Adam Sorrenti, #500903848
Department of Computer Science
Toronto Metropolitan University

July 25, 2024

CP8314 GRADUATE STUDENT PROJECT — New Evaluations

New Evaluation 1: Exam Question – Temporal Fluent

Goal of the Exam Question

The goal of this exam question is to assess students' abilities in applying temporal situation calculus to model a smart home heating system. The task is aimed at evaluating students' skills in defining dynamic behaviors and interactions within a system through logical formulations. This includes crafting precondition axioms, successor state axioms, and state evolution axioms to manage multiple, interacting components such as heating and ventilation. The exam question also tests the integration of physical and practical constraints, ensuring realistic and applicable system models. By synthesizing knowledge from various disciplines including logic and physics, the question prepares students for real-world applications in systems design.

Steps to Solution

To effectively tackle the exam question, students should follow these steps:

1. **Identify Key Components:** Understand and list the fluents (such as `temp`, `heating`, and `venting`) and actions (including `turnOnHeater`, `turnOffHeater`, `openVent`, `closeVent`).
2. **Formulate Precondition Axioms:** Define the conditions under which each action can be performed. Consider the current states of fluents and other relevant conditions.
3. **Define Successor State Axioms:** Determine how each action affects the fluents' states. Create logical expressions that describe the transition of states post-action.
4. **Establish Initial State Axioms:** Specify how the fluents are initialized or maintained when actions occur. Ensure that these axioms correctly reflect the physical constraints and initial conditions.
5. **Develop State Evolution Axioms:** Write expression that describe how the system's state evolves over time given the actions taken and the system's dynamics, including thermal properties and ambient conditions.
6. **Apply Physical Constraints:** Ensure that all temperature calculations respect the absolute zero constraint and other mentioned constraints.

Question:

Consider a smart home heating system designed to regulate the temperature of a house automatically. The system is meant to maintain indoor comfort based on the external temperature fluctuations and user-set temperature limits. Your task is to formalize the operation of this system using temporal situation calculus. Consider all temperatures in Celsius.

The heat output can be controlled based on predefined temperature thresholds with actions $turnOnHeater(t)$ and $turnOffHeater(t)$. These action instantaneously change the fluent $heating(s)$ process. Let the temporal functional fluent $temp(t, s)$ represent the temperature of the home at time t. The system allows for a constant heat output H_{out} (per unit time) when the heater is on and a constant heat decrease rate H_{dec} (per unit time), influenced (multiplicatively) by the home's thermal mass M and constant ambient temperature T_{amb}. The system is controlled to only change depending on a minimum T_{min} and a maximum T_{max} (i.e., you can only start heating if below T_{min}, or stop heating if above T_{max}, this is inversely true for venting) while also respecting the physical laws of thermodynamics, such as not allowing the temperature to drop below absolute zero (-273.15). Additionally, the system features a ventilation system which can cool the house more rapidly if needed. With actions $openVent(t)$ and $closeVent(t)$ the fluent $venting(s)$ process is instantaneously changed. When venting, the rate of H_{dec} is doubled. Assume the temperature gain while heating is always more than ambient heat loss in the system. If the system is venting and heating, the temperature does not change.

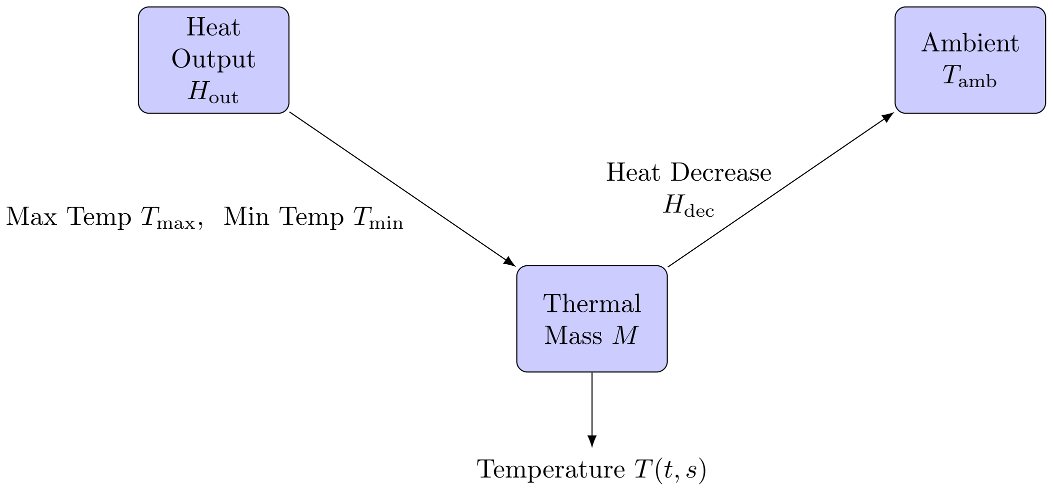
Heat Output H_{out}
Ambient T_{amb}
Heat Decrease H_{dec}
Max Temp $T_{\max}$, Min Temp $T_{\min}$
Thermal Mass M
Temperature $T(t,s)$

Figure 1: Diagram of the Smart Home Heating System

Grading rubric:

(10%) Define the **preconditions for the actions** $turnOnHeater(t)$, $turnOffHeater(t)$, $openVent(t)$, $closeVent(t)$. Consider under what conditions these actions are possible based on the system's current state and the time of action.

(10%) Define the **Successor State Axioms** for the fluents $heating(s)$ and $venting(s)$ process. These axioms should account for the changes in the state of these fluents based on the actions taken.

(30%) Specify how the **temporal fluent** *temp* **should be initialized** following any action. Assume that there are no leaks in the system, meaning that temperature changes only occur due to the actions specified and the natural processes modeled.

(50%) Develop a State Evolution Axiom for $temp(t,s)$ that considers whether the heating or ventilation systems are active, and includes conditions for their influence on temperature over time. Also, incorporate the physical constraint on temperature.

Instructions:

Your answers should be logical formulas that clearly define how the smart home heating system operates according to the given constraints and actions. This exercise tests your understanding of situation calculus in designing complex system controls with multiple interacting components.

Solution:

Precondition Axioms with Temporal Constraints:

$$\begin{aligned}
Poss(turnOnHeater(t),s) &\leftrightarrow \neg heating(s) \land t \geq start(s) \land (temp(t,s) < T_{\min}).\\
Poss(turnOffHeater(t),s) &\leftrightarrow heating(s) \land t \geq start(s) \land (temp(t,s) > T_{\max}).\\
Poss(openVent(t),s) &\leftrightarrow \neg venting(s) \land t \geq start(s) \land (temp(t,s) > T_{\max}).\\
Poss(closeVent(t),s) &\leftrightarrow venting(s) \land t \geq start(s) \land (temp(t,s) < T_{\min}).
\end{aligned}$$

SSAs for Atemporal Fluents:

$$\begin{aligned}
heating(do(a,s)) \leftrightarrow\ & \exists t(a = turnOnHeater(t) \land \neg heating(s) \land temp(time(a),s) < T_{\min}) \lor \\
& (heating(s) \land \neg\exists t(a = turnOffHeater(t) \land temp(time(a),s) > T_{\max})). \\
venting(do(a,s)) \leftrightarrow\ & \exists t(a = openVent(t) \land \neg venting(s) \land temp(time(a),s) > T_{\max}) \lor \\
& (venting(s) \land \neg\exists t(a = closeVent(t) \land temp(time(a),s) < T_{\min})).
\end{aligned}$$

SSAs to Initialize Temporal Fluents

Initial value SSA for temperature:

$$temp_{init}(do(a,s)) = T \leftrightarrow temp(time(a),s) = T.$$

State Evolution Axiom (SEA) for Temperature

$$\begin{aligned}
temp(t,s) = T \leftrightarrow \exists T_0 \exists t_0 \Big(& temp_{init}(s) = T_0 \land start(s) = t_0 \land \\
& \Big((heating(s) \land \neg venting(s) \land T = T_0 + H_{\text{out}} \cdot (t - t_0) - H_{\text{dec}} \cdot M \cdot T_{\text{amb}} \cdot (t - t_0)) \lor \\
& (\neg heating(s) \land venting(s) \land T = max\{T_0 - 2 \cdot H_{\text{dec}} \cdot M \cdot T_{\text{amb}} \cdot (t - t_0), -273.15\}) \lor \\
& (heating(s) \land venting(s) \land T = T_0) \lor \\
& (\neg heating(s) \land \neg venting(s) \land T = max\{T_0 - H_{\text{dec}} \cdot M \cdot T_{\text{amb}} \cdot (t - t_0), -273.15\})\Big)
\end{aligned}$$

New Evaluation 2: Assignment – Forgetting and Progression

Goal of the Assignment

The primary goal of this assignment is to apply progression to a new domain within a software management system.

Steps to Solution

To arrive at the solution for this assignment, students should:

1. Model the system by defining the initial state and the fluents (conditions) involved.
2. Analyze the preconditions and effects of the `updateSoftware` action within the system.
3. Apply Successor State Axioms (SSAs) to derive the state of the system post-action execution.
4. Identify which fluents are affected locally and globally by the action to understand its scope and impact.
5. Transform the SSAs to reflect changes based on the specific actions taken and the initial system state.
6. Define and utilize argument and characteristic sets for each relevant fluent to effectively model the system's progression.
7. Implement the technique of forgetting and progression to multiple fluents.

Assignment:

Considers a computer system with various software components that can be updated. We address fluents that indicate the situation in terms of updates, vulnerabilities, and restart requirements.

Fluents

- $\texttt{updated}(component, s)$: True if a software component is updated in situation s.
- $\texttt{vulnerable}(component, s)$: True if a software component is vulnerable in situation s.
- $\texttt{requiresRestart}(system, s)$: True if the system requires a restart to apply updates in situation s.

The initial state S_0 is defined as follows:

$$\texttt{updated}(os, S_0) \land \neg\texttt{updated}(av, S_0) \land \texttt{vulnerable}(av, S_0) \land \neg\texttt{requiresRestart}(sys, S_0)$$

Implicit types `software(os)` and `software(av)` are assumed for the operating system (os) and antivirus (av).

Consider the following action:
updateSoftware(*component*): Action to update a specific software component.

- **Precondition:**
$$\texttt{poss}(\texttt{do}(\texttt{updateSoftware}(c), s)) \leftrightarrow \texttt{software}(c) \land \texttt{vulnerable}(c, s)$$
- **Effect:**
$$\texttt{updated}(c, \texttt{do}(\texttt{updateSoftware}(c), s))$$
$$\texttt{requiresRestart}(sys, \texttt{do}(\texttt{updateSoftware}(c), s))$$

Successor State Axioms (SSAs)

$$\begin{aligned}
\texttt{updated}(c, \texttt{do}(a, s)) &\leftrightarrow \texttt{software}(c) \land \texttt{vulnerable}(c, s) \land a = \texttt{updateSoftware}(c) \lor \texttt{updated}(c, s) \\
&\quad \land \neg(a = \texttt{injectVirus}(c) \lor (\exists sys).a = \texttt{restart}(sys)) \\
\texttt{vulnerable}(c, \texttt{do}(a, s)) &\leftrightarrow a = \texttt{injectVirus(c)} \lor \texttt{vulnerable}(c, s) \land \neg(a = \texttt{updateSoftware}(c)) \\
\texttt{requiresRestart}(sys, \texttt{do}(a, s)) &\leftrightarrow (\exists c).a = \texttt{updateSoftware}(c) \lor \texttt{requiresRestart}(sys, s) \land \neg(a = \texttt{restart}(sys))
\end{aligned}$$

Suppose the next situation is $S_\alpha = \texttt{do}(\texttt{updateSoftware}(av), S_0)$. We would like to compute progression of the initial theory.

Assignment questions:

1. **(5%)** Identify which fluents are local effect wrt. updateSoftware(c).
2. **(10%)** Derive transformed SSAs.
3. **(15%)** Define the argument set Δ_{Fj} for each relevant fluent F_j. Generate the characteristic set Ω of α, yeilding $\Omega(S_0)$.
4. **(30%)** Instantiate SSA of F_j on LHS with constants from ΔF_j, combine SSAs yeilding $D_{ss}[\Omega]$ representing new values of the fluents.
5. **(40%)** Forgetting and Progression: Let ϕ be $D_{S_0} \land D_{ss}[\Omega]$. The progression D_{S_α} is forget$(\phi, \Omega(S_0))$, then replace S_0 with S_α.

Let $\top$ and $\bot$ be a new propositional constant as an abbreviation for $(S_0 = S_0)$ and $\neg(S_0 = S_0)$, True and False respectively.

Solution:

Q1: Identify which fluents are local effect wrt. updateSoftware(c).

Local effect:

- updated(c,s) – updateSoftware(c) has local effect on fluent updated since all object arguments are included.
- vulnerable(c,s) – updateSoftware(c) has local effect on fluent vulnerable since all object arguments are included.

Global effect:

- requiresRestart(sys,s) – updateSoftware(c) has global effect on fluent requiresRestart since the action has at least one universally-quantified object argument not included.

Q2: Derive transformed SSAs.

Updated SSA transformation:

$$\texttt{updated}(c, \texttt{do}(\texttt{updateSoftware}(av), S_0)) \leftrightarrow$$
$$\texttt{software}(c) \wedge \texttt{vulnerable}(c, S_0) \wedge \texttt{updateSoftware}(av) = \texttt{updateSoftware}(c) \vee \texttt{updated}(c, S_0)$$
$$\wedge \neg(\texttt{updateSoftware}(av) = \texttt{injectVirus}(c) \vee (\exists sys).\texttt{updateSoftware}(av) = \texttt{restart}(sys))$$

$$\equiv \texttt{updated}(c, \texttt{do}(\texttt{updateSoftware}(av), S_0)) \leftrightarrow \texttt{software}(c) \wedge \texttt{vulnerable}(c, S_0) \wedge av = c \vee \texttt{updated}(c, S_0) \wedge \neg(\bot \vee \bot) \quad \text{(UNA \& Simplify)}$$
$$\equiv \texttt{updated}(c, \texttt{do}(\texttt{updateSoftware}(av), S_0)) \leftrightarrow \texttt{software}(c) \wedge \texttt{vulnerable}(c, S_0) \wedge av = c \vee \texttt{updated}(c, S_0) \quad \text{(Simplify)}$$

Vulnerable SSA transformation:

$$\texttt{vulnerable}(c, \texttt{do}(\texttt{updateSoftware}(av), S_0)) \leftrightarrow \texttt{updateSoftware}(av) = \texttt{injectVirus(c)} \vee \texttt{vulnerable}(c, S_0)$$
$$\wedge \neg(\texttt{updateSoftware}(av) = \texttt{updateSoftware}(c))$$

$$\equiv \texttt{vulnerable}(c, \texttt{do}(\texttt{updateSoftware}(av), S_0)) \leftrightarrow \bot \vee \texttt{vulnerable}(c, S_0) \wedge \neg(av = c) \quad \text{(UNA \& Simplify)}$$
$$\equiv \texttt{vulnerable}(c, \texttt{do}(\texttt{updateSoftware}(av), S_0)) \leftrightarrow \texttt{vulnerable}(c, S_0) \wedge \neg(av = c) \quad \text{(Simplify)}$$

requiresRestart SSA transformation:

$$\texttt{requiresRestart}(sys, \texttt{do}(\texttt{updateSoftware}(av), S_0)) \leftrightarrow (\exists c).\texttt{updateSoftware}(av) = \texttt{updateSoftware}(c)$$
$$\vee \texttt{requiresRestart}(sys, S_0) \wedge \neg(\texttt{updateSoftware}(av) = \texttt{restart}(sys))$$

$$\equiv \texttt{requiresRestart}(sys, \texttt{do}(\texttt{updateSoftware}(av), S_0)) \leftrightarrow (\exists c).av = c \vee \texttt{requiresRestart}(sys, S_0) \wedge \neg(\bot) \quad \text{(UNA \& Simplify)}$$
$$\equiv \texttt{requiresRestart}(sys, \texttt{do}(\texttt{updateSoftware}(av), S_0)) \leftrightarrow (\exists c).av = c \vee \texttt{requiresRestart}(sys, S_0) \quad \text{(Simplify)}$$
$$\equiv \texttt{requiresRestart}(sys, \texttt{do}(\texttt{updateSoftware}(av), S_0)) \leftrightarrow \top \vee \texttt{requiresRestart}(sys, S_0) \quad \text{(By BAT)}$$
$$\equiv \texttt{requiresRestart}(sys, \texttt{do}(\texttt{updateSoftware}(av), S_0)) \leftrightarrow \top \quad \text{(Simplify)}$$

Q3: Define the argument set Δ_{Fj} for each relevant fluent F_j. Generate the characteristic set Ω of α.

Argument sets:

$$\Delta_{updated} = \{av\}$$
$$\Delta_{vulnerable} = \{av\}$$

Characteristic set:

$$\Omega(S_0) = \{\texttt{vulnerable}(av, S_0), \texttt{updated}(av, S_0)\}$$

These should be forgotten.

Q4: Instantiate SSA of F_j on LHS with constants from ΔF_j, combine SSAs yeilding $D_{ss}[\Omega]$ representing new fluent values.

Instantiate updated SSA:

$$\begin{aligned}
&\texttt{updated}(av, \texttt{do}(\texttt{updateSoftware}(av), S_0)) \leftrightarrow \texttt{software}(av) \wedge \texttt{vulnerable}(av, S_0) \wedge av = av \vee \texttt{updated}(av, S_0) \\
\equiv&\texttt{updated}(av, \texttt{do}(\texttt{updateSoftware}(av), S_0)) \leftrightarrow \texttt{software}(av) \wedge \texttt{vulnerable}(av, S_0) \vee \texttt{updated}(av, S_0) && \text{(Simplify } \top) \\
\equiv&\texttt{updated}(av, \texttt{do}(\texttt{updateSoftware}(av), S_0)) \leftrightarrow \top \wedge \top \vee \bot && \text{(By } S_0) \\
\equiv&\texttt{updated}(av, \texttt{do}(\texttt{updateSoftware}(av), S_0))
\end{aligned}$$

Instantiate vulnerable SSA:

$$\begin{aligned}
&\texttt{vulnerable}(av, \texttt{do}(\texttt{updateSoftware}(av), S_0)) \leftrightarrow \texttt{vulnerable}(av, S_0) \wedge \neg(av = av) \\
\equiv&\texttt{vulnerable}(av, \texttt{do}(\texttt{updateSoftware}(av), S_0)) \leftrightarrow \texttt{vulnerable}(av, S_0) \wedge \neg(\top) \\
\equiv&\neg\texttt{vulnerable}(av, \texttt{do}(\texttt{updateSoftware}(av), S_0))
\end{aligned}$$

$$D_{ss}[\Omega] = \{\texttt{updated}(av, \texttt{do}(\texttt{updateSoftware}(av), S_0)), \neg\texttt{vulnerable}(av, \texttt{do}(\texttt{updateSoftware}(av), S_0))\}$$

Q5: Forgetting and Progression

Let ϕ be $D_{S_0} \wedge D_{ss}[\Omega]$. The progression D_{S_α} is forget$(\phi, \Omega(S_0))$, then replace S_0 with S_α.

$$\phi \equiv (\texttt{updated}(os, S_0) \wedge \neg\texttt{updated}(av, S_0) \wedge \texttt{vulnerable}(av, S_0) \wedge \neg\texttt{requiresRestart}(sys, S_0)) \vee$$
$$(\texttt{updated}(av, \texttt{do}(\texttt{updateSoftware}(av), S_0)) \wedge \neg\texttt{vulnerable}(av, \texttt{do}(\texttt{updateSoftware}(av), S_0)))$$

The forget replacements for vulnerable and updated fluents:

$$\phi[\texttt{vulnerable}] = x = av \wedge y = S_0 \wedge \texttt{vulnerable}(av, S_0) \vee \neg(x = av \wedge y = S_0) \wedge \texttt{vulnerable}(x, y)$$

$$\phi[\texttt{updated}] = x = av \wedge y = S_0 \wedge \texttt{updated}(av, S_0) \vee \neg(x = av \wedge y = S_0) \wedge \texttt{updated}(x, y)$$

Forgetting two different fluents sequentially.

Forget vulnerable first:

$$\texttt{forget}(\phi, \texttt{vulnerable}) = \phi^+ \vee \phi^-$$

$\phi^+ \equiv (\texttt{updated}(os, S_0) \wedge \neg\texttt{updated}(av, S_0) \wedge (av = av \wedge S_0 = S_0 \wedge \texttt{vulnerable}(av, S_0) \vee \neg(av = av \wedge S_0 = S_0) \wedge$ $\texttt{vulnerable}(av, S_0)) \wedge \neg\texttt{requiresRestart}(sys, S_0)) \vee (\texttt{updated}(av, \texttt{do}(\texttt{updateSoftware}(av), S_0)) \wedge \neg(av = av \wedge$ $\texttt{do}(\texttt{updateSoftware}(av), S_0) = S_0 \wedge \texttt{vulnerable}(av, S_0) \vee \neg(av = av \wedge \texttt{do}(\texttt{updateSoftware}(av), S_0) = S_0)$ $\wedge\texttt{vulnerable}(av, \texttt{do}(\texttt{updateSoftware}(av), S_0)))$

$\phi^+ \equiv (\texttt{updated}(os, S_0) \wedge \neg\texttt{updated}(av, S_0) \wedge (\top \wedge \top \wedge \texttt{vulnerable}(av, S_0) \vee \neg(\top \wedge \top) \wedge \texttt{vulnerable}(av, S_0)) \wedge$ $\neg\texttt{requiresRestart}(sys, S_0)) \vee (\texttt{updated}(av, \texttt{do}(\texttt{updateSoftware}(av), S_0)) \wedge \neg(\top \wedge \bot \wedge \texttt{vulnerable}(av, S_0) \vee$ $\neg(\top \wedge \bot) \wedge \texttt{vulnerable}(av, \texttt{do}(\texttt{updateSoftware}(av), S_0)))$

$\phi^+ \equiv (\texttt{updated}(os, S_0) \wedge \neg\texttt{updated}(av, S_0) \wedge \neg\texttt{requiresRestart}(sys, S_0) \vee$ $(\texttt{updated}(av, \texttt{do}(\texttt{updateSoftware}(av), S_0)) \wedge \neg\texttt{vulnerable}(av, \texttt{do}(\texttt{updateSoftware}(av), S_0)))$

$\phi^+ \equiv (\texttt{updated}(av, \texttt{do}(\texttt{updateSoftware}(av), S_0)) \wedge \neg\texttt{vulnerable}(av, \texttt{do}(\texttt{updateSoftware}(av), S_0)))$

Now, interpret $\texttt{vulnerable}(av, S_0)$ as false:
$\phi^- \equiv (\texttt{updated}(os, S_0) \wedge \neg\texttt{updated}(av, S_0) \wedge (\texttt{vulnerable}(av, S_0) \wedge \neg\texttt{requiresRestart}(sys, S_0)) \vee$ $(\texttt{updated}(av, \texttt{do}(\texttt{updateSoftware}(av), S_0)) \wedge \neg\texttt{vulnerable}(av, \texttt{do}(\texttt{updateSoftware}(av), S_0)))$

$\phi^- \equiv \texttt{updated}(av, \texttt{do}(\texttt{updateSoftware}(av), S_0)) \wedge \neg\texttt{vulnerable}(av, \texttt{do}(\texttt{updateSoftware}(av), S_0))$

Therefore,
$\texttt{forget}(\phi, \texttt{vulnerable}) =$

$$\texttt{updated}(av, \texttt{do}(\texttt{updateSoftware}(av), S_0)) \wedge \neg\texttt{vulnerable}(av, \texttt{do}(\texttt{updateSoftware}(av), S_0))$$

[LiuLakemeyer2009, Theorem 2.4] Let G be a finite set of ground atoms and ϕ a formula. Then, $\text{forget}(\phi, G)$ and $\bigvee_{A \in \mathcal{M}(G)} \phi^A$ are logically equivalent.
Forgetting about a set G of ground literals amounts to forgetting about each literal one after another, i.e., if $G = \{P_1, \ldots, P_n\}$, then $\text{forget}(\phi, G) = \text{forget}(\ldots \text{forget}(\text{forget}(\phi, P_1), P_2) \ldots, P_n)$.

Therefore by (LiuLakemeyer2009, Theorem 2.4) we now forget update sequentially using $\phi[\texttt{updated}]$:

$\texttt{forget}(\texttt{forget}(\phi, \texttt{vulnerable}), \texttt{update}) \equiv \phi^+ \vee \phi^-$

$\phi^+ \equiv (av = av \wedge \texttt{do}(\texttt{updateSoftware}(av), S_0) = S_0 \wedge \texttt{updated}(av, S_0) \vee$
$\neg(av = av \wedge \texttt{do}(\texttt{updateSoftware}(av), S_0) = S_0) \wedge \texttt{updated}(av, \texttt{do}(\texttt{updateSoftware}(av), S_0))) \wedge$
$\neg\texttt{vulnerable}(av, \texttt{do}(\texttt{updateSoftware}(av), S_0))$

$\phi^+ \equiv (\top \wedge \bot \wedge \top \vee \neg(\top \wedge \bot) \wedge \texttt{updated}(av, \texttt{do}(\texttt{updateSoftware}(av), S_0))) \wedge \neg\texttt{vulnerable}(av, \texttt{do}(\texttt{updateSoftware}(av), S_0))$
$\phi^+ \equiv \texttt{updated}(av, \texttt{do}(\texttt{updateSoftware}(av), S_0)) \wedge \neg\texttt{vulnerable}(av, \texttt{do}(\texttt{updateSoftware}(av), S_0))$

$\phi^- \equiv (av = av \wedge \texttt{do}(\texttt{updateSoftware}(av), S_0) = S_0 \wedge \texttt{updated}(av, S_0) \vee$
$\neg(av = av \wedge \texttt{do}(\texttt{updateSoftware}(av), S_0) = S_0) \wedge \texttt{updated}(av, \texttt{do}(\texttt{updateSoftware}(av), S_0))) \wedge$
$\neg\texttt{vulnerable}(av, \texttt{do}(\texttt{updateSoftware}(av), S_0))$

$\phi^- \equiv (\top \wedge \bot \wedge \bot \vee \neg(\top \wedge \bot) \wedge \texttt{updated}(av, \texttt{do}(\texttt{updateSoftware}(av), S_0))) \wedge \neg\texttt{vulnerable}(av, \texttt{do}(\texttt{updateSoftware}(av), S_0))$

$\phi^- \equiv \texttt{updated}(av, \texttt{do}(\texttt{updateSoftware}(av), S_0)) \wedge \neg\texttt{vulnerable}(av, \texttt{do}(\texttt{updateSoftware}(av), S_0))$

Therefore,
$\texttt{forget}(\texttt{forget}(\phi, \texttt{vulnerable}), \texttt{update}) = \texttt{forget}(\phi, \Omega(S_0)) =$

$$\texttt{updated}(av, \texttt{do}(\texttt{updateSoftware}(av), S_0)) \wedge \neg\texttt{vulnerable}(av, \texttt{do}(\texttt{updateSoftware}(av), S_0))$$

Finally, we replace S_0 with S_α to arrive at D_{S_α}:

$$D_{S_\alpha} \equiv \texttt{updated}(av, \texttt{do}(\texttt{updateSoftware}(av), S_\alpha)) \wedge \neg\texttt{vulnerable}(av, \texttt{do}(\texttt{updateSoftware}(av), S_\alpha))$$

This completes the progression.

References:

Soutchanski, Mikhail. CPS822 Lectures & Notes, 2024, Toronto Metropolitan University, Toronto, ON.

☐ In working on and submitting this evaluation, I pledge to adhere to Policy 60 and the framework for the evaluation as specified by my instructor, and as mentioned in the CPS822 Course Management Form. The work submitted is my own work. I have not consulted with other people. I cited explicitly all external sources, if any.

Submitted by Adam Sorrenti, #500903848 on July 25, 2024.