Integrating Computational Properties at the Term Level

Volker Sorge

University of Birmingham, UK

joint work with

Martin Pollet, University of Saarbrücken, Germany

Motivation

- As soon as you formalise a concrete mathematical object it is often hard to see what it was

Motivation

- As soon as you formalise a concrete mathematical object it is often hard to see what it was

concrete object	formal construction
5	$s(s(s(s(s(zero)))))$

Motivation

- As soon as you formalise a concrete mathematical object it is often hard to see what it was

concrete object		formal construction
5	$\longleftrightarrow$	$s(s(s(s(s(zero)))))$

Motivation

- As soon as you formalise a concrete mathematical object it is often hard to see what it was

concrete object		formal construction
5	$\longleftrightarrow$	$s(s(s(s(s(zero)))))$

- Treat some objects as the constants they actually are
- Abstract from the construction of objects

Examples

	Object	Construction
Numbers	5	$s(s(s(s(s(zero)))))$
Lists	$(a\ b\ c)$	$cons(a, cons(b, cons(c, nil)))$
Sets	$\{a, b, c\}$	$\lambda x.(x{=}a \vee x{=}b \vee x{=}c)$
Tuples	$(a, 1, \alpha)$	$pair(a, pair(1, \alpha))$

Annotated Constants (Idea)

Pragmatic approach to

- Identify computational objects as constants
- Attach relevant information on the object to the constants

$\Rightarrow$ Ease communication (with CAS)

$\Rightarrow$ Have special display representation

$\Rightarrow$ Abstract from simple properties (via built-in equality, . . .)

Annotated Constants (Defin.)

Triple $(k, t, \mathbf{a})$ with

- constant k of the signature of formal language $\mathcal{L}$
- term $t \in \mathcal{L}$: the formal definition of k
- annotation $\mathbf{a}$: arbitrary data-structure representing k

$\Rightarrow$ k can be identified given $\mathbf{a}$

$\Rightarrow$ t can be generated given $\mathbf{a}$

Annotated Constants for Sets

Annotation: data-structure of sets with terms in $\mathcal{L}$

$\{b, a, c\}$ with $a, b, c \in \mathcal{L}$

Constant: identifi er generated from a duplicate free ordering of the elements

$k_{\{a,b,c\}} \in \mathcal{L}$

Definition: term generated from ordered set

$\lambda x.(x{=}a \vee x{=}b \vee x{=}c) \in \mathcal{L}$

Annotated Constants for Sets

Annotation: data-structure of sets with terms in $\mathcal{L}$

$\{b, a, c\}$ with $a, b, c \in \mathcal{L}$

Constant: identifi er generated from a duplicate free ordering of the elements

$k_{\{a,b,c\}} \in \mathcal{L}$

Definition: term generated from ordered set

$\lambda x.(x{=}a \vee x{=}b \vee x{=}c) \in \mathcal{L}$

$\Longrightarrow \{b, a, c\} = \{a, b, c\}$

Annotated Constants for Cycles

Annotation: duplicate-free list with integers in $\mathcal{L}$

$(3\ 1\ 2)$ with $1, 2, 3 \in \mathcal{L}$

Constant: identifi er generated from normalised cycle

$k_{(1\ 2\ 3)} \in \mathcal{L}$

Definition: term generated from normalised cycle

$cons(1, cons(2, cons(3, nil))) \in \mathcal{L}$

Annotated Constants for Cycles

Annotation: duplicate-free list with integers in $\mathcal{L}$

$(3\ 1\ 2)$ with $1, 2, 3 \in \mathcal{L}$

Constant: identifi er generated from normalised cycle

$k_{(1\ 2\ 3)} \in \mathcal{L}$

Definition: term generated from normalised cycle

$cons(1, cons(2, cons(3, nil))) \in \mathcal{L}$

$\Longrightarrow (3\ 1\ 2) = (1\ 2\ 3)$

$\Longrightarrow$ Ensure duplicate-freeness when creating annotation

Implementation in Omega

- Extension of the data-structures for terms
- Annotated constant treated as logical constant
- Definition expansion dynamically created from annotation
- Additional reader/pretty-printing function for each kind of annotation
- Check of additional properties during parsing

Manipulation

- Specialised tactics implement computations
- Operate directly on annotation
- Employ efficient algorithms
  within the prover OR apply external CAS
- Choice of implementing annotations as
  efficient data-structure OR input syntax for CAS

Correctness

- Correctness of tactics checked by expansion to calculus level
- Annotated constants replaced by formal defi nition
- Verifi cation of properties explicit during tactic expansion

Example:

L_1 $\quad\quad$ $cycle((1\ 2\ 3))$ $\quad\quad$ *is-cycle*

Correctness

- Correctness of tactics checked by expansion to calculus level
- Annotated constants replaced by formal defi nition
- Verifi cation of properties explicit during tactic expansion

Example:

L_1	$cycle((1\ 2\ 3))$	*defn-expand* $(1\ 2\ 3)$ L_2
L_2	$cycle(cons(1, cons(2, cons(3, nil))))$	*defn-expand* $cycle$ L_3
L_3	$1 \notin \{2, 3\} \wedge cycle(cons(2, cons(3, nil)))$	$\wedge$-*I* L_4, L_5

$$\vdots$$

Case Study

Certifying solutions to permutation group problems
[with A. Cohen, S. Murray, CADE-19]

- Permutations are sets of disjoint cycles
  Example: $\{(3,9)(4,5)(6,10)(7,11)\}$
- Annotation similar to input syntax of GAP
- Specialised tactics employ GAP

Assessment

Nice things

- Recognisable computational objects
- Easier to handle by prover and external CAS
- Eases input and display of objects
- Conservative extension

Assessment

Nice things

- Recognisable computational objects
- Easier to handle by prover and external CAS
- Eases input and display of objects
- Conservative extension

Not so nice things

- New objects require implementation of new constants type plus tactics and equality methods
- Cannot handle free variables in objects
- Conservative extension

Future Work

- Handle various representations in parallel
- Support switch of representations
- Deal with free variables
- Generalise concept to more complex objects