CRISYS:
Constrained-system Rewriting Induction SYStem

Naoki Nishida

joint work with T. Sakata, Y. Nakano, W.-L. Ding, M. Sakai, T. Sakabe, K. Kusakari

Nagoya University

Kickoff Meeting of Austria–Japan Joint Project
Gamagori, July 4, 2012
My Very Boring Work in the First Semester (2005–)

C-programming exercise class

- 1 TA to mark reports.
- 70− students (60− are active).
- 30− exercises (3 in a week) and 10+ additional ones (1 in a week).

Hard to prove procedural programs, especially written by students.

To make this work interesting, I started a research on automated inductive theorem proving on constrained TRSs (2005).
My Very Boring Work in the First Semester (2005–)

C-programming exercise class

- 1 TA to mark reports.
- 70^- students (60^- are active).
- 30^- exercises (3 in a week) and 10^+ additional ones (1 in a week).
- 180^+ reports to review in a week (2K^+ in a semester).

Hard to prove procedural programs, especially written by students.

To make this work interesting, I started a research on automated inductive theorem proving on constrained TRSs (2005).
My Very Boring Work in the First Semester (2005–)

C-programming exercise class

- 1 TA to mark reports.
- 70\(^{-}\) students (60\(^{-}\) are active).
- 30\(^{-}\) exercises (3 in a week) and 10\(^{+}\) additional ones (1 in a week).
- 180\(^{+}\) reports to review in a week (2K\(^{+}\) in a semester).
- Hard to prove procedural programs, especially written by students.
My Very Boring Work in the First Semester (2005–)

C-programming exercise class

- 1 TA to mark reports.
- 70− students (60− are active).
- 30− exercises (3 in a week) and 10+ additional ones (1 in a week).
- 180+ reports to review in a week (2K+ in a semester).
- Hard to prove procedural programs, especially written by students.

To make this work interesting, I started a research on automated inductive theorem proving on constrained TRSs (2005).
Our Research Topics on Constrained TRSs

- Inductionless induction based on completion [Furuichi et al, 08]
  - Transformation of C programs into constrained TRSs
- Rewriting induction [Sakata et al, 09]
  - main part of theorem proving
- Termination prover for constrained TRSs [Sakata et al, 11]
  - necessary in the RI method
- Lemma generation [Nakabayashi et al, 10]
  - necessary in many cases
- Verification via tree homomorphisms [Takakuwa et al, 11]
  - light equivalence prover
- Constrained tee automata [Nishida et al, 12]
  - necessary(?) for automating the RI method
Constrained Rewriting [Bouhoula et al, 08][Furuichi et al, 08]

- Given
  - $\mathcal{F}$ a set of uninterpretable function symbols,
  - $\mathcal{G}$ a set of interpretable function symbols,
  - $\mathcal{P}$ a set of predicate symbols,
  - $S$ a structure for $\mathcal{G}$ and $\mathcal{P}$ (e.g., supported by SMT solvers),

a constrained TRS $\mathcal{R}$ is a finite set of constrained rewrite rules

$$l \rightarrow r \left[ \phi \right]$$

where $l \in T(\mathcal{F} \cup \mathcal{G}, \mathcal{V}) \setminus \mathcal{V}$, $r \in T(\mathcal{F} \cup \mathcal{G}, \mathcal{V})$, and $\phi$ is a formula over $\mathcal{G}, \mathcal{P}, \mathcal{V}$.

- $C[l\sigma] \rightarrow_{\mathcal{R}} C[r\sigma]$ iff
  $$l \rightarrow r \left[ \phi \right] \in \mathcal{R}, \forall x \in \text{fv}(\phi). \sigma(x) \in T(\mathcal{G}, \mathcal{V}), \text{ and } \sigma(\phi) \text{ is } S\text{-valid.}$$
Example of Constrained TRSs (LIA constraints)

- $\mathcal{F} = \{ \text{sum} \}$
- $\mathcal{G}_{\text{LIA}} = \{ 0, s, p, \text{add} \}$,
- $\mathcal{P}_{\text{LIA}} = \{ =, \neq, <, \leq, >, \geq \}$,
- $\mathcal{S}_{\text{LIA}}$ with the universe $\mathbb{N}$ and
  - $0^{\mathcal{S}_{\text{LIA}}} := 0$,
  - $s^{\mathcal{S}_{\text{LIA}}}(x) := x + 1$,
  - $p^{\mathcal{S}_{\text{LIA}}}(x) := x - 1$,
  - $\text{add}^{\mathcal{S}_{\text{LIA}}}(x, y) := x + y$,
  - $x =^{\mathcal{S}_{\text{LIA}}} x := x = y$,
  - ...

$$\mathcal{R}_{\text{sum}} = \begin{cases} 
\text{sum}(x) \rightarrow 0 & \quad \left[ x \leq 0 \right] \\
\text{sum}(s(x)) \rightarrow \text{add}(s(x), \text{sum}(x)) & \quad \left[ x \geq 0 \right] \\
\text{add}(0, y) \rightarrow y \\
\text{add}(s(x), y) \rightarrow s(\text{add}(x, y)) \\
\text{add}(p(x), y) \rightarrow p(\text{add}(x, y)) \\
\text{s}(p(x)) \rightarrow x \\
\text{p}(s(x)) \rightarrow x \\
\end{cases}$$

$\text{sum}(s^{10}(0)) \rightarrow^{\mathcal{R}} \text{add}(s^{10}(0), \text{sum}(s^{9}(0))) \rightarrow^{*}_{\mathcal{R}} s^{55}(0)$
Example: outline of verifying C programs [Furuichi et al, 08]

C program

```c
int sum1(int x){
    int i=0, z=0;
    for( i=0 ; i<=x ; i++ ){
        z += i;
    }
    return z;
}
```

Specification

\[
\begin{align*}
\text{sum}(x) &= 0 & \text{if } x \leq 0 \\
\text{sum}(s(x)) &= s(x) + \text{sum}(x) & \text{if } x \geq 0
\end{align*}
\]

1. The C program and specification are transformed and simplified into

\[
R_{\text{sum1}} = R_{\text{plus}} \cup \left\{
\begin{array}{l}
\text{sum1}(x) \rightarrow U_1(x, 0, 0) \\
U_1(x, i, z) \rightarrow U_1(x, s(i), \text{plus}(z, i)) \ [i \leq x] \\
U_1(x, i, z) \rightarrow z \ [\neg(i \leq x)]
\end{array}\right\}
\]

\[
R_{\text{sum}} = \ldots
\]

\[
R_{\text{plus}} = \{ \text{plus}(0, y) \rightarrow y \quad \text{plus}(s(x), y) \rightarrow s(\text{plus}(x, y)) \ldots \}
\]

2. If \( \text{sum1}(x) \approx \text{sum}(x) \) is an inductive theorem of \( R_{\text{sum1}} \cup R_{\text{sum}} \), then \( \text{sum1} \) satisfies the specification on \( \text{sum} \).
Our Inductive Theorem Prover

- has the own SMT solver for LIA.
- has the own termination prover.
- automatically proves that \( \text{sum1}(x) \approx \text{sum}(x) \) is an inductive theorem of \( R_{\text{sum1}} \cup R_{\text{sum}} \).
  - An appropriate lemma is automatically generated.
Why Implemented an SMT Solver?

- When proving \( \text{sum1}(x) \approx \text{sum}(x) \), satisifiability of the following formula has to be decided:

\[
\forall x. \forall x_2. \forall i. \forall x_3.
((x + 1 > i \land x_2 + 1 > x_3 \land x = x_2 \land i + 1 = x_3) \implies -x_3 > x_7)
\]

- Satisfiability of LIA formulas is decidable.
- Yices, CVC3, Z3 return “unknown”.
Rewriting Induction for Constrained Equations [Sakata et al, 09]

If

$$(E_0, \emptyset) = (E_0, \mathcal{H}_0) \vdash (E_1, \mathcal{H}_1) \vdash \cdots \vdash (E_n, \mathcal{H}_n) = (\emptyset, \mathcal{H}_0)$$

then all equations in $E$ are inductive theorems of $\mathcal{R}$, where

**Simplification** $(E \cup \{ s \simeq C[l\sigma] \supset \phi \}, \mathcal{H}) \vdash (E \cup \{ s \simeq C[r\sigma] \supset \phi \}, \mathcal{H})$

where $l \rightarrow r \supset \psi \in \mathcal{R} \cup \mathcal{H}$, $\phi$ is $S$-sat, and $\phi \Rightarrow \sigma(\psi)$ is $S$-valid.

**EQ-Deletion** $(E \cup \{ C[s] \approx C[t] \supset \phi \}, \mathcal{H})$

$$\vdash (E \cup \{ C[s] \approx C[t] \supset \phi \wedge s \neq t \}, \mathcal{H})$$

where $s, t \in \mathcal{T}(\mathcal{G}, \mathcal{V})$ and $\mathcal{V}ar(s, t) \subseteq \mathcal{fv}(\phi)$.

**Deletion** $(E \cup \{ s \approx t \supset \phi \}, \mathcal{H}) \vdash (E, \mathcal{H})$

where $s \equiv t$ or $\phi$ is not $S$-sat.

**Expansion** $(E \cup \{ s \approx t \supset \phi \}, \mathcal{H})$

$$\vdash (E \cup \text{Expd}_p(s \rightarrow t \supset \phi), \mathcal{H} \cup \{ s \rightarrow t \supset \phi \})$$

where $\mathcal{R} \cup \mathcal{H} \cup \{ s \rightarrow t \supset \phi \}$ terminates, $p$ is an $\mathcal{R}$-complete occurrence, and $\text{Expd}_p(s \rightarrow t \supset \phi)$ is the set of critical pairs between $s \rightarrow t \supset \phi$ and rules in $\mathcal{R}$ at position $p$ of $s$. 
Standard Strategy for Inferences

Given $R$ and $\mathcal{E}$, apply the following steps to $(\mathcal{E}, \emptyset)$ until $E$ becomes empty:

1. apply Simplification as much as possible,
2. apply EQ-Deletion once to each equation,
3. apply Deletion as much as possible,
4. if $\mathcal{E}$ is empty, then halt successfully, and o/w, apply Expansion once. If Expansion is not applicable, then halt unsuccessfully.
Divergence of Constrained Equations

\[ \mathcal{R}_{\text{sum1}} \cup \mathcal{R}_{\text{sum}} = \begin{cases} 
(1) \ \text{sum1}(x) \rightarrow U_1(x, s(0), 0) \\
(2) \ U_1(x, i, z) \rightarrow U_1(x, s(i), \text{plus}(z, i)) \ \{ i \leq x \} \\
(3) \ U_1(x, i, z) \rightarrow z \ \{ \neg i \leq x \} \\
(4) \ \text{sum}(x) \rightarrow 0 \ \{ x \leq 0 \} \\
(5) \ \text{sum}(s(x)) \rightarrow \text{plus}(\text{sum}(x), s(x)) \ \{ x \geq 0 \} 
\end{cases} \bigcup \mathcal{R}_{\text{plus}} \]

\[ \mathcal{E} = \{ \text{sum1}(x) \approx \text{sum}(x) \} \]

The proof of \( \mathcal{E} \) has the following divergence:

\[ \text{plus}(U_1(x, s(0), 0), s(x)) \approx U_1(s(x), s^2(0), \text{plus}(0, s(0))) \ \{ 0 \leq s(x) \} \]
\[ \text{plus}(U_1(x, s^2(0), s(0)), s(x)) \approx U_1(s(x), s^3(0), \text{plus}(s(0), s^2(0))) \ \{ s^2(0) \leq s(x) \} \]

\[ : \]

A desired lemma is

\[ \text{plus}(U_1(x, i, z), s(x)) \approx U_1(s(x), s(i), \text{plus}(i, z)) \ \{ i \leq s(x) \} \]
How to Get a (Candidate of) Lemma Equations

\[ \mathcal{R}_{\text{sum1}} \cup \mathcal{R}_{\text{sum}} = \left\{ \begin{array}{ll}
(1) & \text{sum1}(x) \rightarrow U_1(x, s(0), 0) \\
(2) & U_1(x, i, z) \rightarrow U_1(x, s(i), \text{plus}(z, i)) \quad \left[ \begin{array}{c}
 i \leq x \\
 \neg i \leq x \\
 x \leq 0 \\
 x \geq 0
\end{array} \right]
(3) & U_1(x, i, z) \rightarrow z \\
(4) & \text{sum}(x) \rightarrow 0 \\
(5) & \text{sum}(s(x)) \rightarrow \text{plus}(\text{sum}(x), s(x)) \end{array} \right\} \cup \mathcal{R}_{\text{plus}} \]

\[ \mathcal{E} = \{ \text{sum1}(x) \approx \text{sum}(x) \} \]

The proof of \( \mathcal{E} \) has the following divergence:

\[ \text{plus}(U_1(x, s(0), 0), s(x)) \approx U_1(s(x), s^2(0), \text{plus}(0, s(0))) \quad \left[ \begin{array}{c}
 0 \leq s(x)
\end{array} \right] \]
\[ \text{plus}(U_1(x, s^2(0), s(0)), s(x)) \approx U_1(s(x), s^3(0), \text{plus}(s(0), s^2(0))) \quad \left[ s^2(0) \leq s(x) \right] \]
\[ : \]

A desired lemma is

\[ \text{plus}(U_1(x, i, z), s(x)) \approx U_1(s(x), s(i), \text{plus}(i, z)) \quad \left[ i \leq s(x) \right] \]
Strategy with Lemma Discovery [Nakabayashi et al, 09]

Given $R$ and $E$, apply the following steps to $(E, \emptyset)$ until $E$ becomes empty:

1. apply Simplification by as much as possible,
2. apply EQ-Deletion once to each equation,
3. apply Deletion as much as possible,
4. if an equation is diverging and the equation can be generalized to $e$,
   then try proving $(\{e\}, \emptyset)$:
   ▶ if succeeded by $(\{e\}, \emptyset) \vdash \cdots \vdash (\emptyset, H')$, then add $H'$ to $H$ and go to 1,
   ▶ o/w, go to the next.
5. if $E$ is empty, then halt successfully, and o/w, apply Expansion once.
   If Expansion is not applicable, then halt unsuccessfully.
Our Research Topics on Constrained TRSs

- Inductionless induction based on completion [Furuichi et al, 08]
  - Transformation of C programs into constrained TRSs
- Rewriting induction [Sakata et al, 09]
  - main part of theorem proving
- Termination prover for constrained TRSs [Sakata et al, 11]
  - necessary in the RI method
- Lemma generation [Nakabayashi et al, 10]
  - necessary in many cases
- Verification via tree homomorphisms [Takakuwa et al, 11]
  - light equivalence prover
- Constrained tee automata [Nishida et al, 12]
  - necessary(?) for automating the RI method
Experiments of Transforming C Programs for sum

[Ishigaki et al, 07][Takakuwa et al, 09]

- An exercise of C Programming Exercise Class in 2006
  - Write a function to, given $n$, compute the summation from 0 to $n$, without recursive calls.
- We succeeded in transforming 59 programs into constrained TRSs.
- After simplifying them, many programs were converted to the similar forms.
Example of Transforming into the Same

```c
int sum1(int x){
    int i=0, z=0;
    for( i=0 ; i<=x ; i++ ){
        z += i;
    }
    return z;
}
```

```c
int sum1(int x){
    int i=0, z=0;
    while( i<=x ){
        i++; z += i-1;
    }
    return z;
}
```

Both of the above are transformed into

\[
\mathcal{R}_{\text{sum1}} = \mathcal{R}_{\text{plus}} \cup \left\{ \begin{array}{l}
    \text{sum1}(x) \rightarrow U_1(x, 0, 0) \\
    U_1(x, i, z) \rightarrow U_1(x, s(i), \text{plus}(z, i)) \ [i \leq x] \\
    U_1(x, i, z) \rightarrow z \ [\neg(i \leq x)] \\
\end{array} \right\}
\]

Example of Transforming into Syntactically Similar Ones

```c
int sum1(int x){
    int i=0, z=0;
    for( i=0 ; i<=x ; i++ ){
        z += i;
    }
    return z;
}
```

```c
int sum1(int y){
    int z=0, j=0, i=0;
    for( i=0 ; i <= y ; i++ ){
        z += i;
    }
    return z;
}
```

The above are transformed into (resp.)

\[
\mathcal{R}_{\text{sum1}} = \mathcal{R}_{\text{plus}} \cup \left\{ \begin{array}{l}
\text{sum1}(x) \rightarrow U_1(x,0,0) \\
U_1(x,i,z) \rightarrow U_1(x,s(i),\text{plus}(z,i)) \quad [i \leq x] \\
U_1(x,i,z) \rightarrow z \quad [\neg (i \leq x)]
\end{array} \right. 
\]

\[
\mathcal{R}'_{\text{sum1}} = \mathcal{R}_{\text{plus}} \cup \left\{ \begin{array}{l}
\text{sum1}(y) \rightarrow U_2(y,0,0,0) \\
U_2(y,z,j,i) \rightarrow U_2(y,\text{plus}(z,i),j,s(i)) \quad [i \leq y] \\
U_2(y,z,j,i) \rightarrow z \quad [\neg (i \leq y)]
\end{array} \right. 
\]
Equivalence of sum1 and sum1

\( \mathcal{R}_{\text{sum1}} = \mathcal{R}_{\text{plus}} \cup \left\{ \begin{array}{l} \text{sum1}(x) \rightarrow U_1(x, 0, 0) \\ U_1(x, i, z) \rightarrow U_1(x, s(i), \text{plus}(z, i)) \quad \lceil i \leq x \rceil \\ U_1(x, i, z) \rightarrow z \quad \lceil \neg(i \leq x) \rceil \end{array} \right\} \)

\( \mathcal{R}'_{\text{sum1}} = \mathcal{R}_{\text{plus}} \cup \left\{ \begin{array}{l} \text{sum1}(y) \rightarrow U_2(y, 0, 0, 0) \\ U_2(y, z, j, i) \rightarrow U_2(y, \text{plus}(z, i), j, s(i)) \quad \lceil i \leq y \rceil \\ U_2(y, z, j, i) \rightarrow z \quad \lceil \neg(i \leq y) \rceil \end{array} \right\} \)

- sum1 of \( \mathcal{R}_{\text{sum1}} \) and sum1 of \( \mathcal{R}'_{\text{sum1}} \) are equivalent.
Equivalence of sum1 and sum1

\[ R_{\text{sum1}} = R_{\text{plus}} \cup \begin{cases} \text{sum1}(x) \rightarrow U_1(x, 0, 0) \\ U_1(x, i, z) \rightarrow U_1(x, s(i), \text{plus}(z, i)) \ [i \leq x] \\ U_1(x, i, z) \rightarrow z \ [\neg (i \leq x)] \end{cases} \]

\[ R'_{\text{sum1}} = R_{\text{plus}} \cup \begin{cases} \text{sum1}(y) \rightarrow U_2(y, 0, 0, 0) \\ U_2(y, z, j, i) \rightarrow U_2(y, \text{plus}(z, i), j, s(i)) \ [i \leq y] \\ U_2(y, z, j, i) \rightarrow z \ [\neg (i \leq y)] \end{cases} \]

- sum1 of \( R_{\text{sum1}} \) and sum1 of \( R'_{\text{sum1}} \) are equivalent.
- \( R_{\text{sum1}} \) is a tree homomorphic image of \( R'_{\text{sum1}} \):
Tree Homomorphism $H$ from $F$ to $\mathcal{T}(F', \mathcal{V})$ [TATA, 07]

- A mapping such that $H(f) \in \mathcal{T}(F', \{x_1, \cdots, x_n\})$ for any $n$-ary $f \in F$.

- Extended to $\mathcal{T}(F \cup G, \mathcal{V})$ as follows:
  - $H(x) = x$ for $x \in \mathcal{V}$,
  - $H(f(t_1, \cdots, t_n)) = H(f)\{x_i \mapsto H(t_i) \mid 1 \leq i \leq n\}$ for $f \in F$, and
  - $H(g(t_1, \cdots, t_n)) = g(H(t_1), \cdots, H(t_n))$ for $g \in G$.

- Extended to constrained TRSs as follows:
  $$H(R) = \{H(l) \rightarrow H(r) \left[ H(\phi) \right] \mid l \rightarrow r \left[ \phi \right] \in R\}$$

  In this talk, we only consider $H$ s.t. $H(\phi) = \phi$.

- **Linear** if $H(f)$ is linear for all $f \in F$. 

Equivalence of sum1 and sum1

\[ R_{\text{sum1}} = R_{\text{plus}} \cup \begin{cases} \text{sum1}(x) \to U_1(x, 0, 0) \\ U_1(x, i, z) \to U_1(x, s(i), \text{plus}(z, i)) \text{ if } i \leq x \\ U_1(x, i, z) \to z \text{ if } \neg(i \leq x) \end{cases} \]

\[ R'_{\text{sum1}} = R_{\text{plus}} \cup \begin{cases} \text{sum1}(y) \to U_2(y, 0, 0, 0) \\ U_2(y, z, j, i) \to U_2(y, \text{plus}(z, i), j, s(i)) \text{ if } i \leq y \\ U_2(y, z, j, i) \to z \text{ if } \neg(i \leq y) \end{cases} \]

- sum1 of \( R_{\text{sum1}} \) and sum1 of \( R'_{\text{sum1}} \) are equivalent.
- \( R_{\text{sum1}} \) is a tree homomorphic image of \( R'_{\text{sum1}} \):
  - Let \( H \) be a linear tree homomorphism \( H \) s.t.
    - \( H(\text{sum1}) = \text{sum1}(x_1) \)
    - \( H(U_2) = U_1(x_1, x_4, x_2) \)
    - \( H(f) = f(x_1, \cdots, x_n) \) for other \( n \)-ary symbols \( f \).
  - Then \( R_{\text{sum1}} = H(R'_{\text{sum1}}) \).
Sufficient Condition for Equivalence between Functions

**Theorem**

Let

- $R_0$ and $R_1$ be constrained TRSs over $(\mathcal{F}_0, \mathcal{G}, \mathcal{P}, S)$ and $(\mathcal{F}_1, \mathcal{G}, \mathcal{P}, S)$, resp., obtained from C functions,
- $H$ be a tree homomorphism from $\mathcal{D}_{R_1}$ to $\mathcal{T}(\mathcal{F}_0 \cup \mathcal{G}, \mathcal{V})$.

Then, $f_0 \in \mathcal{D}_{R_0}$ and $f_1 \in \mathcal{D}_{R_1}$ are equivalent if all of the following hold:

- $R_0 = H(R_1)$,
- $\text{arity}(f_0) = \text{arity}(f_1)$,
- $H(f_1) = f_0(x_1, \cdots, x_{\text{arity}(f_0)})$,
- $H$ is linear, and
- $H$ is injective w.r.t. root symbols.

The number of $H$ is finite, and thus, this sufficient condition is decidable.
Experiments

- Two exercises of C Programming Exercise Class in 2006
  - Write functions `sum` (fib, resp.) to, given \( n \), compute \( \sum_{i=0}^{n} i \) (the \( n \)-th Fibonacci number, resp.), without recursive calls.
- We succeeded in transforming 59 and 21 programs into constrained TRSs.
Experiments

- Two exercises of C Programming Exercise Class in 2006
  - Write functions sum (fib, resp.) to, given \( n \), compute \( \sum_{i=0}^{n} i \) (the \( n \)-th Fibonacci number, reps.), without recursive calls.

- We succeeded in transforming 59 and 21 programs into constrained TRSs.

- After simplifying them \((P_1, \cdots, P_k)\), we grouped them as follows:
  - if \( P_i \) belongs to Group \( j \), then the index \( i \) is minimum or \( \exists k < i \). \( P_k \) belongs to Group \( j \) and \( P_k \) is a tree homomorphic image of \( P_i \).
Experiments

- Two exercises of C Programming Exercise Class in 2006
  - Write functions sum (fib, resp.) to, given $n$, compute $\sum_{i=0}^{n} i$ (the $n$-th Fibonacci number, reps.), without recursive calls.
- We succeeded in transforming 59 and 21 programs into constrained TRSs.
- After simplifying them ($P_1, \ldots, P_k$), we grouped them as follows:
  - if $P_i$ belongs to Group $j$, then the index $i$ is minimum or $\exists k < i$. $P_k$ belongs to Group $j$ and $P_k$ is a tree homomorphic image of $P_i$.

<table>
<thead>
<tr>
<th></th>
<th>#</th>
<th>groups</th>
<th>total time</th>
<th>ave. time</th>
</tr>
</thead>
<tbody>
<tr>
<td>sum</td>
<td>59</td>
<td>25</td>
<td>16.7 sec (2412 checks)</td>
<td>6.9 msec/check</td>
</tr>
<tr>
<td>fib</td>
<td>21</td>
<td>21</td>
<td>3.0 sec (420 checks)</td>
<td>7.1 msec/check</td>
</tr>
</tbody>
</table>

machine spec.: Athlon 64 X2 4800+ (2.4 GHz/L2cache 2 * 1 MB), 4GB memory
Conclusion

- The C programming exercise class is not so boring.
- The framework is applicable to 1 exercise only.
- Future work:
  - apply the framework to programs with arrays, pointers, etc.