Wave Equation Numerical Resolution: a Comprehensive Mechanized Proof of a C Program

Sylvie Boldo François Clément Jean-Christophe Filliâtre
Micaela Mayero **Guillaume Melquiond** Pierre Weis

ANR F∮ST

Inria Saclay–Île-de-France
Inria Paris–Rocquencourt
Université Paris-Sud, LRI
Université Paris-Nord, LIPN

July 26th, 2012

Scientific Computing

PDE (Partial Differential Equation)
⇒ weather forecast, numerical simulation, control, . . .

Scientific Computing

PDE (Partial Differential Equation)
⇒ weather forecast, numerical simulation, control, . . .

Usually no simple solutions ⇒ need for discretization.
Example: numerical scheme over a regular grid.

Scientific Computing

PDE (Partial Differential Equation)
⇒ weather forecast, numerical simulation, control, . . .

Usually no simple solutions ⇒ need for discretization.
Example: numerical scheme over a regular grid.

Goal: formally prove the C implementation of a numerical scheme.

One-Dimensional Wave Equation

Goal: finding a solution to the equation

$$\frac{\partial^2 u(x,t)}{\partial t^2} - c^2 \frac{\partial^2 u(x,t)}{\partial x^2} = s(x,t)$$

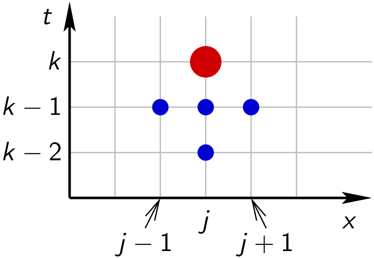
given the initial position $u_0(x)$ and the initial speed $u_1(x)$.

Applications: rope oscillations, sound propagation, radar, oil exploration.

Numerical Scheme

Discretization: $u_j^k \approx u(j\Delta x, k\Delta t)$.

$$\frac{u_j^k - 2u_j^{k-1} + u_j^{k-2}}{\Delta t^2} - c^2 \frac{u_{j+1}^{k-1} - 2u_j^{k-1} + u_{j-1}^{k-1}}{\Delta x^2} = s_j^{k-1}$$



Second-order centered finite difference scheme:
u_j^k depends on u_{j-1}^{k-1}, u_j^{k-1}, u_{j+1}^{k-1}, and u_j^{k-2}.

C Program

- Features: dynamic allocation, floating-point computations.
- Main loop:

```
/* Evolution problem and boundary conditions. */
/* Propagation = time loop. */
for (k=1; k<nk; k++) {
  /* Left boundary. */
  p[0][k+1] = 0.;
  /* Time iteration k = space loop. */
  for (i=1; i<ni; i++) {
    dp = p[i+1][k] - 2.*p[i][k] + p[i-1][k];
    p[i][k+1] = 2.*p[i][k] - p[i][k-1] + a*dp;
  }
  /* Right boundary. */
  p[ni][k+1] = 0.;
}
```

Verification

Goal: ensure that computed values are accurate approximations of the mathematical solution.

Verification

Goal: ensure that computed values are accurate approximations of the mathematical solution.

Issues:

- Method error:
  Is the numerical scheme an accurate enough approximation of the exact solution?

Verification

Goal: ensure that computed values are accurate approximations of the mathematical solution.

Issues:

- Method error:
  Is the numerical scheme an accurate enough approximation of the exact solution?
- Round-off errors:
  Floating-point computations introduce additional errors.
  How much do they perturb the results?

Verification

Goal: ensure that computed values are accurate approximations of the mathematical solution.

Issues:

- Method error:
  Is the numerical scheme an accurate enough approximation of the exact solution?
- Round-off errors:
  Floating-point computations introduce additional errors.
  How much do they perturb the results?
- Coding errors:
  Does the C program match the numerical scheme?
  Is the program free of runtime errors?

Available Methods and Tools

- Scientific computing:
  abundant literature, comprehensive and correct overall.
- Floating-point arithmetic:
  standard IEEE-754, literature, tools (Gappa).
- Formal proofs:
  tools (Coq), libraries ($\mathbb{R}$, FP numbers, etc).
- Program specification verification:
  tools (Frama-C, Jessie, Why).

Outline

Outline

1. Introduction
2. Tools
3. Method Error
4. Round-off Error
5. Verification
6. Conclusion

Process and Tools

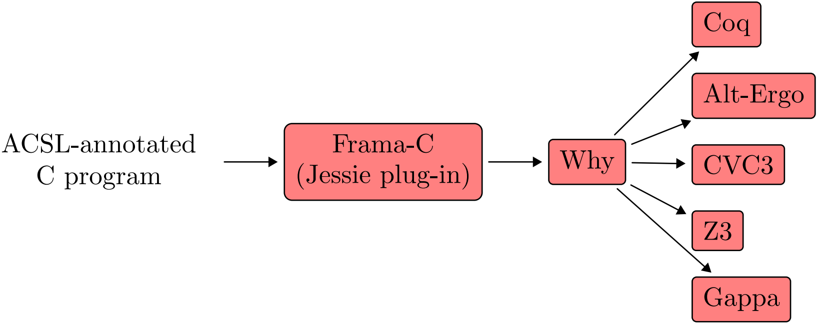
To prove a program:

- annotate it with its specification,
- compute its weakest preconditions,
- prove the resulting theorem statements.

Process and Tools

To prove a program:

- annotate it with its specification,
- compute its weakest preconditions,
- prove the resulting theorem statements.



ACSL-annotated C program

Frama-C (Jessie plug-in)

Why

Coq

Alt-Ergo

CVC3

Z3

Gappa

Frama-C/Jessie/Why

- Input: a C program with ACSL comments.
- Output: a set of theorem statements.
- Purpose: proving the theorems is sufficient to ensure that the program complies with its specification.

```
/*@ requires
  @   ni >= 2 && nk >= 2 && l != 0 &&
  @   dt > 0. && \exact(dt) > 0. &&
  @   ...;
  @
  @ ensures
  @   \forall integer k; 0 <= k <= nk ==>
  @   norm_dx_conv_err(\result, \exact(dt), ni, k) <=
  @     C_conv * (1. / (ni * ni) + \exact(dt) * \exact(dt));
  @ */
double **forward_prop(int ni, int nk, double dt, double v,
                      double xs, double l) {
```

SMT Solvers

- Automated theorem provers.
- Especially useful for statements related to coding errors: memory allocations, out-of-bound array accesses, etc.
- Based on Satisfiability Modulo Theories: linear integer/rational arithmetic, arrays, bit vectors, etc.
- Prominent solvers: Alt-Ergo, CVC3, Z3.

Gappa

- Automated theorem prover.
- Dedicated to FP-related properties:
  no overflow, bounded round-off errors, etc.
- Based on interval arithmetic and rewriting.
- Usable inside a Coq proof.

Coq

- Interactive proof assistant.
- Higher-order logic.
- Useful for anything that exceeds automated theorem provers.

Outline

1. Introduction
2. Tools
3. **Method Error**
4. Round-off Error
5. Verification
6. Conclusion

Method Error: What to Bound?

Goal: prove that the exact solution u and the discrete approximation u_j^k are close when $(\Delta x, \Delta t) \to 0$.

Convergence error: $e_j^k = \bar{u}_j^k - u_j^k$
with $\bar{u}_j^k$ the value of u at grid point (j, k).

Method Error: What to Bound?

Goal: prove that the exact solution u and the discrete approximation u_j^k are close when $(\Delta x, \Delta t) \to 0$.

Convergence error: $e_j^k = \bar{u}_j^k - u_j^k$
with $\bar{u}_j^k$ the value of u at grid point (j, k).

Averaged convergence error: $\left\| e^{k_{\Delta t}(t)} \right\|_{\Delta x}$
for all the grid points at time $k_{\Delta t}(t) = \left\lfloor \frac{t}{\Delta t} \right\rfloor \Delta t$.

Convergence of this numerical scheme is
a well-known mathematical result:

$$\left\| e^{k_{\Delta t}(t)} \right\|_{\Delta x} = O_{[0, t_{\max}]}(\Delta x^2 + \Delta t^2)$$

Proof Sketch: 1/3 Consistency

The discrete approximation u_j^k is a solution of the numerical scheme (by definition).

Truncation error: how close is $\bar{u}_j^k$ from being a solution of the numerical scheme?

$$\varepsilon_j^{k-1} = \frac{\bar{u}_j^k - 2\bar{u}_j^{k-1} + \bar{u}_j^{k-2}}{\Delta t^2} - c^2 \frac{\bar{u}_{j+1}^{k-1} - 2\bar{u}_j^{k-1} + \bar{u}_{j-1}^{k-1}}{\Delta x^2} - s_j^{k-1}$$

Proof Sketch: 1/3 Consistency

The discrete approximation u_j^k is a solution of the numerical scheme (by definition).

Truncation error: how close is $\bar{u}_j^k$ from being a solution of the numerical scheme?

$$\varepsilon_j^{k-1} = \frac{\bar{u}_j^k - 2\bar{u}_j^{k-1} + \bar{u}_j^{k-2}}{\Delta t^2} - c^2 \frac{\bar{u}_{j+1}^{k-1} - 2\bar{u}_j^{k-1} + \bar{u}_{j-1}^{k-1}}{\Delta x^2} - s_j^{k-1}$$

Consistency: the truncation error is bounded.

$$\left\| \varepsilon^{k_{\Delta t}(t)} \right\|_{\Delta x} = O_{[0,t_{\max}]}(\Delta x^2 + \Delta t^2)$$

Proved in Coq thanks to Taylor expansions and lots of computations.

Proof Sketch: 2/3 Stability

Stability: u_j^k does not diverge when $k \to \infty$.

Proof Sketch: 2/3 Stability

Stability: u_j^k does not diverge when $k \to \infty$.

Discrete energy:

$$E^{k+\frac{1}{2}} = \frac{1}{2}\left\|\frac{u^{k+1}-u^k}{\Delta t}\right\|_{\Delta x}^2 + \frac{1}{2}\left\langle u^k, u^{k+1}\right\rangle_A$$

kinetic energy potential energy

with $\langle v, w\rangle_A = \langle A(v), w\rangle_{\Delta x}$ and $A(v)_j = -c^2\frac{v_{j+1}-2v_j+v_{j-1}}{\Delta x^2}$.

Proof Sketch: 2/3 Stability

Stability: u_j^k does not diverge when $k \to \infty$.

Discrete energy:

$$E^{k+\frac{1}{2}} = \underbrace{\frac{1}{2}\left\|\frac{u^{k+1}-u^k}{\Delta t}\right\|_{\Delta x}^2}_{\text{kinetic energy}} + \underbrace{\frac{1}{2}\left\langle u^k, u^{k+1}\right\rangle_A}_{\text{potential energy}}$$

with $\langle v, w\rangle_A = \langle A(v), w\rangle_{\Delta x}$ and $A(v)_j = -c^2\frac{v_{j+1}-2v_j+v_{j-1}}{\Delta x^2}$.

Note: physical approach specific to this PDE.

Proof Sketch: 3/3 Convergence

The convergence error $e_j^k = \bar{u}_j^k - u_j^k$ is a solution of the numerical scheme when the source term s_j^k is replaced by the truncation error ε_j^{k+1} (and a few other changes).

Proof Sketch: 3/3 Convergence

The convergence error $e_j^k = \bar{u}_j^k - u_j^k$ is a solution of the numerical scheme when the source term s_j^k is replaced by the truncation error ε_j^{k+1} (and a few other changes).

1. ε_j^{k+1} is bounded on average, by consistency.
2. e_j^k does not diverge, by stability.
3. Thus u_j^k converges to $\bar{u}_j^k$ when $(\Delta x, \Delta t) \to 0$.

Convergence

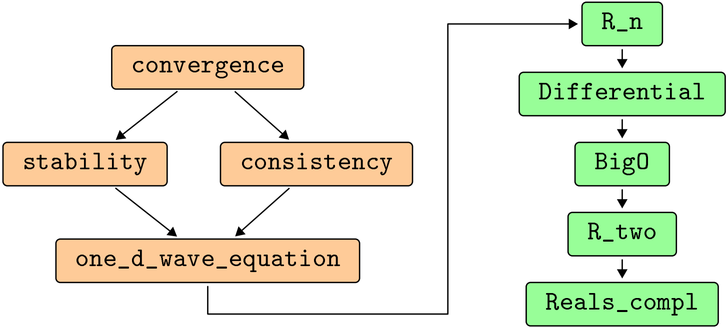
$$\left\| e_h^{k_{\Delta t}(t)} \right\|_{\Delta x} = O_{\left|\begin{array}{l} t \in [0, t_{\max}] \\ \\ (\Delta x, \Delta t) \to 0 \\ 0 < \Delta x \,\wedge\, 0 < \Delta t \,\wedge \\ c\frac{\Delta t}{\Delta x} \leq 1 - \xi \end{array}\right.} (\Delta x^2 + \Delta t^2).$$

Convergence

$$\left\| e_h^{k_{\Delta t}(t)} \right\|_{\Delta x} = O_{\left|\begin{array}{l} t \in [0, t_{\max}] \\ \\ (\Delta x, \Delta t) \to 0 \\ 0 < \Delta x \wedge 0 < \Delta t \wedge \\ c\frac{\Delta t}{\Delta x} \le 1 - \xi \end{array}\right.} (\Delta x^2 + \Delta t^2).$$

Note: the definition of the big O is much more detailed than in traditional pen-and-paper proofs.

Method Error: Conclusion



4500 lines of Coq
≈ as long as a detailed pen-and-paper proof.

Outline

Round-off Errors

Main loop:

```
dp = p[i+1][k] - 2.*p[i][k] + p[i-1][k];
p[i][k+1] = 2.*p[i][k] - p[i][k-1] + a*dp;
```

Operations are performed with floating-point arithmetic;
it introduces round-off errors which propagate along computations.

Example: $\texttt{a*dp} = \texttt{a} \times \texttt{dp} \times (1 + \varepsilon)$ with $|\varepsilon| \le 2^{-53}$
assuming that $\texttt{a} \times \texttt{dp}$ does not underflow.

Round-off Errors

Main loop:

```
dp = p[i+1][k] - 2.*p[i][k] + p[i-1][k];
p[i][k+1] = 2.*p[i][k] - p[i][k-1] + a*dp;
```

Operations are performed with floating-point arithmetic;
it introduces round-off errors which propagate along computations.

Example: $\texttt{a*dp} = \texttt{a} \times \texttt{dp} \times (1+\varepsilon)$ with $|\varepsilon| \le 2^{-53}$
assuming that $\texttt{a} \times \texttt{dp}$ does not underflow.

Naive forward error analysis gives

$$|p_i^k - u_i^k| \le O(2^k 2^{-53}).$$

Note: if that was optimal, the numerical scheme would be useless.

Local Round-off Errors

Main loop:

```
dp = p[i+1][k] - 2.*p[i][k] + p[i-1][k];
p[i][k+1] = 2.*p[i][k] - p[i][k-1] + a*dp;
```

Local round-off error:

$$\varepsilon_i^{k+1} = p_i^{k+1} - \left(2p_i^k - p_i^{k-1} + \left(c\frac{\Delta t}{\Delta x}\right)^2 (p_{i+1}^k - 2p_i^k + p_{i-1}^k)\right).$$

Local Round-off Errors

Main loop:

```
dp = p[i+1][k] - 2.*p[i][k] + p[i-1][k];
p[i][k+1] = 2.*p[i][k] - p[i][k-1] + a*dp;
```

Local round-off error:

$$\varepsilon_i^{k+1} = p_i^{k+1} - \left(2p_i^k - p_i^{k-1} + \left(c\frac{\Delta t}{\Delta x}\right)^2 (p_{i+1}^k - 2p_i^k + p_{i-1}^k)\right).$$

Assuming that all the computed values p_i^k are less than 2 (the values u_i^k are less than 1), then Gappa gives

$$|\varepsilon_i^k| \leq 85 \times 2^{-52}.$$

Round-off Propagation

The global error $p_i^k - u_i^k$ depends on the following local errors:

$$\begin{array}{ccccccccc}
 & & & & \varepsilon_i^k & & & & \\
 & & & \varepsilon_{i-1}^{k-1} & \varepsilon_i^{k-1} & \varepsilon_{i+1}^{k-1} & & & \\
 & & \varepsilon_{i-2}^{k-2} & \varepsilon_{i-1}^{k-2} & \varepsilon_i^{k-2} & \varepsilon_{i+1}^{k-2} & \varepsilon_{i+2}^{k-2} & & \\
 & \iddots & & & \vdots & & & \ddots & \\
\varepsilon_{i-k}^0 & & \cdots & & \varepsilon_i^0 & & \cdots & & \varepsilon_{i+k}^0
\end{array}$$

Round-off Propagation

The global error $p_i^k - u_i^k$ depends on the following local errors:

$$\begin{array}{ccccccccc}
 & & & & \varepsilon_i^k & & & & \\
 & & & \varepsilon_{i-1}^{k-1} & \varepsilon_i^{k-1} & \varepsilon_{i+1}^{k-1} & & & \\
 & & \varepsilon_{i-2}^{k-2} & \varepsilon_{i-1}^{k-2} & \varepsilon_i^{k-2} & \varepsilon_{i+1}^{k-2} & \varepsilon_{i+2}^{k-2} & & \\
 & \iddots & & & \vdots & & & \ddots & \\
\varepsilon_{i-k}^0 & & \cdots & & \varepsilon_i^0 & & \cdots & & \varepsilon_{i+k}^0
\end{array}$$

The relation is given by the convolution product of ε by the solution λ of the numerical scheme applied to a single nonzero initial condition ($\lambda_0^0 = 1$).

$$p_i^k - u_i^k = (\lambda * \varepsilon)_i^k = \sum_{l=0}^{k} \sum_{j=-l}^{l} \lambda_j^l \, \varepsilon_{i+j}^{k-l}.$$

Round-off Error

- $p_i^k - u_i^k = \sum_{l=0}^{k} \sum_{j=-l}^{l} \lambda_j^l \, \varepsilon_{i+j}^{k-l}$.
- λ is solution of the numerical scheme: stability ⇒ bounded.

Round-off Error

- $p_i^k - u_i^k = \sum_{l=0}^{k} \sum_{j=-l}^{l} \lambda_j^l \, \varepsilon_{i+j}^{k-l}$.
- λ is solution of the numerical scheme: stability ⇒ bounded.

Round-off error: $O(k^2 2^{-53})$.

$$|p_i^k - u_i^k| \leq 85 \times 2^{-53} \times (k+1) \times (k+2).$$

Outline

1. Introduction
2. Tools
3. Method Error
4. Round-off Error
5. Verification
6. Conclusion

Annotations: Mathematical Definitions

```
/*@ axiomatic dirichlet_maths {
  @ logic real psol(real x, real t);

  @ logic real psol_1(real x, real t);
  @ axiom psol_1_def:
  @   \forall real x; \forall real t;
  @   \forall real eps; 0 < eps ==>
  @   \exists real C; 0 < C && \forall real dx; \abs(dx) < C ==>
  @   \abs((psol(x + dx, t) - psol(x, t)) / dx - psol_1(x, t)) < eps;

  @ logic real p0(real x);
  @ axiom wave_eq_0: \forall real x; 0 <= x <= 1 ==> psol(x, 0) == p0(x);
  @ ...
  @ } */
```

Annotations: Mathematical Definitions

```
/*@ axiomatic dirichlet_maths {
  @ logic real psol(real x, real t);

  @ logic real psol_1(real x, real t);
  @ axiom psol_1_def:
  @   \forall real x; \forall real t;
  @   \forall real eps; 0 < eps ==>
  @   \exists real C; 0 < C && \forall real dx; \abs(dx) < C ==>
  @   \abs((psol(x + dx, t) - psol(x, t)) / dx - psol_1(x, t)) < eps;

  @ logic real p0(real x);
  @ axiom wave_eq_0: \forall real x; 0 <= x <= 1 ==> psol(x, 0) == p0(x);
  @ ...
  @ } */
```

Exact solution u

Annotations: Mathematical Definitions

Definition of $\frac{\partial u}{\partial t}$

```
/*@ axiomatic dirichlet_maths {
  @ logic real psol(real x, real t);

  @ logic real psol_1(real x, real t);
  @ axiom psol_1_def:
  @   \forall real x; \forall real t;
  @   \forall real eps; 0 < eps ==>
  @   \exists real C; 0 < C && \forall real dx; \abs(dx) < C ==>
  @   \abs((psol(x + dx, t) - psol(x, t)) / dx - psol_1(x, t)) < eps;

  @ logic real p0(real x);
  @ axiom wave_eq_0: \forall real x; 0 <= x <= 1 ==> psol(x, 0) == p0(x);
  @ ...
  @ } */
```

Annotations: Mathematical Definitions

```
/*@ axiomatic dirichlet_maths {
  @ logic real psol(real x, real t);

  @ logic real psol_1(real x, real t);
  @ axiom psol_1_def:
  @   \forall real x; \forall real t;
  @   \forall real eps; 0 < eps ==>
  @   \exists real C; 0 < C && \forall real dx; \abs(dx) < C ==>
  @   \abs((psol(x + dx, t) - psol(x, t)) / dx - psol_1(x, t)) < eps;

  @ logic real p0(real x);
  @ axiom wave_eq_0: \forall real x; 0 <= x <= 1 ==> psol(x, 0) == p0(x);
  @ ...
  @ } */
```

Initial condition

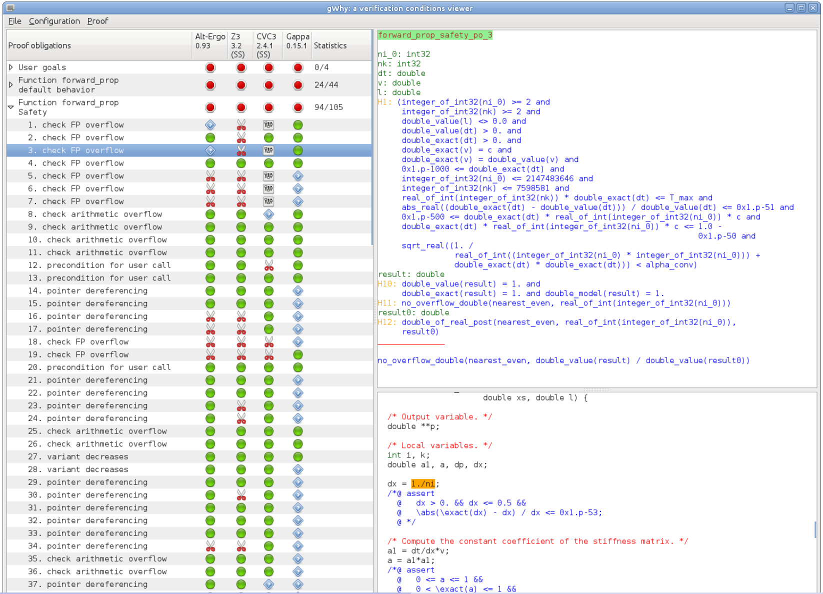
Annotations: Relation Between Mathematics and Program

```
/*@ requires (l != 0);
  @ ensures
  @   \round_error(\result) <= 14 * 0x1.p-52 &&
  @   \exact(\result) == p0(\exact(x));
  @ */
double p_zero(double xs, double l, double x);
```

Annotations: Program Behavior

```
/*@ loop invariant
  @   1 <= k <= nk &&
  @   analytic_error(p, ni, ni, k, a, dt);
  @ loop variant nk - k; */
for (k=1; k<nk; k++) {
  p[0][k+1] = 0.;
  /*@ loop invariant
    @   1 <= i <= ni &&
    @   analytic_error(p, ni, i - 1, k + 1, a, dt);
    @ loop variant ni - i; */
  for (i=1; i<ni; i++) {
    dp = p[i+1][k] - 2.*p[i][k] + p[i-1][k];
    p[i][k+1] = 2.*p[i][k] - p[i][k-1] + a*dp;
  }
  p[ni][k+1] = 0.;
  /*@ assert analytic_error(p, ni, ni, k + 1, a, dt); */
}
```

Verification: Proof Obligations



Proof obligations	Alt-Ergo 0.93	Z3 3.2 (SS)	CVC3 2.4.1 (SS)	Gappa 0.15.1	Statistics
User goals	🔴	🔴	🔴	🔴	0/4
Function forward_prop default behavior	🔴	🔴	🔴	🔴	24/44
Function forward_prop Safety	🔴	🔴	🔴	🔴	94/105
1. check FP overflow	◈	✂	▣	🟢	
2. check FP overflow	🟢	✂	🟢	🟢	
3. check FP overflow	◈	✂	▣	🟢	
4. check FP overflow	🟢	🟢	🟢	🟢	
5. check FP overflow	✂	✂	▣	◈	
6. check FP overflow	✂	✂	▣	◈	
7. check FP overflow	✂	✂	▣	◈	
8. check arithmetic overflow	🟢	🟢	◈	🟢	
9. check arithmetic overflow	🟢	🟢	🟢	🟢	
10. check arithmetic overflow	🟢	🟢	🟢	🟢	
11. check arithmetic overflow	🟢	🟢	🟢	🟢	
12. precondition for user call	🟢	🟢	✂	🟢	
13. precondition for user call	🟢	🟢	🟢	🟢	
14. pointer dereferencing	🟢	🟢	🟢	◈	
15. pointer dereferencing	🟢	🟢	🟢	◈	
16. pointer dereferencing	✂	✂	🟢	◈	
17. pointer dereferencing	✂	✂	🟢	◈	
18. check FP overflow	✂	✂	✂	◈	
19. check FP overflow	✂	✂	✂	🟢	
20. precondition for user call	🟢	🟢	🟢	🟢	
21. pointer dereferencing	🟢	🟢	🟢	◈	
22. pointer dereferencing	🟢	🟢	🟢	◈	
23. pointer dereferencing	🟢	✂	🟢	◈	
24. pointer dereferencing	🟢	✂	🟢	◈	
25. check arithmetic overflow	🟢	🟢	🟢	🟢	
26. check arithmetic overflow	🟢	🟢	🟢	🟢	
27. variant decreases	🟢	🟢	🟢	🟢	
28. variant decreases	🟢	🟢	🟢	◈	
29. pointer dereferencing	🟢	🟢	🟢	◈	
30. pointer dereferencing	🟢	✂	🟢	◈	
31. pointer dereferencing	🟢	🟢	🟢	◈	
32. pointer dereferencing	🟢	🟢	🟢	◈	
33. pointer dereferencing	🟢	🟢	🟢	◈	
34. pointer dereferencing	✂	✂	🟢	◈	
35. check arithmetic overflow	🟢	🟢	🟢	◈	
36. check arithmetic overflow	🟢	🟢	🟢	◈	
37. pointer dereferencing	🟢	🟢	◈	◈	

```
forward_prop_safety_po_3

ni_0: int32
nk: int32
dt: double
v: double
l: double
H1: (integer_of_int32(ni_0) >= 2 and
     integer_of_int32(nk) >= 2 and
     double_value(l) <> 0.0 and
     double_value(dt) > 0. and
     double_exact(dt) > 0. and
     double_exact(v) = c and
     double_exact(v) = double_value(v) and
     0x1.p-1000 <= double_exact(dt) and
     integer_of_int32(ni_0) <= 2147483646 and
     integer_of_int32(nk) <= 7598581 and
     real_of_int(integer_of_int32(nk)) * double_exact(dt) <= T_max and
     abs_real((double_exact(dt) - double_value(dt))) / double_value(dt) <= 0x1.p-51 and
     0x1.p-500 <= double_exact(dt) * real_of_int(integer_of_int32(ni_0)) * c and
     double_exact(dt) * real_of_int(integer_of_int32(ni_0)) * c <= 1.0 -
                                                              0x1.p-50 and
     sqrt_real((1. /
                real_of_int((integer_of_int32(ni_0) * integer_of_int32(ni_0))) +
                double_exact(dt) * double_exact(dt))) < alpha_conv)
result: double
H10: double_value(result) = 1. and
     double_exact(result) = 1. and double_model(result) = 1.
H11: no_overflow_double(nearest_even, real_of_int(integer_of_int32(ni_0)))
result0: double
H12: double_of_real_post(nearest_even, real_of_int(integer_of_int32(ni_0)),
     result0)
________________
no_overflow_double(nearest_even, double_value(result) / double_value(result0))
```

```
                   double xs, double l) {

  /* Output variable. */
  double **p;

  /* Local variables. */
  int i, k;
  double a1, a, dp, dx;

  dx = 1./ni;
  /*@ assert
    @   dx > 0. && dx <= 0.5 &&
    @   \abs(\exact(dx) - dx) / dx <= 0x1.p-53;
    @ */

  /* Compute the constant coefficient of the stiffness matrix. */
  a1 = dt/dx*v;
  a = a1*a1;
  /*@ assert
    @   0 <= a <= 1 &&
    @   0 < \exact(a) <= 1 &&
```

Verification: Automation and Formal Proof

Verification conditions:

- behavior: what the program is supposed to perform,
- safety: the condition for it to terminate without error.

Prover	Proved Behavior VC	Proved Safety VC	Total
Alt-Ergo	18	80	98
CVC3	18	89	107
Gappa	2	20	22
Z3	21	63	84
Automatic	**23**	**94**	**117**
Coq	**21**	**11**	**32**
Total	44	105	149

Outline

Conclusion

Development: `http://fost.saclay.inria.fr/`

- 60 lines of C code,
- 180 lines of ACSL annotations,
- 15000 lines of Coq definitions, theorems, and proofs.

Conclusion

Development: `http://fost.saclay.inria.fr/`

- 60 lines of C code,
- 180 lines of ACSL annotations,
- 15000 lines of Coq definitions, theorems, and proofs.

What was verified?

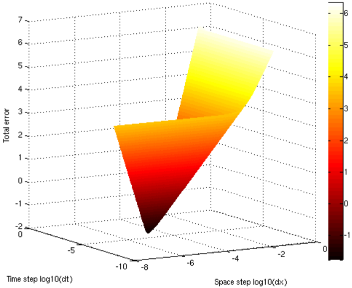
- The C program terminates without any runtime error.
- The method error is the expected one.
- The round-off error is small enough.

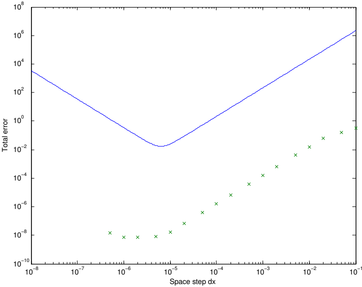
Benefits from Formal Proofs

- Confidence, confidence, confidence.

Benefits from Formal Proofs

- Confidence, confidence, confidence.
- One can extract the constants hidden inside the big O from the formal proofs:





Perspectives

- Prove Lax equivalence for several schemes:
  consistency ⇒ (stability ⇔ convergence).
- Do not rely on energy-based proofs,
  e.g. use Fourier transforms.

Questions?

`http://fost.saclay.inria.fr/`