 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 \rangle$$

$$TraceBehavior \Rightarrow \mathbf{\exists}\, x,\ y,\ tickTock : Model!Safety$$

Model-Based Trace-Checking in TLA$^+$

$$\textsc{variables}\ z,\ i$$
$$tvars \triangleq \langle z,\ i \rangle$$

$$
\begin{aligned}
&Read &&\triangleq \textsc{let}\ Rec \triangleq Trace[i]\ \textsc{in}\ \ z = Rec\\
&ReadNext &&\triangleq \textsc{let}\ Rec \triangleq Trace[i']\ \textsc{in}\ \ z' = Rec\\
&InitTrace &&\triangleq i = 1 \land Read\\
&NextTrace &&\triangleq i < Len(Trace) \land i' = i + 1 \land ReadNext
\end{aligned}
$$

Model-Based Trace-Checking in TLA+

$$\text{VARIABLES } z,\ i$$
$$tvars \triangleq \langle z,\ i \rangle$$

$$\begin{aligned}
Read &\triangleq \text{LET } Rec \triangleq Trace[i] \text{ IN } z = Rec \\
ReadNext &\triangleq \text{LET } Rec \triangleq Trace[i'] \text{ IN } z' = Rec \\
InitTrace &\triangleq i = 1 \land Read \\
NextTrace &\triangleq i < Len(Trace) \land i' = i + 1 \land ReadNext
\end{aligned}$$

$$\text{VARIABLE } tt$$

$$\begin{aligned}
Init &\triangleq InitTrace \land tt = 0 \\
Next &\triangleq \lor \land i < Len(Trace) \\
&\qquad\ \land tt' = 1 - tt \\
&\qquad\ \land \lor tt = 0 \land NextTrace \\
&\qquad\quad\ \ \lor tt = 1 \land \text{UNCHANGED } tvars \\
&\quad\ \ \lor \text{UNCHANGED } \langle tt,\ tvars \rangle
\end{aligned}$$

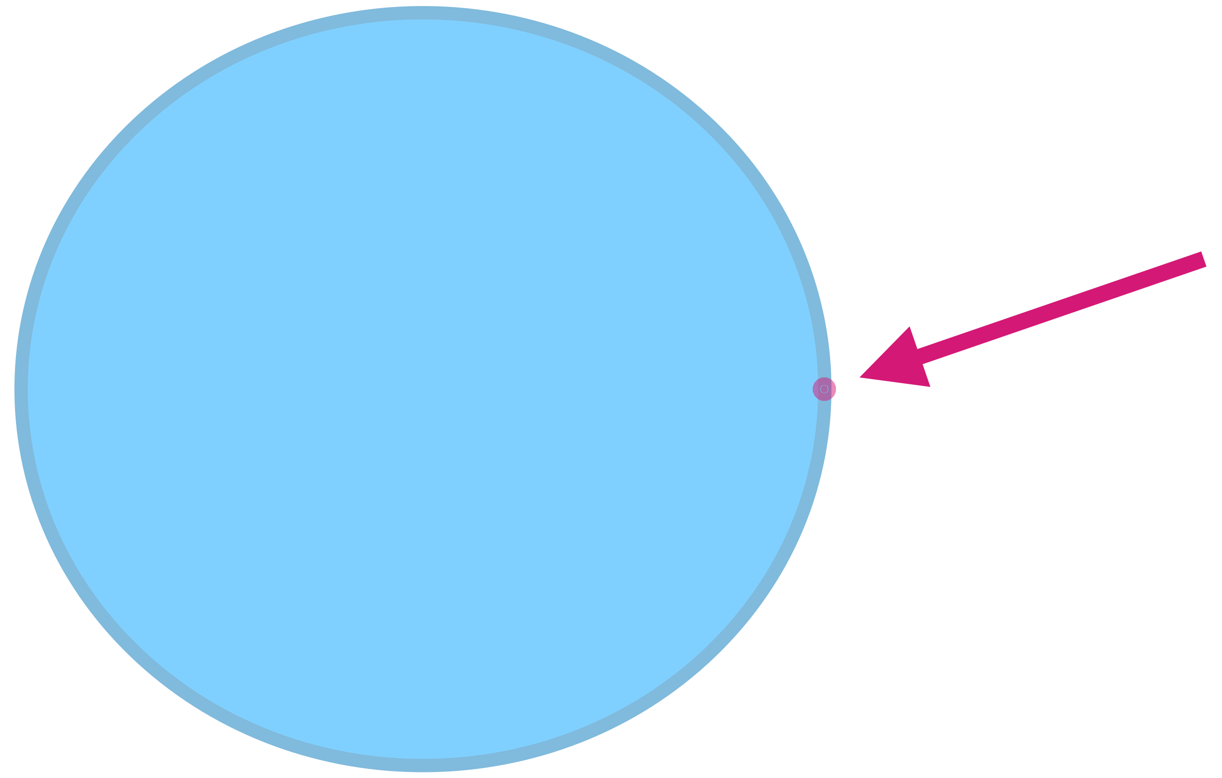
So that we don't get a deadlock error in TLC

$$TraceBehavior \triangleq Init \land \Box[Next]_{\langle tt,\ z,\ i \rangle}$$

$$\begin{aligned}
Model \triangleq \text{INSTANCE } System \text{ WITH } & tickTock \leftarrow \text{IF } tt = 0 \text{ THEN } \text{“tick”} \text{ ELSE } \text{“tock”}, \\
& x \leftarrow \text{IF } tt = 0 \text{ THEN } z \text{ ELSE } z - 1, \\
& y \leftarrow 1
\end{aligned}$$

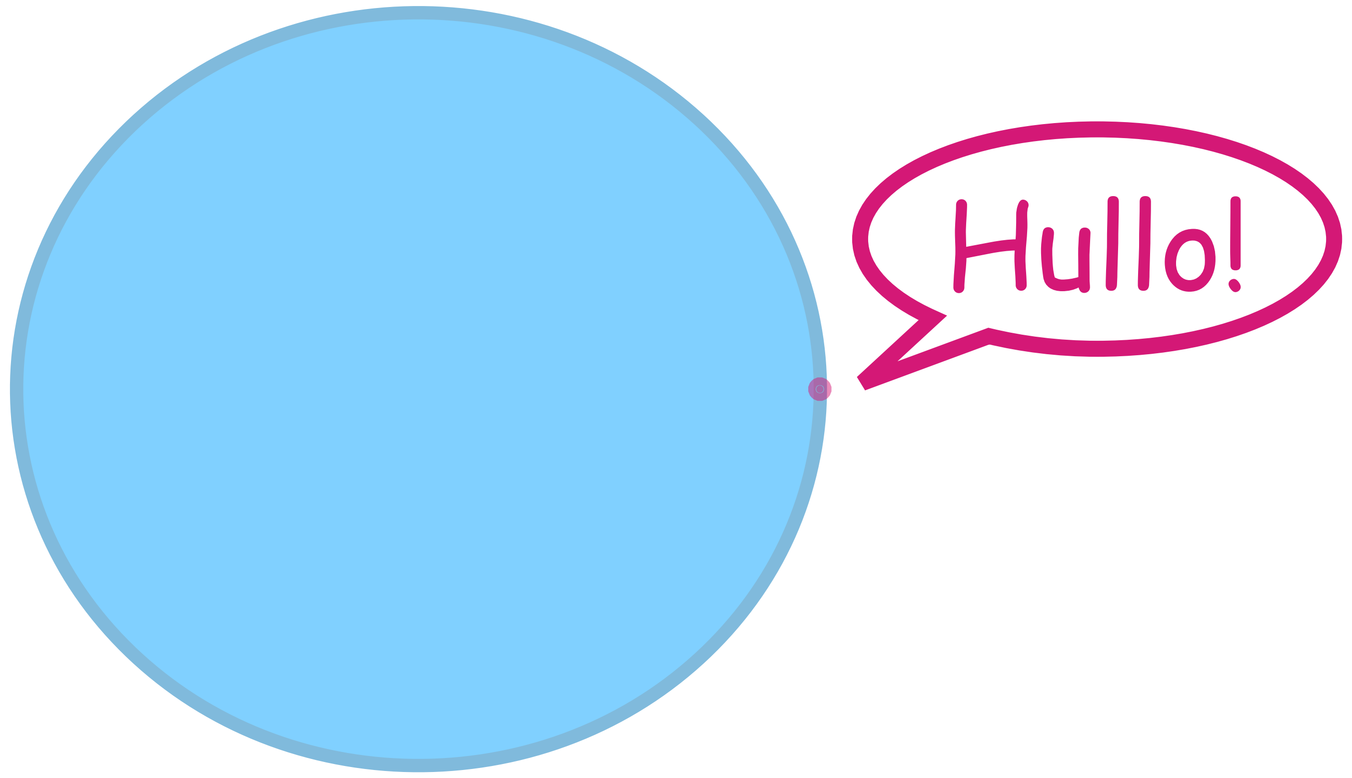
Model-Based Trace-Checking in TLA⁺

$$TraceBehavior \Rightarrow \exists\, x,\ y,\ tickTock : Model!Safety$$



Model-Based Trace-Checking in TLA$^+$

$$TraceBehavior \Rightarrow \exists\, x,\ y,\ tickTock : Model!Safety$$



NOT A THEOREM $Model!Safety \land TraceBehavior \equiv \text{FALSE}$

NOT A THEOREM $Model!Safety \land TraceBehavior \Rightarrow \text{FALSE}$

Model-Based Trace-Checking in TLA+

$$vars \triangleq \langle x,\ y,\ z,\ tickTock \rangle$$
$$tvars \triangleq \langle z,\ i \rangle$$

$$Model \triangleq \text{INSTANCE } System$$

$$ComposedSpec \triangleq \quad Model!Safety \land TraceBehavior \equiv$$
$$\land\ Model!Init \land InitTrace$$
$$\land\ \Box[Compose(Model!Next,\ vars,\ NextTrace,\ tvars)]_{\langle vars,\ tvars \rangle}$$

Model-Based Trace-Checking in TLA+

$$TraceFinished \triangleq i \geq Len(Trace)$$

$$Check \triangleq ComposedSpec \Rightarrow \Box(\neg TraceFinished)$$

Model-Based Trace-Checking in TLA+

$$TraceFinished \triangleq i \geq Len(Trace)$$

$$Check \triangleq ComposedSpec \Rightarrow \Box(\neg TraceFinished)$$

Statistics

State space progress (click column header for graph)

Time	Diameter	States Found	Distinct States
00:00:02	20	5,704	1,057
00:00:02	0	100	100

Statistics

State space progress (click column header for graph)

Time	Diameter	States Found	Distinct States
00:00:03	3	12,000	2,000
00:00:02	0	100	100

Invariant ~TraceFinished is violated.

Error-Trace Exploration

Error-Trace

Name	Value
<Initial predicate>	State (num = 1)
i	1
tickTock	"tick"
x	0
y	1
z	0
<Action line 377, col 1 t...	State (num = 2)
i	2
tickTock	"tock"
x	0
y	1
z	1
<Action line 377, col 1 t...	State (num = 3)
i	2
tickTock	"tick"
x	0
y	2
z	1
<Action line 377, col 1 t...	State (num = 4)
i	3
tickTock	"tock"
x	0
y	2
z	2
<Action line 377, col 1 t...	State (num = 5)
i	3
tickTock	"tick"
x	0
y	3
z	2
<Action line 377, col 1 t...	State (num = 6)
i	4
tickTock	"tock"
x	0
y	3
z	3
<Action line 377, col 1 t...	State (num = 7)
i	4
tickTock	"tick"

Model-Based Trace-Checking in TLA⁺

Resources

https://pron.github.io/files/Trace.pdf

https://github.com/tlaplus/CommunityModules

https://github.com/lemmy/BlockingQueue

Conjunction Capers

A TLA$^+$ Truffle

Image: Amazon