Research Report:
Formal Methods and "Modeling" HPC software

Alper Altuntas

Thanks to Gustavo Marques and Bill Large

[1] Alper Altuntas, John Baugh. "Hybrid Theorem Proving as a Lightweight Method for Verifying Numerical Software", In Correctness 2018: Second International Workshop on Software Correctness for HPC Applications @ SC18

[2] Alper Altuntas, Ganesh Gopalakrishnan, Mike Lam, Markus Schordan. Panel on "Facilitating the Adoption of Correctness Tools in HPC Applications", In Correctness 2018 @ SC18

Overview

- A lightweight verification approach numerical software
  - Complementary to testing.
  - Based on viewing numerical models as hybrid systems.
  - Interplay of discrete updates and continuous processes.
  - KPP scheme in MOM6+CESM
- "Modeling" numerical software:
  - **formal methods:** mathematically based techniques for the specification and verification of software systems.

Formal Methods in Scientific Computing

- Advantages over testing:
  - Exhaustive and rigorous (albeit high level abstractions).
  - Computationally efficient.
  - Allows for *incremental* modeling.
- Uses in practice: (statement-level)
  - Race conditions, deadlocks, floating-point operations
  - Functional equivalence
- Higher-level use cases may be effective as well:
  - Algorithm correctness
  - Software design and abstraction

> *"Code is a poor medium for exploring abstractions."* (Jackson, 2006)

Hybrid theorem proving for verifying numerical software

A lightweight formal methods approach

Hybrid Theorem Proving

- **Cyber-physical systems** are compositions of:
  1. Continuous evolution — *real world physics*
  2. Discrete behavior — *computer sampling and intervention*

  - Examples: self-driving cars, ATC, robots, etc.
  - Verification tools: hybrid theorem provers, e.g., KeYmaera X

- **Hybrid programs** that model cyber-physical systems:
  1. ODEs model real world physics.
  2. Discrete programs model computer intervention.

Hybrid theorem proving for verifying numerical software

- Based on viewing numerical models as a hybrid system.
  1. **Continuous processes:** differential equations (DEs) solved by the model.
     e.g., evolution of water surface height
  2. **Discrete updates:** often arise from ad-hoc and empirical modeling.
     e.g., a location becoming wet/dry

- DEs are discretized in time and space, yet they may taken to be continuous.
  - to abstract away from numerical methods.
  - to focus on discrete decisions and updates.
    - the oracle problem!

Hybrid theorem proving for verifying numerical software

1-D shallow water equations:

$$\eta' = -\frac{\partial uh}{\partial x}, ... \tag{1}$$

- In an **actual numerical model**, discretize both in time and space:

$$\frac{\eta_i^{n+1} - \eta_i^n}{\Delta t} = \Big((uh)_{i+1}^n - (uh)_{i-1}^n\Big)/(\Delta x), ... \tag{2}$$

- In a **hybrid verification model**, discretize in space only:

$$\eta' = \Big((uh)_{i+1} - (uh)_{i-1}\Big)/(\Delta x), ... \tag{3}$$

where η is water elevation, h is water height, u is velocity.

Hybrid theorem proving for verifying numerical software

Abstract discrete grids:

- Small discrete grids for tractability.
- Non-determinism to represent external states.

Rationale: By the CFL condition, domain of dependence is limited.

Hybrid theorem proving for verifying numerical software

Hybrid model of the 1-D wetting and drying:

	initialConditions() $\rightarrow$	*initial condition*
	$\left[\{\right.$	*execution begins*
over some Δt $\left\{\right.$	$\{\ \eta' = (uh_{i+1} - uh_{i-1})/(\Delta x),..\}$	*continuous evolution*
instantaneous $\left\{\right.$	wettingDrying();	*discrete assignment*
	$\left.\}^*\right]$	*loop or terminate*
	safetyCondition()	*postcondition*

where η is water elevation, h is water height, u is velocity.

Test Case: The Keymaera X Model of the KPP scheme

Application of hybrid theorem proving in Earth System Modeling

Test Case: The Keymaera X Model of the KPP scheme

The KPP scheme (Large et al., 1994)

- The continuous evolution of a scalar quantity λ over a vertical water column:

$$\frac{\partial \overline{\lambda}}{\partial t} = \frac{\partial}{\partial z}(\overline{w'\lambda'} + \overline{w}\,\overline{\lambda}) \tag{4}$$

- The unresolved turbulent flux parameterized as a diffusive process:

$$\overline{w'\lambda'} = -K_\lambda(\frac{\partial \overline{\lambda}}{\partial z} + \gamma_\lambda) \tag{5}$$

- The diffusivity K_λ at depth d within the OBL:

$$K_\lambda = h \cdot w_\lambda(\sigma) \cdot G_\lambda(\sigma) \tag{6}$$

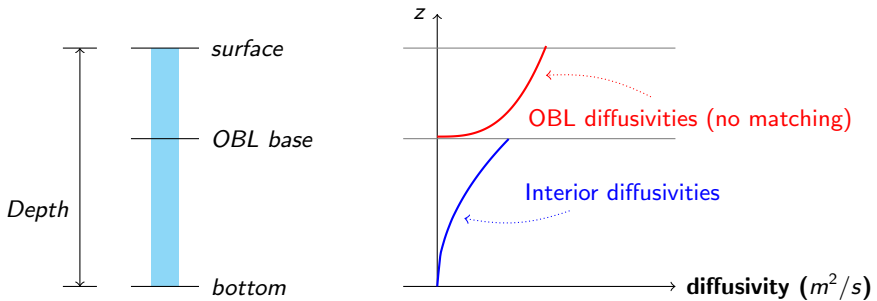
- Shape function for the OBL diffusivities:

$$G_\lambda(\sigma) = a_0 + a_1\sigma + a_2\sigma^2 + a_3\sigma^3 \tag{7}$$

Test Case: The Keymaera X Model of the KPP scheme

The KPP scheme

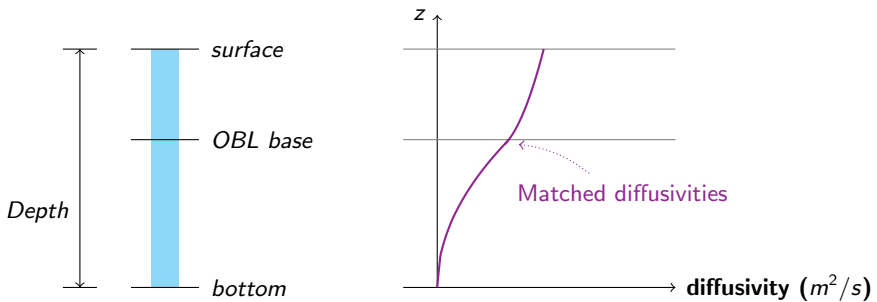
- Compute the ocean boundary layer (OBL) depth.
- Compute the diffusivities within ocean interior.
- Compute the OBL diffusivities (no matching).



Test Case: The Keymaera X Model of the KPP scheme

The KPP scheme

- Compute the ocean boundary layer (OBL) depth.
- Compute the diffusivities within ocean interior.
- Compute the OBL diffusivities **(matching)**.



Test Case: The Keymaera X Model of the KPP scheme

- **Undesired behavior:**
  - Turning on the matching algm leads to negative diffusivities.
- **Debugging:**
  - To pinpoint the source of the issue: inconsistency between matching and interpolation types
- **Fix:**
  - Corrected the matching and interpolation types (for cases when diffusivity gradient at OBL base is negative)
- **Verification** / A KeYmaera model of the KPP scheme
  - to reproduce the undesired behavior.
  - to confirm that no corner case is being missed.

Test Case: The Keymaera X Model of the KPP scheme

The hybrid model of the KPP scheme:

$$
\begin{array}{l}
\hline
\mathit{initialConditions}() \rightarrow \\
\quad \left[\{\right. \\
\quad\quad \mathit{compute\_OBL\_depth}(); \qquad\qquad // \textit{ discrete updates} \\
\quad\quad \mathit{compute\_interior\_K}(); \\
\quad\quad \mathit{compute\_OBL\_K}(); \\
\quad\quad \{z'_{cr} = -z_{cr}\} \qquad\qquad // \textit{ continuous system} \\
\quad \left.\{\right\}^*\right] \\
K > 0 \\
\hline
\end{array}
$$

where K is diffusivity and z_{cr} is the depth at which Ri_B is equal to Ri_{cr}.

Test Case: The Keymaera X Model of the KPP scheme

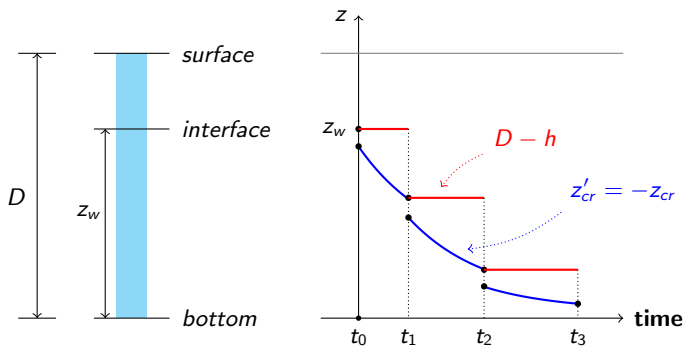
The hybrid model of the KPP scheme:

$$
\begin{array}{l}
\textit{initialConditions}() \rightarrow \\
\quad \left[\{\right. \\
\quad\quad \left.\begin{array}{l} \textit{compute\_OBL\_depth}(); \\ \textit{compute\_interior\_K}(); \\ \textit{compute\_OBL\_K}(); \end{array}\right\} \text{instantaneous} \qquad // \textit{ discrete updates} \\
\quad\quad \left\{\right. \{z'_{cr} = -z_{cr}\} \left.\right\} \text{over some } \Delta t \qquad // \textit{ continuous system} \\
\quad \left.\{\}^*\right] \\
\quad K > 0
\end{array}
$$

where K is diffusivity and z_{cr} is the depth at which Ri_B is equal to Ri_{cr}.

Test Case: The Keymaera X Model of the KPP scheme

Continuous evolution of z_{cr} + discrete changes:



Generating Proof Tactics

- Hilbert-style formal deduction: a finite sequence of formulas (axioms and inferences)

$$\phi \rightarrow [\alpha]\psi$$

Modal logic

$$\frac{\Gamma \vdash J, \Delta \qquad J \vdash [\alpha]J \qquad J \vdash P}{\Gamma \vdash [\alpha^*]P, \Delta}$$

Predicate logic
(e.g. loop invariant)

$$\frac{Q \vdash [x' := f(x)](F)'}{F \vdash [x' = f(x)\&Q]F}$$

Differential dynamic logic
(e.g. differential invariant)

The complete KeYmaera Model and the proof tactic:
https://github.com/alperaltuntas/keymaera-kpp

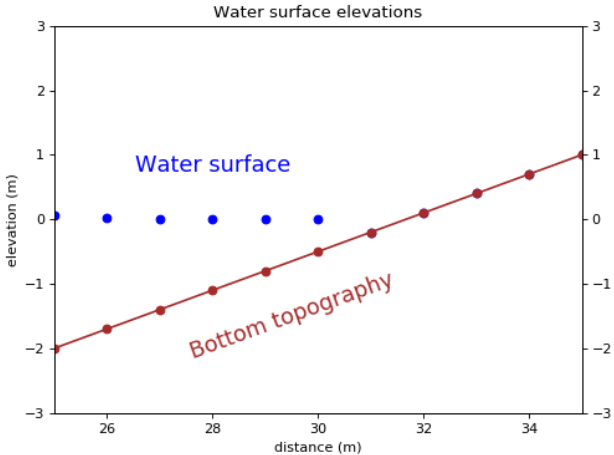
Conclusions

- A lightweight formal methods application.
- Highly efficient compared to testing.
- Provides more confidence.
  - Generality (due to nondeterminism)
  - The coverage in the temporal dimension is much greater.
- Limitations:
  - floating point arithmetic
  - numerical issues

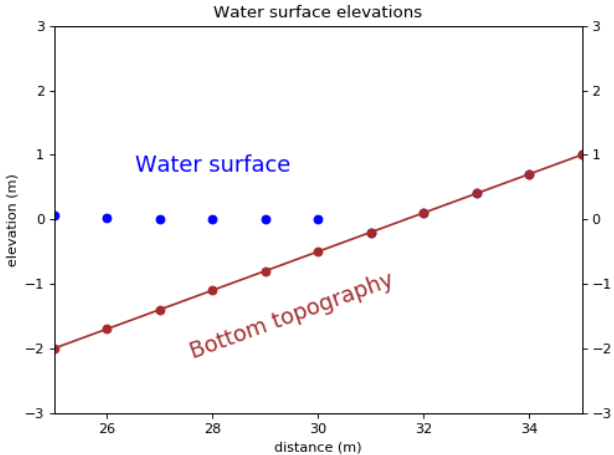
Thanks

altuntas@ucar.edu

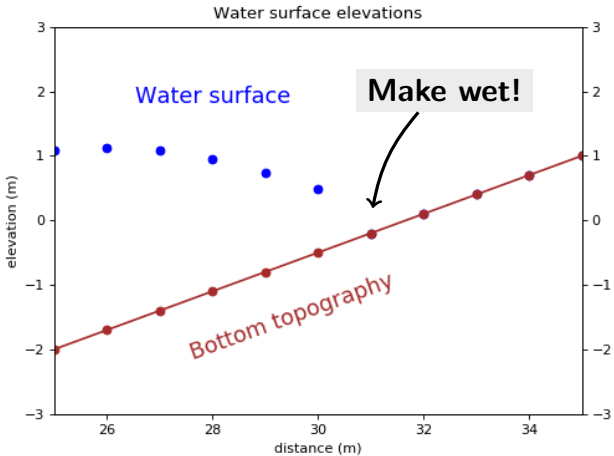
Hybrid theorem proving for verifying numerical software



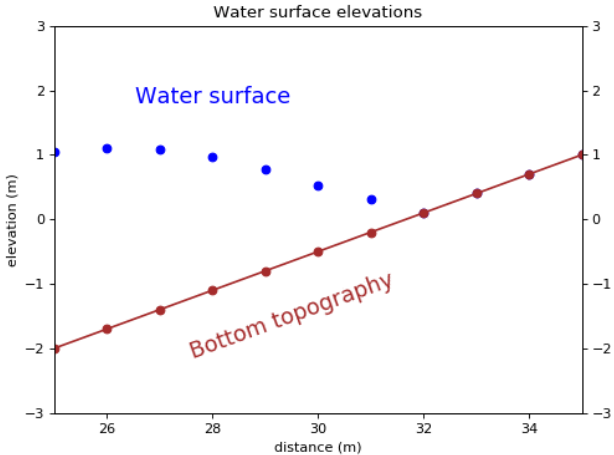
Hybrid theorem proving for verifying numerical software



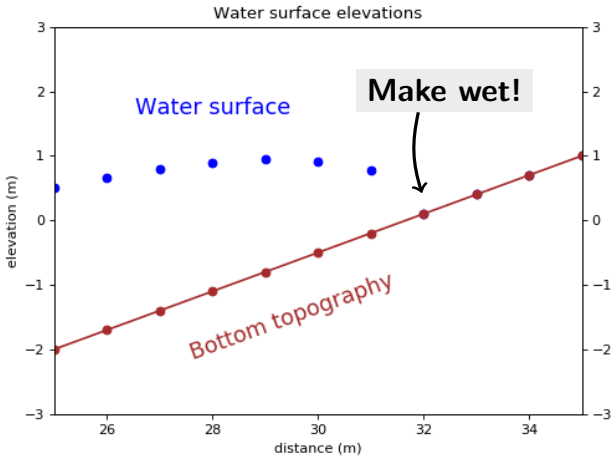
Hybrid theorem proving for verifying numerical software



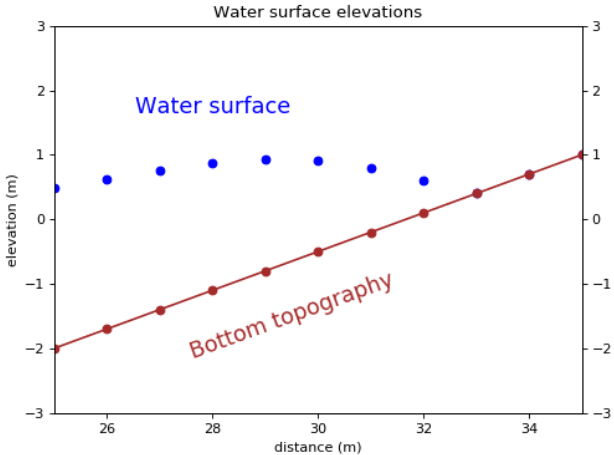
Hybrid theorem proving for verifying numerical software



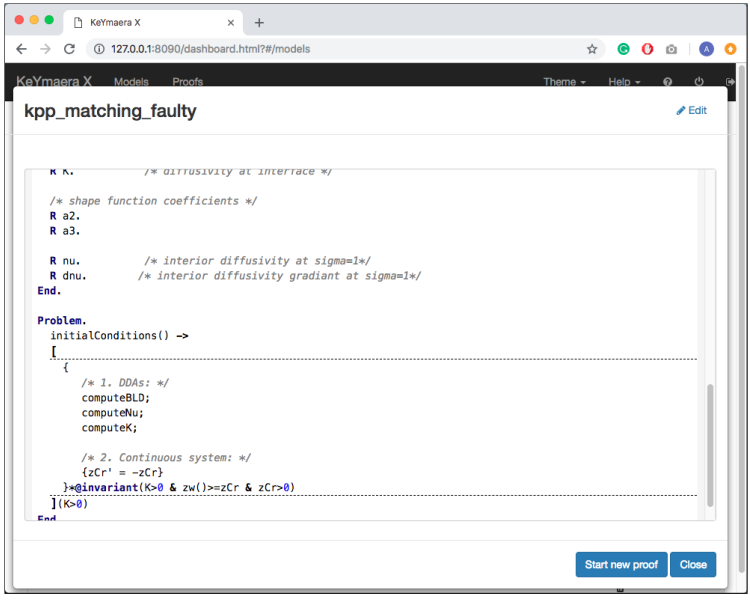
Hybrid theorem proving for verifying numerical software



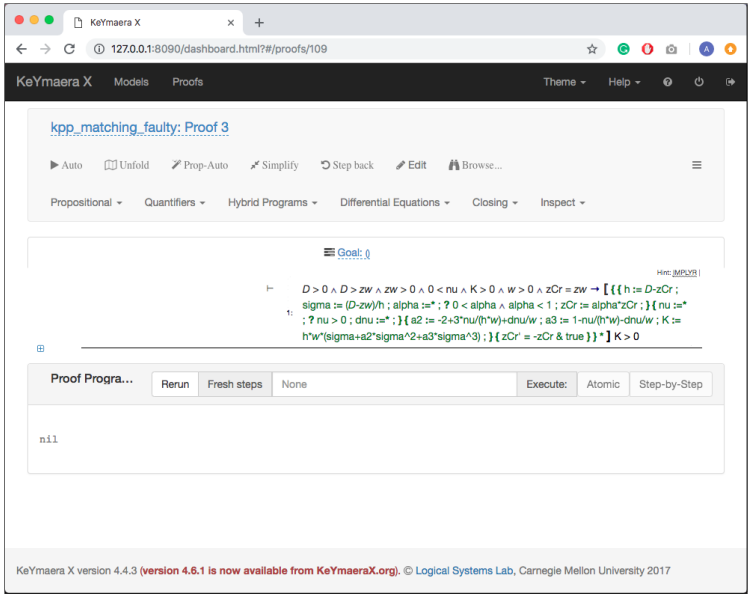
Hybrid theorem proving for verifying numerical software



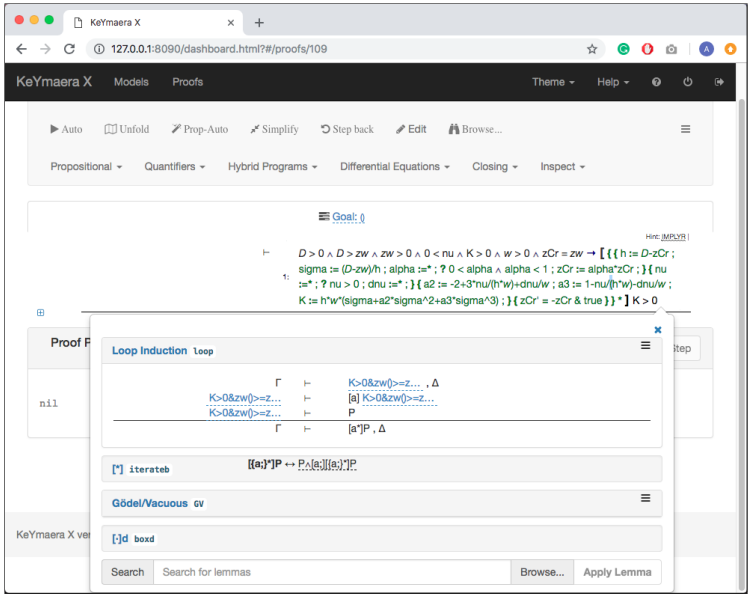
KeYmaera X UI:
Develop the hybrid model.



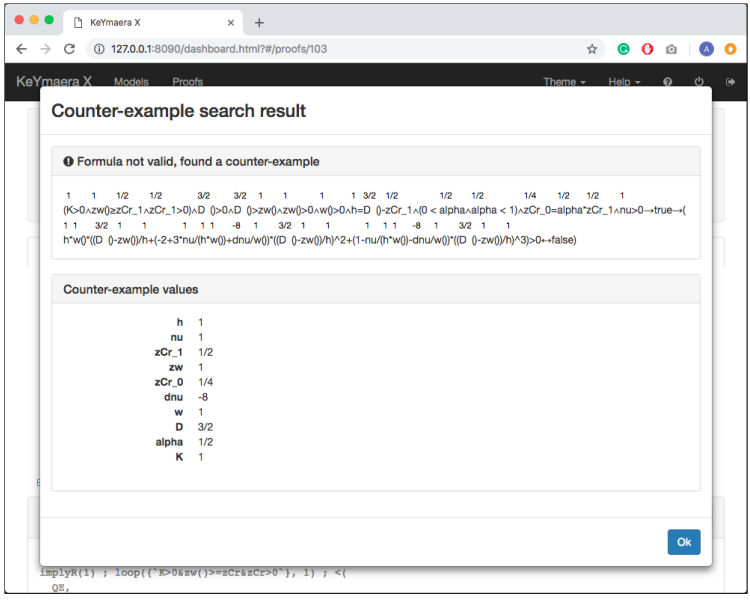
KeYmaera X UI:
Construct a proof.



KeYmaera X UI:
Steer the proof.



KeYmaera X UI:
Generate counter-example if formula is invalid.



KeYmaera X UI:
Close the proof.