A machine-comparison between two program-synthesis techniques

– Proof-mining a *classical* proof of Dickson-2-2 Lemma –

Mircea Dan HERNEST

Project LogiCal – Paris, FRANCE and GKLI – Munich, GERMANY

LAMA-Chambery Workshop, 13-14 April 2005

Outline

Outline

Outline

Outline

Why $\forall x \exists y G(x, y)$ specifications ? [$G \equiv$ Goal Formula]

- Specifications describe wanted behavior for our Program.
- Programs have inputs – x and outputs – y .
- Therefore specifications are formulas $\forall x \exists y G(x, y)$ where
- $G(x, y)$ is a formula describing the desired relationship between the given input x and the desired output y .
- Exists proof $\mathcal{P}$ of $\forall x \exists y G(x, y)$ in some logical system $\mathcal{S}$.
- We want to be able to *uniformly* produce by an Algorithm a program t which *realizes* the given specification, i.e., $\forall x G(x, t(x))$ is provable in some (other) logical system $\mathcal{S}'$.
- Such Algorithms taking inputs $\mathcal{P}$ are Program-Extraction procedures and the term t is called extracted program.

Why $\forall x \exists y G(x, y)$ specifications ? [$G \equiv$ Goal Formula]

- Specifications describe wanted behavior for our Program.
- Programs have inputs – x and outputs – y .
- Therefore specifications are formulas $\forall x \exists y G(x, y)$ where
- $G(x, y)$ is a formula describing the desired relationship between the given input x and the desired output y .
- Exists proof $\mathcal{P}$ of $\forall x \exists y G(x, y)$ in some logical system $\mathcal{S}$.
- We want to be able to *uniformly* produce by an Algorithm a program t which *realizes* the given specification, i.e., $\forall x G(x, t(x))$ is provable in some (other) logical system $\mathcal{S}'$.
- Such Algorithms taking inputs $\mathcal{P}$ are *Program Extraction procedures* and the term t is called *extracted program* .

Why $\forall x \exists y G(x, y)$ specifications ? [$G \equiv$ Goal Formula]

- Specifications describe wanted behavior for our Program.
- Programs have inputs – x and outputs – y .
- Therefore specifications are formulas $\forall x \exists y G(x, y)$ where
- $G(x, y)$ is a formula describing the desired relationship between the given input x and the desired output y .
- Exists proof $\mathcal{P}$ of $\forall x \exists y G(x, y)$ in some logical system $\mathcal{S}$.
- We want to be able to *uniformly* produce by an Algorithm a program t which *realizes* the given specification, i.e., $\forall x G(x, t(x))$ is provable in some (other) logical system $\mathcal{S}'$.
- Such Algorithms taking inputs $\mathcal{P}$ are *Program Extraction procedures* and the term t is called *extracted program* .

Why $\forall x \exists y G(x, y)$ specifications ? [$G \equiv$ Goal Formula]

- Specifications describe wanted behavior for our Program.
- Programs have inputs – x and outputs – y .
- Therefore specifications are formulas $\forall x \exists y G(x, y)$ where
- $G(x, y)$ is a formula describing the desired relationship between the given input x and the desired output y .
- Exists proof $\mathcal{P}$ of $\forall x \exists y G(x, y)$ in some logical system $\mathcal{S}$.
- We want to be able to *uniformly* produce by an Algorithm a program t which *realizes* the given specification, i.e., $\forall x G(x, t(x))$ is provable in some (other) logical system $\mathcal{S}'$.
- Such Algorithms taking inputs $\mathcal{P}$ are *Program Extraction procedures* and the term t is called *extracted program* .

Classical versus Constructive proofs

Recall that $\mathcal{P}$ is a proof of $\forall x\, \exists y\, G(x,y)$ in the logical system $\mathcal{S}$!

- G can be arbitrary only when $\mathcal{S}$ is constructive, otherwise ...
- *Constructive* means *intuitionistic* plus *Markov's Principle* :
  $\neg\neg\exists z\, G_0(z) \to \exists z\, G_0(z)$ or $\underbrace{\exists^{cl} z}_{\neg\forall z\,\neg}\, G_0(z) \to \exists z\, G_0(z)$
- Only extraction techniques based on Gödel's **Dialectica** interpretation allow Markov's Principle (as axiom) in $\mathcal{S}$.
- *Refined* ***A****–translation* first brings the *Minimal Logic* proof [*minimal* $\equiv$ intuitionistic $\setminus\ \bot \to F$] of $\forall x\, \exists^{cl} y\, G(x,y)$ to a corresponding intuitionistic proof of $\forall x\, \exists y\, G(x,y)$
- Kreisel's **Modified Realizability** applies to the latter .
- G can be a *goal formula* – i.e., more than quantifier-free .

Classical versus Constructive proofs

Recall that $\mathcal{P}$ is a proof of $\forall x\, \exists y\, G(x,y)$ in the logical system $\mathcal{S}$!

- G can be arbitrary only when $\mathcal{S}$ is constructive, otherwise ...
- *Constructive* means *intuitionistic* plus *Markov's Principle* :
  $\neg\neg\exists z\, G_0(z) \to \exists z\, G_0(z)$ or $\underbrace{\exists^{cl} z}_{\neg\forall z\,\neg}\, G_0(z) \to \exists z\, G_0(z)$
- Only extraction techniques based on Gödel's **Dialectica** interpretation allow Markov's Principle (as axiom) in $\mathcal{S}$.
- *Refined* ***A****–translation* first brings the *Minimal Logic* proof [*minimal* $\equiv$ intuitionistic $\setminus\ \bot \to F$] of $\forall x\, \exists^{cl} y\, G(x,y)$ to a corresponding intuitionistic proof of $\forall x\, \exists y\, G(x,y)$
- Kreisel's **Modified Realizability** applies to the latter .
- G can be a *goal formula* – i.e., more than quantifier-free .

Classical versus Constructive proofs

Recall that $\mathcal{P}$ is a proof of $\forall x\, \exists y\, G(x,y)$ in the logical system $\mathcal{S}$!

- G can be arbitrary only when $\mathcal{S}$ is constructive, otherwise ...
- *Constructive* means *intuitionistic* plus *Markov's Principle* :
  $\neg\neg\exists z\, G_0(z) \to \exists z\, G_0(z)$ or $\underbrace{\exists^{cl} z}_{\neg\forall z\,\neg}\, G_0(z) \to \exists z\, G_0(z)$
- Only extraction techniques based on Gödel's **Dialectica** interpretation allow Markov's Principle (as axiom) in $\mathcal{S}$.
- *Refined* ***A****–translation* first brings the *Minimal Logic* proof [*minimal* $\equiv$ intuitionistic $\setminus\ \bot \to F$] of $\forall x\, \exists^{cl} y\, G(x,y)$ to a corresponding intuitionistic proof of $\forall x\, \exists y\, G(x,y)$
- Kreisel's **Modified Realizability** applies to the latter .
- G can be a *goal formula* – i.e., more than quantifier-free .

Classical versus Constructive proofs

Recall that $\mathcal{P}$ is a proof of $\forall x\, \exists y\, G(x,y)$ in the logical system $\mathcal{S}$!

- G can be arbitrary only when $\mathcal{S}$ is constructive, otherwise ...
- *Constructive* means *intuitionistic* plus *Markov's Principle* :
  $\neg\neg\exists z\, G_0(z) \to \exists z\, G_0(z)$ or $\underbrace{\exists^{cl} z}_{\neg\forall z\,\neg}\, G_0(z) \to \exists z\, G_0(z)$
- Only extraction techniques based on Gödel's **Dialectica** interpretation allow Markov's Principle (as axiom) in $\mathcal{S}$.
- *Refined* ***A****–translation* first brings the *Minimal Logic* proof [*minimal* $\equiv$ intuitionistic $\setminus\ \bot \to F$] of $\dfrac{\forall x\, \exists^{cl} y\, G(x,y)}{\forall x\, \exists y\, G(x,y)}$ to a corresponding intuitionistic proof of
- Kreisel's **Modified Realizability** applies to the latter .
- G can be a *goal formula* – i.e., more than quantifier-free .

Classical versus Constructive proofs

Recall that $\mathcal{P}$ is a proof of $\forall x\, \exists y\, G(x,y)$ in the logical system $\mathcal{S}$!

- G can be arbitrary only when $\mathcal{S}$ is constructive, otherwise ...
- *Constructive* means *intuitionistic* plus *Markov's Principle* :
  $\neg\neg\exists z\, G_0(z) \to \exists z\, G_0(z)$ or $\underbrace{\exists^{cl} z}_{\neg\forall z\,\neg}\, G_0(z) \to \exists z\, G_0(z)$
- Only extraction techniques based on Gödel's **Dialectica** interpretation allow Markov's Principle (as axiom) in $\mathcal{S}$.
- *Refined* ***A****–translation* first brings the *Minimal Logic* proof [*minimal* $\equiv$ intuitionistic $\setminus\ \bot \to F$] of $\dfrac{\forall x\, \exists^{cl} y\, G(x,y)}{\forall x\, \exists y\, G(x,y)}$ to a corresponding intuitionistic proof of
- Kreisel's **Modified Realizability** applies to the latter .
- G can be a *goal formula* – i.e., more than quantifier-free .

Outline

The Dickson-2-2 Lemma as a Program Specification

- "For any two functions $f, g : \mathbb{N} \mapsto \mathbb{N}$ defined within the set of natural numbers $\mathbb{N}$ there exist indexes $i < j$ such that both $f(i) \leq f(j)$ and $g(i) \leq g(j)$.
- The simple algorithm which enumerates pairs (i, j) with $i < j$ and checks for given f, g whether indeed $f(i) \leq f(j)$ and $g(i) \leq g(j)$ – *is not very efficient* for inputs f_n, g_n for which the first solution (i_n, j_n) has i_n bigger than a lower bound 2^n, 2^{2^n}, $2^{2^{2^n}}$, ... etc.
- Various approaches have been proposed over the last years for synthesizing programs computing solutions (i, j) as functionals of (f, g) from the *classical* proof of Dickson-2-2 Lemma (which is shorter and simpler to give).

The Dickson-2-2 Lemma as a Program Specification

- "For any two functions $f, g : \mathbb{N} \mapsto \mathbb{N}$ defined within the set of natural numbers $\mathbb{N}$ there exist indexes $i < j$ such that both $f(i) \leq f(j)$ and $g(i) \leq g(j)$.
- The simple algorithm which enumerates pairs (i, j) with $i < j$ and checks for given f, g whether indeed $f(i) \leq f(j)$ and $g(i) \leq g(j)$ – *is not very efficient* for inputs f_n, g_n for which the first solution (i_n, j_n) has i_n bigger than a lower bound 2^n, 2^{2^n}, $2^{2^{2^n}}$, ... etc.
- Various approaches have been proposed over the last years for synthesizing programs computing solutions (i, j) as functionals of (f, g) from the *classical* proof of Dickson-2-2 Lemma (which is shorter and simpler to give).

The Dickson-2-2 Lemma as a Program Specification

- "For any two functions $f, g : \mathbb{N} \mapsto \mathbb{N}$ defined within the set of natural numbers $\mathbb{N}$ there exist indexes $i < j$ such that both $f(i) \leq f(j)$ and $g(i) \leq g(j)$.
- The simple algorithm which enumerates pairs (i, j) with $i < j$ and checks for given f, g whether indeed $f(i) \leq f(j)$ and $g(i) \leq g(j)$ – *is not very efficient* for inputs f_n, g_n for which the first solution (i_n, j_n) has i_n bigger than a lower bound 2^n, 2^{2^n}, $2^{2^{2^n}}$, ... etc.
- Various approaches have been proposed over the last years for synthesizing programs computing solutions (i, j) as functionals of (f, g) from the *classical* proof of Dickson-2-2 Lemma (which is shorter and simpler to give).

Outline

The Demo

We will now present a comparison of the machine-performance of two such automated program-synthesis algorithms implemented in the proof and program-extraction system MINLOG: check http://www.minlog-system.de

1. The Berger-Buchholz-Schwichtenberg (2002) refined A–translation + Kreisel's (1959) Modified Realizability .
2. Gödel's (1958) functional *Dialectica* interpretation .

We will thereafter attempt to also mention the following:

- U. Kohlenbach's (1993) *monotone* Dialectica interpretation.
- F. Ferreira and P. Oliva's (2004) *bounded* Dialectica interp.
- Our *light* Dialectica interpretation (accepted at CSL'2005).
- C. Raffalli's (2003) heuristic technique – best practical (i.e., machine) results for the more general Dickson-m-n.

The Demo

We will now present a comparison of the machine-performance of two such automated program-synthesis algorithms implemented in the proof and program-extraction system MINLOG: check http://www.minlog-system.de

1. The Berger-Buchholz-Schwichtenberg (2002) refined **A**–translation + Kreisel's (1959) Modified Realizability .
2. Gödel's (1958) functional *Dialectica* interpretation .

We will thereafter attempt to also mention the following:

- U. Kohlenbach's (1993) *monotone* Dialectica interpretation.
- F. Ferreira and P. Oliva's (2004) *bounded* Dialectica interp.
- Our *light* Dialectica interpretation (accepted at CSL'2005).
- C. Raffalli's (2003) heuristic technique – best practical (i.e., machine) results for the more general Dickson-m-n.

The Demo

We will now present a comparison of the machine-performance of two such automated program-synthesis algorithms implemented in the proof and program-extraction system MINLOG: check http://www.minlog-system.de

1. The Berger-Buchholz-Schwichtenberg (2002) refined **A**–translation + Kreisel's (1959) Modified Realizability .
2. Gödel's (1958) functional *Dialectica* interpretation .

We will thereafter attempt to also mention the following:

- U. Kohlenbach's (1993) *monotone* Dialectica interpretation.
- F. Ferreira and P. Oliva's (2004) *bounded* Dialectica interp.
- Our *light* Dialectica interpretation (accepted at CSL'2005).
- C. Raffalli's (2003) heuristic technique – best practical (i.e., machine) results for the more general Dickson-m-n.

The Demo

We will now present a comparison of the machine-performance of two such automated program-synthesis algorithms implemented in the proof and program-extraction system MINLOG: check http://www.minlog-system.de

1. The Berger-Buchholz-Schwichtenberg (2002) refined **A**–translation + Kreisel's (1959) Modified Realizability .
2. Gödel's (1958) functional *Dialectica* interpretation .

We will thereafter attempt to also mention the following:

- U. Kohlenbach's (1993) *monotone* Dialectica interpretation.
- F. Ferreira and P. Oliva's (2004) *bounded* Dialectica interp.
- Our *light* Dialectica interpretation (accepted at CSL'2005).
- C. Raffalli's (2003) heuristic technique – best practical (i.e., machine) results for the more general Dickson-m-n.

Short List of related Papers I

U. Berger, W. Buchholz, and H. Schwichtenberg.
Refined program extraction from classical proofs.
Annals of Pure and Applied Logic, 114:3–25, 2002.

M.-D. Hernest.
A comparison between two techniques of program extraction from classical proofs.
In M. Baaz, J. Makovsky, and A. Voronkov, editors, *CSL 2003: Extended Posters*, vol. VIII of *Kurt Gödel Society's Collegium Logicum*, pp. 99–102. Springer Verlag, 2004.

M.-D. Hernest and U. Kohlenbach.
A complexity analysis of functional interpretations.
Theoretical Computer Science, 338(1-3):200–246, 2005.

Short List of related Papers II

C. Raffalli.
Getting results from programs extracted from classical proofs.
Theoretical Computer Science, 323(1-3):49–70, 2004.

C. Paulin-Mohring and B. Werner.
Synthesis of ML programs in the system Coq.
Journal of Symbolic Computation, 15(5/6):607–640, 1993.

U. Kohlenbach and P. Oliva.
Proof mining: a systematic way of analysing proofs in Mathematics.
Proc. of the Steklov Institute of Mathematics, 242:136–164, 2003.