Specification and Verification of Algorithms from Computational Logic

Bachelor Thesis Report

Johannes Grünberger

Johannes Kepler University, Linz, Austria
j.gruenberger@live.at

November 14, 2019

Overview

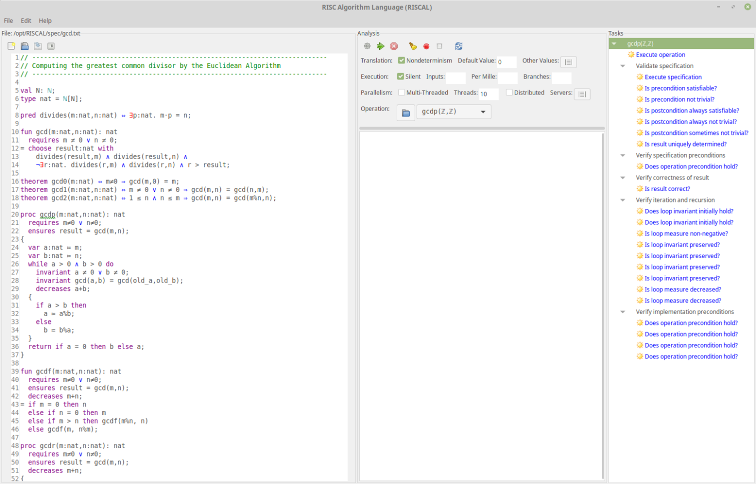
- Bachelor Thesis (Start in October 2019)
- **Goal:** Specify and Verify algorithms from computational logic with the RISC Algorithm Language.
- Content of this presentation:
  - RISC Algorithm Language
  - Propositional Logic
    - Goal: Syntax and Semantics
    - Goal: Substitution
    - Goal: Normal Forms
    - Goal: SAT Solving
  - First-Order Logic
    - Goal: Syntax and Semantics
    - Goal: Prenex Normal Form and Skolemnization
  - Summary

RISC Algorithm Language (RISCAL)

RISCAL is a **specification language** and **associated software tool** to ...

- Describe mathematical theories and algorithms
- Specify the behavior of algorithms:
  - Preconditions and Postconditions
  - Termination Measures
  - Loop Invariants
- Verify this theories over **finite domains**

RISCAL II



RISC Algorithm Language (RISCAL)
File Edit Help
File: /opt/RISCAL/spec/gcd.txt
// --
// Computing the greatest common divisor by the Euclidean Algorithm
// --
val N: ℕ;
type nat = ℕ[N];
pred divides(m:nat,n:nat) ⇔ ∃p:nat. m·p = n;
fun gcd(m:nat,n:nat): nat
requires m ≠ 0 ∨ n ≠ 0;
= choose result:nat with
divides(result,m) ∧ divides(result,n) ∧
¬∃r:nat. divides(r,m) ∧ divides(r,n) ∧ r > result;
theorem gcd0(m:nat) ⇔ m≠0 ⇒ gcd(m,0) = m;
theorem gcd1(m:nat,n:nat) ⇔ m ≠ 0 ∨ n ≠ 0 ⇒ gcd(m,n) = gcd(n,m);
theorem gcd2(m:nat,n:nat) ⇔ 1 ≤ n ∧ n ≤ m ⇒ gcd(m,n) = gcd(m%n,n);
proc gcdp(m:nat,n:nat): nat
requires m≠0 ∨ n≠0;
ensures result = gcd(m,n);
{
var a:nat = m;
var b:nat = n;
while a > 0 ∧ b > 0 do
invariant a ≠ 0 ∨ b ≠ 0;
invariant gcd(a,b) = gcd(old_a,old_b);
decreases a+b;
{
if a > b then
a = a%b;
else
b = b%a;
}
return if a = 0 then b else a;
}
fun gcdf(m:nat,n:nat): nat
requires m≠0 ∨ n≠0;
ensures result = gcd(m,n);
decreases m+n;
= if m = 0 then n
else if n = 0 then m
else if m > n then gcdf(m%n, n)
else gcdf(m, n%m);
proc gcdr(m:nat,n:nat): nat
requires m≠0 ∨ n≠0;
ensures result = gcd(m,n);
decreases m+n;
{
Analysis
Translation: Nondeterminism Default Value: 0 Other Values:
Execution: Silent Inputs: Per Mille: Branches:
Parallelism: Multi-Threaded Threads: 10 Distributed Servers:
Operation: gcdp(ℤ,ℤ)
Tasks
gcdp(ℤ,ℤ)
Execute operation
Validate specification
Execute specification
Is precondition satisfiable?
Is precondition not trivial?
Is postcondition always satisfiable?
Is postcondition always not trivial?
Is postcondition sometimes not trivial?
Is result uniquely determined?
Verify specification preconditions
Does operation precondition hold?
Verify correctness of result
Is result correct?
Verify iteration and recursion
Does loop invariant initially hold?
Does loop invariant initially hold?
Is loop measure non-negative?
Is loop invariant preserved?
Is loop invariant preserved?
Is loop invariant preserved?
Is loop invariant preserved?
Is loop measure decreased?
Is loop measure decreased?
Verify implementation preconditions
Does operation precondition hold?
Does operation precondition hold?
Does operation precondition hold?
Does operation precondition hold?

Goal: Syntax and Semantics of Propositional Logic

The goal is a RISCAL specification containing:

- Data types *Formula* (recursive) and *Valuation*
- A predicate *satisfies* denoting whether a particular valuation satisfies a formula.
- Predicates for derived notions *valid*, *satisfiable*, *logically equivalent*, ...
- Theorems stating the connection between those predicates

Propositional Logic

The logic of **propositions**:

- A proposition must be either **True** or **False** in a particular interpretation.
- Many applications in mathematics and computer science:
  - Mathematical Proof Theory
  - Foundation for First- and Higher-Order Logic
- The foundation for Formal Methods and Automated Theorem Proving

Syntax of Propositonal Formulas

- **Truth Constants:** $\{\mathbb{T}, \mathbb{F}\}$
- **Atoms:** $a \in \mathcal{V}$, for a finite set of variables $\mathcal{V}$.
- **Negations:** $\neg\varphi$, for propositional formula φ
- **Logical Connectives:** $\varphi * \psi$, for propositional formulas φ, ψ, $* \in \{\vee, \wedge, \Rightarrow, \Leftrightarrow\}$
- **Parenthesis:** (φ), for propositional formula φ

Semantics of Logical Connectives

$\mathcal{B}_\neg :=$

	$\mathcal{B}_\neg$
$\mathbb{T}$	$\mathbb{F}$
$\mathbb{F}$	$\mathbb{T}$

$\mathcal{B}_\wedge :=$

$\mathcal{B}_\wedge$	$\mathbb{T}$	$\mathbb{F}$
$\mathbb{T}$	$\mathbb{T}$	$\mathbb{F}$
$\mathbb{F}$	$\mathbb{F}$	$\mathbb{F}$

$\mathcal{B}_\Rightarrow :=$

$\mathcal{B}_\Rightarrow$	$\mathbb{T}$	$\mathbb{F}$
$\mathbb{T}$	$\mathbb{T}$	$\mathbb{F}$
$\mathbb{F}$	$\mathbb{T}$	$\mathbb{T}$

$\mathcal{B}_\vee :=$

$\mathcal{B}_\vee$	$\mathbb{T}$	$\mathbb{F}$
$\mathbb{T}$	$\mathbb{T}$	$\mathbb{T}$
$\mathbb{F}$	$\mathbb{T}$	$\mathbb{F}$

$\mathcal{B}_\Leftrightarrow :=$

$\mathcal{B}_\Leftrightarrow$	$\mathbb{T}$	$\mathbb{F}$
$\mathbb{T}$	$\mathbb{T}$	$\mathbb{F}$
$\mathbb{F}$	$\mathbb{F}$	$\mathbb{T}$

Semantics of Propositonal Formulas

- A **valuation** v maps to every atom a truth value.

$$v : \mathcal{A} \to \{\mathbb{T}, \mathbb{F}\}$$

- The **meaning** $\langle \varphi \rangle_v$ maps to every formula φ a truth value under the valuation v:

$$\begin{aligned}
\langle \mathbb{T} \rangle_v &= \mathbb{T} \\
\langle \mathbb{F} \rangle_v &= \mathbb{F} \\
\langle a \rangle_v &= v(a), \text{ for atom } a \\
\langle \neg\varphi \rangle_v &= \mathcal{B}_\neg(\langle \varphi \rangle_v) \\
\langle \varphi * \psi \rangle_v &= \mathcal{B}_*(\langle \varphi \rangle_v, \langle \psi \rangle_v), \text{ for } * \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}
\end{aligned}$$

- A valuation v **satisfies** a formula φ iff $\langle \varphi \rangle_v = \mathbb{T}$

Satisfiability, Validity of Propositional Formulas

A propositional formula φ is ...

- ... **satisfiable** iff **some** valuation satisfies φ
- ... **valid** iff **all** valuations satisfy φ
- ... **failable** iff **some** valuation does **not** satisfy φ
- ... **unsatisfiable** iff **no** valuation satisfies φ

Theorem: A formula φ is valid iff $\neg\varphi$ is unsatisfiable.

Logical Consequence, Logical Equivalence

A propositional formula φ is a **logical consequence** ($\Gamma \models \varphi$) of a set of formulas Γ iff all valuations v that satisfy all $\gamma \in \Gamma$ also satisfy φ.

Two propositional formulas φ, ψ are **logically equivalent** ($\varphi \equiv \psi$) iff they have the same truth value in every valuation. This means, for every valuation $v, \langle\varphi\rangle_v = \langle\psi\rangle_v$ holds.

$(\varphi \equiv \psi)$ iff $\varphi \models \psi$ and $\psi \models \varphi$.

Syntax of Propositional Formulas in RISCAL

```
// the number of atoms
val N: ℕ; // e.g. 3;

// the recursion height
val H: ℕ;

// the raw types and the variously constrained subtypes
type Variable = ℤ[1,N];

rectype(H) Formula =
  T | F |
  VAR(Variable) | NOT(Formula) |
  AND(Formula,Formula) | OR(Formula,Formula) |
  IMPLIES(Formula,Formula) | IFF(Formula,Formula);
```

Semantics of Propositional Formulas in RISCAL

```
type LiteralBase = ℤ[-N,N];
type Literal = LiteralBase with value ≠ 0;
type Valuation = Set[Literal]
  with |value|=N ∧ (∀l∈value. ¬(-l∈value));

pred satisfies(V:Valuation, f:Formula)
decreases height(f);
⇔ match f with
  {
    T -> true;
    F -> false;
    VAR(v:Variable) -> v ∈ V;
    NOT(f1:Formula) -> ¬satisfies(V,f1);
    AND(f1:Formula, f2:Formula) ->
      satisfies(V,f1) ∧ satisfies(V,f2);
    OR(f1:Formula, f2:Formula) ->
      satisfies(V,f1) ∨ satisfies(V,f2);
    IMPLIES(f1:Formula, f2:Formula) ->
      satisfies(V,f1) ⇒ satisfies(V,f2);
    IFF(f1:Formula, f2:Formula) ->
      satisfies(V,f1) ⇔ satisfies(V,f2);
   };

pred satisfiable(f:Formula)
⇔ (∃V:Valuation. satisfies(V,f));
```

Goal: Substitution

Goal: A RISCAL function substituting every occurrence of an atom in a formula with another formula.

Example:
Original formula: $(A \wedge B) \vee (A \wedge C)$
Substituting A with $(\neg D \Rightarrow C)$ leads to new formula

$$((\neg D \Rightarrow C) \wedge B) \vee ((\neg D \Rightarrow C) \wedge C)$$

Theorem: A tautology stays a tautology after substitution

- Specification of this theorem in RISCAL

Goal: Normal Forms

The goal is a RISCAL specification containing:

- Non-recursive data types for CNF, DNF.
- A Predicate *satisfies* for the non-recursive data types.
- Predicates for derived notions *valid*, *satisfiable*, *logically equivalent*, ...
- Functions computing CNF, DNF from recursive representation.
    - Verification of the logical equivalence of the resulting and the original formula.

Negation Normal Form (NNF) - Definition

A propositional formula is in **Negation Normal Form (NNF)** iff it **does not** contain the connectives $\Leftrightarrow$, $\Rightarrow$ and negations are only applied on atomic values.

Definition: A **literal** is either an atom or the negation of an atom.

A formula in NNF can be expressed by truth values, literals, connectives $\vee$, $\wedge$ and parenthesis.

Negation Normal Form (NNF) - Computation

Apply transformations:

- Eliminate $\Leftrightarrow$ and $\Rightarrow$

$$p \Leftrightarrow q \equiv (p \Rightarrow q) \wedge (q \Rightarrow p)$$

$$p \Rightarrow q \equiv \neg p \vee q$$

- Push negations inside **(De Morgan's laws)**

$$\neg(p \wedge q) \equiv \neg p \vee \neg q$$

$$\neg(p \vee q) \equiv \neg p \wedge \neg q$$

- Negation of negation

$$\neg\neg p \equiv p$$

Conjunctive Normal Form (CNF)

A propositional formula is in **conjunctive normal form** (CNF) iff it is a conjunction of disjunctions of literals.

This means, the formula is in the form

$$C_1 \wedge C_2 \wedge \cdots \wedge C_n$$

and for $i = 1..n$, C_i is a disjunction of literals, which means

$$a_{i,1} \vee a_{i,2} \vee \cdots \vee a_{i,m}$$

with literals $a_{i,k}, k = 1..m$

Disjunctive Normal Form (DNF)

A propositional formula is in **disjunctive normal form** (DNF) iff it is a disjunction of conjunctions of literals.

This means, the formula is in the form

$$D_1 \vee D_2 \vee \cdots \vee D_n$$

and for $i = 1..n, D_i$ is a conjunction of literals, which means

$$a_{i,1} \wedge a_{i,2} \wedge \cdots \wedge a_{i,m}$$

with literals $a_{i,k}, k = 1..m$

Computation of DNF / CNF

CNF/DNF can be computed by systematic application of transformations.

$$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$$
$$p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$$

Goal: SAT Solving

The goal is a RISCAL specification containing:

- Recursive function implementing the DPLL algorithm.
- Iterative procedure implementing DPLL.
- Verification of correctness of both (pre-/postconditions, termination measures, invariants)

There already exist basic implementations of DPLL in both recursive and iterative way. (RISCAL Samples)

Goal: Extend these to "real" algorithm with some optimizations.

Boolean satisfiability problem (SAT)

- Problem: Is a propositional formula satisfiable?
- Common problem in artificial intelligence, automated theorem proving, ...
- For n variables there exist 2^n different valuations.

Is there a better approach than Brute Force?

DPLL

- Deciding satisfiability for formulas in CNF
- 1960: first algorithm by Davis and Putnam
- 1962: enhanced algorithm by Davis, Logemann and Loveland
- Foundation for modern SAT-solvers.
- **Idea**: apply rules to eliminate literals step-by-step
  - If we obtain an **empty set of clauses**, the formula is **satisfiable**.
  - If we obtain **some empty clause**, the formula is **unsatisfiable**.

- **Input:** φ .. a propositional formula in **CNF**
- **Output:** $\mathbb{T}$ if φ is satisfiable, $\mathbb{F}$ otherwise

One Literal Rule

Given a formula in CNF

$$C_1 \wedge C_2 \wedge \cdots \wedge C_n$$

If there is a C_i that contains only a **single literal** a we will

- **eliminate all clauses containing** a
- **remove** $\neg a$ **from every clause**

without affecting the satisfiability of the formula.

Pure Literal Rule

Given a formula in CNF

$$C_1 \wedge C_2 \wedge \cdots \wedge C_n$$

If there is a **literal** a that does occur in some C_i but $\neg a$ does not occur in any C_j we will

- **eliminate all clauses containing** a

without affecting the satisfiability of the formula.

Splitting Rule

Given a formula in CNF

$$C_1 \wedge C_2 \wedge \cdots \wedge C_n$$

If there is a literal a that does occur in some C_i and also $\neg a$ does occur in some C_j we will

- **split the problem in two subproblems**

$$C_1 \wedge \cdots \wedge C_n \wedge a$$

$$C_1 \wedge \cdots \wedge C_n \wedge \neg a$$

the original formula is satisfiable iff one of the two resulting formulas is satisfiable

Goal: Syntax and Semantics of First-Order Logic

The goal is a RISCAL specification containing:

- Data types *Term*, *Formula* (both recursive), *Interpretation* and *Valuation*
- Functions computing the meaning of terms and formulas in particular interpretation and valuation.
- A predicate *satisfies* that denotes whether a given interpretation satisfies a formula.
- Predicates for derived notions *valid*, *satisfiable*, *logically equivalent*, *equisatisfiable* ...
- Theorems stating the connection between those predicates
- A function computing the free variables of a formula.

First-Order Logic

Propositional Logic is not always enough:

How to express the following in a propositional formula?
For every y there exists an x such that x is greater than y.

We will introduce:

- A **domain of terms** for variables
- **Functions** to map terms to other terms
- **Predicates** to assign truth values to terms
- **Quantifiers**

Syntax of First-Order Logic

- **Terms** t
  - **variables** v
  - **constants** c
  - **functions** $f(t_1, \ldots, t_n)$
    map n terms to another term
- **Formulas** φ
  - **truth constants** $\mathbb{T}, \mathbb{F}$
  - **predicates** $p(t_1, \ldots, t_n)$
    map n terms to a truth value
  - **connectives** $\neg\varphi, \varphi_1 \wedge \varphi_2, \varphi_1 \vee \varphi_2, \varphi_1 \Rightarrow \varphi_2, \varphi_1 \Leftrightarrow \varphi_2$
  - **quantified formulas** $\exists v.\varphi, \forall v.\varphi$
  - **parentheses** (φ)

Free Variables

- a variable that occurs after a quantifier is called **bound** ($\exists v.\varphi$ or $\forall v.\varphi$)
- a variable is **free** if it is not bound

$$
\begin{aligned}
\mathit{freevars}(v) &= \{v\} \\
\mathit{freevars}(f(t_1, \ldots, t_n)) &= \mathit{freevars}(t_1) \cup \cdots \cup \mathit{freevars}(t_n)
\end{aligned}
$$

$$
\begin{aligned}
\mathit{freevars}(\mathbb{T}) &= \emptyset \\
\mathit{freevars}(\mathbb{F}) &= \emptyset \\
\mathit{freevars}(p(t_1, \ldots, t_n)) &= \mathit{freevars}(t_1) \cup \cdots \cup \mathit{freevars}(t_n) \\
\mathit{freevars}(\neg\varphi) &= \mathit{freevars}(\varphi) \\
\mathit{freevars}(\varphi_1 * \varphi_2) &= \mathit{freevars}(\varphi_1) \cup \mathit{freevars}(\varphi_2) \\
\mathit{freevars}(\exists v.\varphi) &= \mathit{freevars}(\varphi) \backslash \{v\} \\
\mathit{freevars}(\forall v.\varphi) &= \mathit{freevars}(\varphi) \backslash \{v\}
\end{aligned}
$$

Semantics of First-Order Logic I

To define semantics of first-order formulas we introduce:

- A **domain** of terms D
- a **valuation** $v : \mathcal{V} \rightarrow D$
  - maps to every variable a term in D
- an **interpretation** I consisting of
  - A mapping c_I for every constant c to an element in D.
  - A mapping f_I for each function f, $f_I : D^n \rightarrow D$.
  - A mapping f_I for each predicate p, $p_I : D^n \rightarrow \{\mathbb{T}, \mathbb{F}\}$

Semantics of First-Order Logic II

Meaning of terms:
$\langle t \rangle_{I,v}$ maps to every term t its meaning (in D) for a particular interpretation I and valuation v

$$\begin{aligned} \langle x \rangle_{I,v} &= v(x) \\ \langle c \rangle_{I,v} &= c_I \\ \langle f(t_1, \ldots, t_n) \rangle_{I,v} &= f_I(\langle t_1 \rangle_{I,v}, \ldots, \langle t_n \rangle_{I,v}) \end{aligned}$$

Semantics of First-Order Logic III

Meaning of formulas:

$\langle \varphi \rangle_{I,v}$ maps to every first-order formula φ its meaning for a particular interpretation I and valuation v

$$
\begin{aligned}
\langle \mathbb{T} \rangle_{I,v} &= \mathbb{T} \\
\langle \mathbb{F} \rangle_{I,v} &= \mathbb{F} \\
\langle p(t_1, \ldots, t_n) \rangle_{I,v} &= p_I(\langle t_1 \rangle_{I,v}, \ldots, \langle t_n \rangle_{I,v}) \\
\langle \neg \varphi \rangle_{I,v} &= \mathcal{B}_\neg(\langle \varphi \rangle_{I,v}) \\
\langle \varphi_1 * \varphi_2 \rangle_{I,v} &= \mathcal{B}_*(\langle \varphi_1 \rangle_{I,v}, \langle \varphi_2 \rangle_{I,v}), \qquad \text{for } * \in \{\wedge, \vee, \Rightarrow \Leftrightarrow\} \\
\langle \exists x. \varphi \rangle_{I,v} &= \begin{cases} \mathbb{T}, & \text{if } \langle \varphi \rangle_{I,v[x \mapsto d]} = \mathbb{T} \text{ for some } d \in D \\ \mathbb{F}, & \text{otherwise} \end{cases} \\
\langle \forall x. \varphi \rangle_{I,v} &= \begin{cases} \mathbb{T}, & \text{if } \langle \varphi \rangle_{I,v[x \mapsto d]} = \mathbb{T} \text{ for all } d \in D \\ \mathbb{F}, & \text{otherwise} \end{cases}
\end{aligned}
$$

Validity, Satisfiability of First-Order Formulas

A first-order formula is ...

- **valid** iff it holds for all interpretations and valuations.
- **satisfied** by an interpretation iff it holds for all valuations under this interpretation.
- **satisfiable** iff there exists some interpretation that satisfies the formula.
- **unsatisfiable** iff it is not satisfied by any interpretation.

A first-order formula φ is valid iff $\neg\varphi$ is unsatisfiable.

First-Order Logic - Terminology

Two first-order formulas φ, ψ are **logically equivalent** iff for all interpretations I and valuations v

$$\langle\varphi\rangle_{I,v} = \langle\psi\rangle_{I,v}$$

holds.

Two first-order formulas φ, ψ are **equisatisfiable** iff φ is satisfiable when ψ is satisfiable and vice versa.

Logically equivalent formulas are also **equisatisfiable**.

But there are **equisatisfiable** formulas which are **not logically equivalent**!

Goal: Prenex Normal Form and Skolemnization

The goal is a RISCAL specification containing:

- A recursive data type for formulas in Prenex Normal Form.
- Predicates describing the syntax of this new data type. (*satisfies*, *satisfiable*, *logically equivalent*, *equi-satisfiable*)
- A Function transforming a formula to Prenex Normal Form.
  - Verification of the logical equivalence
- Predicates denoting whether a formula is in Prenex Normal Form / Skolem Normal Form
- A function implementing Skolemization.
  - *Verification of the equi-satisfiability.*

Prenex Normal Form (PNF)

A first-order formula is in **Prenex Normal Form (PNF)** iff there is no quantifier appearing as a subformula of a connective.

Example:

- $\forall x.\exists y.(p(x,y) \wedge q(y))$
  - prenex normal form.
- $\exists x.p(x) \vee \forall y.q(y)$
  - **not** in prenex normal form.

For every first-order formula there is a logically equivalent formula in PNF.

Computation of Prenex Normal Form

- eliminate $\Leftrightarrow, \Rightarrow$
- push negations inside
  - De Morgan's laws
  - Negation on quantifiers

$$\neg\forall x.q \equiv \exists x.\neg q \qquad \neg\exists x.q \equiv \forall x.\neg q$$

- pull out quantifiers:
  - ensure bounded variables have unique names (no free or other bound variables with the same name)
  - apply transformations

$$(\exists x.q) \wedge p \equiv \exists x.(q \wedge p) \qquad (\exists x.q) \vee p \equiv \exists x.(q \vee p)$$

$$(\forall x.q) \wedge p \equiv \forall x.(q \wedge p) \qquad (\forall x.q) \vee p \equiv \forall x.(q \vee p)$$

Skolem Normal Form

A first-order formula is in **Skolem Normal Form** iff it contains no existential quantifiers and also is in Prenex Normal Form.

For every first-order formula there is a formula in Skolem Normal Form that is **equisatisfiable** to the original one.

Skolemization

Input: first-order formula
Output: formula in Skolem Normal Form equisatisfiable to the input
The following two statements are equivalent:

1. for all $x \in D$ there exists $y \in D$ such that $P(x, y)$ holds.
2. there exists a function $f : D \to D$ such that for all $x \in D, P(x, f(x))$ holds.

Idea: Introduce new functions to eliminate existential quantifiers.

Example:

$$\forall u \exists v \forall w \exists x . P(u, v, w, x)$$

$$\rightsquigarrow$$

$$\forall u \forall w . P(u, f(u), w, g(u, w))$$

Summary: Goals of the thesis

- Goal: RISCAL specifications for Computational Logic
  - (recursive) data types, predicates, theorems
  - functions, procedures
  - pre- and postconditions, invariants and termination measures
- Propositional Logic
  - Syntax and Semantics
  - Substitution
  - Normal Forms
  - DPLL with optimizations
  - Application: Digital Circuits
- First-Order Logic
  - Syntax and Semantics
  - Syntactic Operations
  - Prenex Normal Form
  - Skolemization