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).

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).

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,
  - $\mathcal{S}$ 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

$$l \to r \llbracket \phi \rrbracket$$

  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 ϕ is a formula over $\mathcal{G}, \mathcal{P}, \mathcal{V}$.

- $C[l\sigma] \to_{\mathcal{R}} C[r\sigma]$ iff
  $l \to r \llbracket \phi \rrbracket \in \mathcal{R}$, $\forall x \in \mathtt{fv}(\phi).\ \sigma(x) \in T(\mathcal{G}, \mathcal{V})$, and $\sigma(\phi)$ is $\mathcal{S}$-valid.

Example of Constrained TRSs (LIA constraints)

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

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

$$\text{sum}(\text{s}^{10}(0)) \to_{\mathcal{R}} \text{add}(\text{s}^{10}(0), \text{sum}(\text{s}^{9}(0))) \to_{\mathcal{R}}^{*} \text{s}^{55}(0)$$

Example: outline of verifying C programs [Furuichi et al, 08]

C program

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

Specification

$$\begin{cases} \mathrm{sum}(x) = 0 \quad \text{if } x \leq 0 \\ \mathrm{sum}(\mathrm{s}(x)) = \mathrm{s}(x) + \mathrm{sum}(x) \\ \qquad\qquad \text{if } x \geq 0 \end{cases}$$

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

$$R_{\mathrm{sum1}} = R_{\mathrm{plus}} \cup \left\{ \begin{array}{l} \mathrm{sum1}(x) \to \mathrm{U}_1(x, 0, 0) \\ \mathrm{U}_1(x, i, z) \to \mathrm{U}_1(x, \mathrm{s}(i), \mathrm{plus}(z, i)) \; [\![i \leq x]\!] \\ \mathrm{U}_1(x, i, z) \to z \; [\![\neg(i \leq x)]\!] \end{array} \right\}$$

$$R_{\mathrm{sum}} = \quad \ldots$$

$$R_{\mathrm{plus}} = \quad \{ \mathrm{plus}(0, y) \to y \quad \mathrm{plus}(\mathrm{s}(x), y) \to \mathrm{s}(\mathrm{plus}(x, y)) \quad \ldots \}$$

2. If $\mathrm{sum1}(x) \approx \mathrm{sum}(x)$ is an inductive theorem of $R_{\mathrm{sum1}} \cup R_{\mathrm{sum}}$, then sum1 satisfies the specification on sum.

Our Inductive Theorem Prover

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

Why Implemented an SMT Solver?

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

$$\forall x.\ \forall x_2.\ \forall i.\ \forall x_3.\ ((x + 1 > i \wedge x_2 + 1 > x_3 \wedge x = x_2 \wedge i + 1 = x_3) \implies -x_3 > x_7)$$

- Satisfiability of LIA formulas is decidable.
- Yices, CVC3, Z3 return "unknown".

Rewriting Induction for Constrained Equations [Sakata et al, 09]

If

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

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

Simplification $(\mathcal{E} \cup \{s \simeq C[l\sigma] \llbracket \phi \rrbracket\}, \mathcal{H}) \vdash (\mathcal{E} \cup \{s \simeq C[r\sigma] \llbracket \phi \rrbracket\}, \mathcal{H})$
where $l \to r \llbracket \psi \rrbracket \in \mathcal{R} \cup \mathcal{H}$, ϕ is $\mathcal{S}$-sat, and $\phi \Rightarrow \sigma(\psi)$ is $\mathcal{S}$-valid.

EQ-Deletion $(\mathcal{E} \cup \{C[s] \approx C[t] \llbracket \phi \rrbracket\}, \mathcal{H})$
$\vdash (\mathcal{E} \cup \{C[s] \approx C[t] \llbracket \phi \wedge s \neq t \rrbracket\}, \mathcal{H})$
where $s, t \in \mathcal{T}(\mathcal{G}, \mathcal{V})$ and $\mathcal{V}ar(s,t) \subseteq \mathtt{fv}(\phi)$.

Deletion $(\mathcal{E} \cup \{s \approx t \llbracket \phi \rrbracket\}, \mathcal{H}) \vdash (\mathcal{E}, \mathcal{H})$
where $s \equiv t$ or ϕ is not $\mathcal{S}$-sat.

Expansion $(\mathcal{E} \cup \{s \approx t \llbracket \phi \rrbracket\}, \mathcal{H})$
$\vdash (\mathcal{E} \cup Expd_p(s \to t \llbracket \phi \rrbracket), \mathcal{H} \cup \{s \to t \llbracket \phi \rrbracket\})$
where $\mathcal{R} \cup \mathcal{H} \cup \{s \to t \llbracket \phi \rrbracket\}$ terminates, p is an $\mathcal{R}$-complete occurrence, and $Expd_p(s \to t \llbracket \phi \rrbracket)$ is the set of critical pairs between $s \to t \llbracket \phi \rrbracket$ and rules in $\mathcal{R}$ at position p of s.

Standard Strategy for Inferences

Given $\mathcal{R}$ 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

$$\mathcal{R}_{\mathsf{sum1}} \cup \mathcal{R}_{\mathsf{sum}} = \left\{ \begin{array}{lll} (1) & \mathsf{sum1}(x) \to \mathsf{U}_1(x, \mathsf{s}(0), 0) & \\ (2) & \mathsf{U}_1(x, i, z) \to \mathsf{U}_1(x, \mathsf{s}(i), \mathsf{plus}(z, i)) & \llbracket\, i \leq x \,\rrbracket \\ (3) & \mathsf{U}_1(x, i, z) \to z & \llbracket\, \neg i \leq x \,\rrbracket \\ (4) & \mathsf{sum}(x) \to 0 & \llbracket\, x \leq 0 \,\rrbracket \\ (5) & \mathsf{sum}(\mathsf{s}(x)) \to \mathsf{plus}(\mathsf{sum}(x), \mathsf{s}(x)) & \llbracket\, x \geq 0 \,\rrbracket \end{array} \right\} \cup \mathcal{R}_{\mathsf{plus}}$$

$$\mathcal{E} = \{ \quad \mathsf{sum1}(x) \approx \mathsf{sum}(x) \quad \}$$

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

$$\begin{array}{rl} \mathsf{plus}(\mathsf{U}_1(x, \mathsf{s}(0), 0), \mathsf{s}(x)) \approx \mathsf{U}_1(\mathsf{s}(x), \mathsf{s}^2(0), \mathsf{plus}(0, \mathsf{s}(0))) & \llbracket\, 0 \leq \mathsf{s}(x) \,\rrbracket \\ \mathsf{plus}(\mathsf{U}_1(x, \mathsf{s}^2(0), \mathsf{s}(0)), \mathsf{s}(x)) \approx \mathsf{U}_1(\mathsf{s}(x), \mathsf{s}^3(0), \mathsf{plus}(\mathsf{s}(0), \mathsf{s}^2(0))) & \llbracket\, \mathsf{s}^2(0) \leq \mathsf{s}(x) \\ \vdots & \end{array}$$

A desired lemma is

$$\mathsf{plus}(\mathsf{U}_1(x, i, z), \mathsf{s}(x)) \approx \mathsf{U}_1(\mathsf{s}(x), \mathsf{s}(i), \mathsf{plus}(i, z)) \ \llbracket\, i \leq \mathsf{s}(x) \,\rrbracket$$

How to Get a (Candidate of) Lemma Equations

$$\mathcal{R}_{\mathsf{sum1}} \cup \mathcal{R}_{\mathsf{sum}} = \left\{ \begin{array}{ll} (1) \quad \mathsf{sum1}(x) \to \mathsf{U}_1(x, \mathsf{s}(0), 0) & \\ (2)\ \mathsf{U}_1(x, i, z) \to \mathsf{U}_1(x, \mathsf{s}(i), \mathsf{plus}(z, i)) & [\![\ i \leq x\ ]\!] \\ (3)\ \mathsf{U}_1(x, i, z) \to z & [\![\ \neg i \leq x\ ]\!] \\ (4) \quad \mathsf{sum}(x) \to 0 & [\![\ x \leq 0\ ]\!] \\ (5)\ \mathsf{sum}(\mathsf{s}(x)) \to \mathsf{plus}(\mathsf{sum}(x), \mathsf{s}(x)) & [\![\ x \geq 0\ ]\!] \end{array} \right\} \cup \mathcal{R}_{\mathsf{plus}}$$

$$\mathcal{E} = \{ \quad \mathsf{sum1}(x) \approx \mathsf{sum}(x) \quad \}$$

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

$$\begin{array}{rl} \mathsf{plus}(\mathsf{U}_1(x, \mathsf{s}(0), 0), \mathsf{s}(x)) \approx \mathsf{U}_1(\mathsf{s}(x), \mathsf{s}^2(0), \mathsf{plus}(0, \mathsf{s}(0))) & [\![\ 0 \leq \mathsf{s}(x)\ ]\!] \\ \mathsf{plus}(\mathsf{U}_1(x, \mathsf{s}^2(0), \mathsf{s}(0)), \mathsf{s}(x)) \approx \mathsf{U}_1(\mathsf{s}(x), \mathsf{s}^3(0), \mathsf{plus}(\mathsf{s}(0), \mathsf{s}^2(0))) & [\![\ \mathsf{s}^2(0) \leq \mathsf{s}(x) \\ \vdots & \end{array}$$

A desired lemma is

$$\mathsf{plus}(\mathsf{U}_1(x, i, z), \mathsf{s}(x)) \approx \mathsf{U}_1(\mathsf{s}(x), \mathsf{s}(i), \mathsf{plus}(i, z)) \ [\![\ i \leq \mathsf{s}(x)\ ]\!]$$

Strategy with Lemma Discovery [Nakabayashi et al, 09]

Given $\mathcal{R}$ and $\mathcal{E}$, apply the following steps to $(\mathcal{E}, \emptyset)$ until $\mathcal{E}$ 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 $(\{e\}, \emptyset) \vdash \cdots \vdash (\emptyset, \mathcal{H}')$, then add $\mathcal{H}'$ to $\mathcal{H}$ and go to 1,
   - o/w, go to the next.
5. if $\mathcal{E}$ 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

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

```
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

$$\mathcal{R}_{\mathsf{sum1}} = \mathcal{R}_{\mathsf{plus}} \cup \left\{ \begin{array}{c} \mathsf{sum1}(x) \to \mathsf{U}_1(x, 0, 0) \\ \mathsf{U}_1(x, i, z) \to \mathsf{U}_1(x, \mathsf{s}(i), \mathsf{plus}(z, i)) \ [\![i \leq x]\!] \\ \mathsf{U}_1(x, i, z) \to z \ [\![\neg(i \leq x)]\!] \end{array} \right\}$$

Example of Transforming into Syntactically Similar Ones

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

```
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.)

$$\mathcal{R}_{\mathsf{sum1}} = \mathcal{R}_{\mathsf{plus}} \cup \left\{ \begin{array}{r} \mathsf{sum1}(x) \to \mathsf{U}_1(x, 0, 0) \\ \mathsf{U}_1(x, i, z) \to \mathsf{U}_1(x, \mathsf{s}(i), \mathsf{plus}(z, i)) \; [\![i \leq x]\!] \\ \mathsf{U}_1(x, i, z) \to z \; [\![\neg(i \leq x)]\!] \end{array} \right\}$$

$$\mathcal{R}'_{\mathsf{sum1}} = \mathcal{R}_{\mathsf{plus}} \cup \left\{ \begin{array}{r} \mathsf{sum1}(y) \to \mathsf{U}_2(y, 0, 0, 0) \\ \mathsf{U}_2(y, z, j, i) \to \mathsf{U}_2(y, \mathsf{plus}(z, i), j, \mathsf{s}(i)) \; [\![i \leq y]\!] \\ \mathsf{U}_2(y, z, j, i) \to z \; [\![\neg(i \leq y)]\!] \end{array} \right\}$$

Equivalence of sum1 and sum1

$$\mathcal{R}_{\mathsf{sum1}} = \mathcal{R}_{\mathsf{plus}} \cup \left\{ \begin{array}{r} \mathsf{sum1}(x) \to \mathsf{U}_1(x, 0, 0) \\ \mathsf{U}_1(x, i, z) \to \mathsf{U}_1(x, \mathsf{s}(i), \mathsf{plus}(z, i)) \ [\![i \le x]\!] \\ \mathsf{U}_1(x, i, z) \to z \ [\![\neg(i \le x)]\!] \end{array} \right\}$$

$$\mathcal{R}'_{\mathsf{sum1}} = \mathcal{R}_{\mathsf{plus}} \cup \left\{ \begin{array}{r} \mathsf{sum1}(y) \to \mathsf{U}_2(y, 0, 0, 0) \\ \mathsf{U}_2(y, z, j, i) \to \mathsf{U}_2(y, \mathsf{plus}(z, i), j, \mathsf{s}(i)) \ [\![i \le y]\!] \\ \mathsf{U}_2(y, z, j, i) \to z \ [\![\neg(i \le y)]\!] \end{array} \right\}$$

- sum1 of $\mathcal{R}_{\mathsf{sum1}}$ and sum1 of $\mathcal{R}'_{\mathsf{sum1}}$ are equivalent.

Equivalence of sum1 and sum1

$$\mathcal{R}_{\mathsf{sum1}} = \mathcal{R}_{\mathsf{plus}} \cup \left\{ \begin{array}{r} \mathsf{sum1}(x) \to \mathsf{U}_1(x, 0, 0) \\ \mathsf{U}_1(x, i, z) \to \mathsf{U}_1(x, \mathsf{s}(i), \mathsf{plus}(z, i)) \; [\![i \leq x]\!] \\ \mathsf{U}_1(x, i, z) \to z \; [\![\neg(i \leq x)]\!] \end{array} \right\}$$

$$\mathcal{R}'_{\mathsf{sum1}} = \mathcal{R}_{\mathsf{plus}} \cup \left\{ \begin{array}{r} \mathsf{sum1}(y) \to \mathsf{U}_2(y, 0, 0, 0) \\ \mathsf{U}_2(y, z, j, i) \to \mathsf{U}_2(y, \mathsf{plus}(z, i), j, \mathsf{s}(i)) \; [\![i \leq y]\!] \\ \mathsf{U}_2(y, z, j, i) \to z \; [\![\neg(i \leq y)]\!] \end{array} \right\}$$

- sum1 of $\mathcal{R}_{\mathsf{sum1}}$ and sum1 of $\mathcal{R}'_{\mathsf{sum1}}$ are equivalent.
- $\mathcal{R}_{\mathsf{sum1}}$ is a tree homomorphic image of $\mathcal{R}'_{\mathsf{sum1}}$:

Tree Homomorphism H from F to $\mathcal{T}(F', \mathcal{V})$ [TATA, 07]

- A mapping such that $H(f) \in \mathcal{T}(\mathcal{F}', \{x_1, \cdots, x_n\})$ for any n-ary $f \in \mathcal{F}$.
- Extended to $\mathcal{T}(\mathcal{F} \uplus \mathcal{G}, \mathcal{V})$ 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 $f \in \mathcal{F}$, and
  - $H(g(t_1, \cdots, t_n)) = g(H(t_1), \cdots, H(t_n))$ for $g \in \mathcal{G}$.
- Extended to constrained TRSs as follows:

$$H(R) = \{H(l) \to H(r)\ [\![H(\phi)]\!] \mid l \to r\ [\![\phi]\!] \in R\}$$

  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

$$\mathcal{R}_{\mathsf{sum1}} = \mathcal{R}_{\mathsf{plus}} \cup \left\{ \begin{array}{c} \mathsf{sum1}(x) \to \mathsf{U}_1(x, 0, 0) \\ \mathsf{U}_1(x, i, z) \to \mathsf{U}_1(x, \mathsf{s}(i), \mathsf{plus}(z, i)) \ [\![i \leq x]\!] \\ \mathsf{U}_1(x, i, z) \to z \ [\![\neg(i \leq x)]\!] \end{array} \right\}$$

$$\mathcal{R}'_{\mathsf{sum1}} = \mathcal{R}_{\mathsf{plus}} \cup \left\{ \begin{array}{c} \mathsf{sum1}(y) \to \mathsf{U}_2(y, 0, 0, 0) \\ \mathsf{U}_2(y, z, j, i) \to \mathsf{U}_2(y, \mathsf{plus}(z, i), j, \mathsf{s}(i)) \ [\![i \leq y]\!] \\ \mathsf{U}_2(y, z, j, i) \to z \ [\![\neg(i \leq y)]\!] \end{array} \right\}$$

- sum1 of $\mathcal{R}_{\mathsf{sum1}}$ and sum1 of $\mathcal{R}'_{\mathsf{sum1}}$ are equivalent.
- $\mathcal{R}_{\mathsf{sum1}}$ is a tree homomorphic image of $\mathcal{R}'_{\mathsf{sum1}}$:
  - Let H be a linear tree homomorphism H s.t.
    - ⋆ $H(\mathsf{sum1}) = \mathsf{sum1}(x_1)$
    - ⋆ $H(\mathsf{U}_2) = \mathsf{U}_1(x_1, x_4, x_2)$
    - ⋆ $H(f) = f(x_1, \cdots, x_n)$ for other n-ary symbols f.
  - Then $\mathcal{R}_{\mathsf{sum1}} = H(\mathcal{R}'_{\mathsf{sum1}})$.

Sufficient Condition for Equivalence between Functions

Theorem

Let

- R_0 *and* R_1 *be constrained TRSs over* $(\mathcal{F}_0, \mathcal{G}, \mathcal{P}, \mathcal{S})$ *and* $(\mathcal{F}_1, \mathcal{G}, \mathcal{P}, \mathcal{S})$*, 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:*

- $R_0 = H(R_1)$*,*
- $\mathtt{arity}(f_0) = \mathtt{arity}(f_1)$*,*
- $H(f_1) = f_0(x_1, \cdots, x_{\mathtt{arity}(f_0)})$*,*
- 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 $\Sigma_{i=0}^{n} i$ (the n-th Fibonacci number, reps.), 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 $\Sigma_{i=0}^{n} i$ (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 $\Sigma_{i=0}^{n} i$ (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.

	#	groups	total time	ave. time
sum	59	25	16.7 sec (2412 checks)	6.9 msec/check
fib	21	21	3.0 sec (420 checks)	7.1 msec/check

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.