Sum Comprehensions in KeY

Christian Engel

Universität Karlsruhe (TH)

8th International KeY Symposium

Motivation

Goal

Support for sums over an index variable: $\sum_{i=0}^{n} t_i$

Numerical Quantifiers in JML

- Described by an index variable (`i`) a range predicate (`b`) and a numerical expression (`e`): `(\sum int i; b; e)`
- Semantics: $\sum_{i:[b]} [e]$
- Problems
  - potentially infinite
  - arbitrary range predicate complicates automated reasoning

Only sums over an integer interval specified by a lower and upper bound are considered

Motivation

Goal

Support for sums over an index variable: $\sum_{i=0}^{n} t_i$

Numerical Quantifiers in JML

- Described by an index variable (`i`) a range predicate (`b`) and a numerical expression (`e`): `(\sum int i; b; e)`
- Semantics: $\sum_{i:\lceil b \rceil} \lceil e \rceil$
- Problems
  - potentially infinite
  - arbitrary range predicate complicates automated reasoning

Only sums over an integer interval specified by a lower and upper bound are considered

Motivation

Goal

Support for sums over an index variable: $\sum_{i=0}^{n} t_i$

Numerical Quantifiers in JML

- Described by an index variable (`i`) a range predicate (`b`) and a numerical expression (`e`): `(\sum int i; b; e)`
- Semantics: $\sum_{i:\lceil b \rceil} \lceil e \rceil$
- Problems
  - potentially infinite
  - arbitrary range predicate complicates automated reasoning

Only sums over an integer interval specified by a lower and upper bound are considered

Motivation

Goal

Support for sums over an index variable: $\sum_{i=0}^{n} t_i$

Numerical Quantifiers in JML

- Described by an index variable (`i`) a range predicate (`b`) and a numerical expression (`e`): `(\sum int i; b; e)`
- Semantics: $\sum_{i:\lceil b \rceil} \lceil e \rceil$
- Problems
  - potentially infinite
  - arbitrary range predicate complicates automated reasoning

Only sums over an integer interval specified by a lower and upper bound are considered

Motivation

Goal

Support for sums over an index variable: $\sum_{i=0}^{n} t_i$

Numerical Quantifiers in JML

- Described by an index variable (`i`) a range predicate (`b`) and a numerical expression (`e`): `(\sum int i; b; e)`
- Semantics: $\sum_{i:\lceil b \rceil} \lceil e \rceil$
- Problems
  - potentially infinite
  - arbitrary range predicate complicates automated reasoning

Only sums over an integer interval specified by a lower and upper bound are considered

Motivation

Goal

Support for sums over an index variable: $\sum_{i=0}^{n} t_i$

Numerical Quantifiers in JML

- Described by an index variable (`i`) a range predicate (`b`) and a numerical expression (`e`): `(\sum int i; b; e)`
- Semantics: $\sum_{i:\lceil b \rceil} \lceil e \rceil$
- Problems
  - potentially infinite
  - arbitrary range predicate complicates automated reasoning

Only sums over an integer interval specified by a lower and upper bound are considered

Motivation

Goal

Support for sums over an index variable: $\sum_{i=0}^{n} t_i$

Numerical Quantifiers in JML

- Described by an index variable (`i`) a range predicate (`b`) and a numerical expression (`e`): `(\sum int i; b; e)`
- Semantics: $\sum_{i:\lceil b \rceil} \lceil e \rceil$
- Problems
  - potentially infinite
  - arbitrary range predicate complicates automated reasoning

Only sums over an integer interval specified by a lower and upper bound are considered

Motivation

Goal

Support for sums over an index variable: $\sum_{i=0}^{n} t_i$

Numerical Quantifiers in JML

- Described by an index variable (`i`) a range predicate (`b`) and a numerical expression (`e`): `(\sum int i; b; e)`
- Semantics: $\sum_{i:\lceil b \rceil} \lceil e \rceil$
- Problems
  - potentially infinite
  - arbitrary range predicate complicates automated reasoning

Only sums over an integer interval specified by a lower and upper bound are considered

Sums in KeY

Syntax

```
\bSum int x;( l; u; t)
```

where

- `x` is a logic variable bound in `t`
- `l`, `u` and `t` are integer terms

Semantics

$$val_{s,\beta}(\backslash bSum\ x; (l; u; t)) := \begin{cases} \sum_{i=val_{s,\beta}(l)}^{val_{s,\beta}(u-1)} val_{s,\beta_x^i}(t) & \text{if } val_{s,\beta}(l) < val_{s,\beta}(u) \\ 0 & otherwise \end{cases}$$

Sums in KeY

Syntax

`\bSum int x;( l; u; t)`

where

- `x` is a logic variable bound in `t`
- `l`, `u` and `t` are integer terms

Semantics

$$val_{s,\beta}(\backslash bSum\ x; (l; u; t)) := \begin{cases} \sum_{i=val_{s,\beta}(l)}^{val_{s,\beta}(u-1)} val_{s,\beta_x^i}(t) & \text{if } val_{s,\beta}(l) < val_{s,\beta}(u) \\ 0 & otherwise \end{cases}$$

Sums in KeY

Syntax

`\bSum int x;( l; u; t)`

where

- `x` is a logic variable bound in `t`
- `l`, `u` and `t` are integer terms

Semantics

$$val_{s,\beta}(\backslash bSum\ x; (l; u; t)) := \begin{cases} \sum_{i=val_{s,\beta}(l)}^{val_{s,\beta}(u-1)} val_{s,\beta_x^i}(t) & \text{if } val_{s,\beta}(l) < val_{s,\beta}(u) \\ 0 & otherwise \end{cases}$$

Taclets

Axioms

- Base Case: $b < a \rightarrow \sum_{i=a}^{b} s_i = 0$
- Step Case : $b \geq a \rightarrow (\sum_{i=a}^{b-1} s_i) + s_b = \sum_{i=a}^{b} s_i$

Derived Rules

- Purpose: Increasing automation and convenience (interactive proving)
- All derived rules are entailed by the two axioms (proven with KeY)

Taclets

Axioms

- Base Case: $b < a \rightarrow \sum_{i=a}^{b} s_i = 0$
- Step Case : $b \geq a \rightarrow (\sum_{i=a}^{b-1} s_i) + s_b = \sum_{i=a}^{b} s_i$

Derived Rules

- Purpose: Increasing automation and convenience (interactive proving)
- All derived rules are entailed by the two axioms (proven with KeY)

Taclets

Axioms

- Base Case: $b < a \rightarrow \sum_{i=a}^{b} s_i = 0$
- Step Case : $b \geq a \rightarrow (\sum_{i=a}^{b-1} s_i) + s_b = \sum_{i=a}^{b} s_i$

Derived Rules

- Purpose: Increasing automation and convenience (interactive proving)
- All derived rules are entailed by the two axioms (proven with KeY)

Taclets

Axioms

- Base Case: $b < a \rightarrow \sum_{i=a}^{b} s_i = 0$
- Step Case : $b \geq a \rightarrow (\sum_{i=a}^{b-1} s_i) + s_b = \sum_{i=a}^{b} s_i$

Derived Rules

- Purpose: Increasing automation and convenience (interactive proving)
- All derived rules are entailed by the two axioms (proven with KeY)

Taclets

Axioms

- Base Case: $b < a \rightarrow \sum_{i=a}^{b} s_i = 0$
- Step Case : $b \geq a \rightarrow (\sum_{i=a}^{b-1} s_i) + s_b = \sum_{i=a}^{b} s_i$

Derived Rules

- Purpose: Increasing automation and convenience (interactive proving)
- All derived rules are entailed by the two axioms (proven with KeY)

Automation

Idea

Providing rules for patterns occuring when employing induction or loop invariants

Induction Base Case

$\backslash bSum\ x; (l; l; t) \rightsquigarrow 0$

Induction Step Case

$$\backslash bSum\ x; (l; u+1; t) \rightsquigarrow$$
$$\backslash bSum\ x; (l; u; t) + \backslash if\,(l \leq u)\,\backslash then(t^{[x/u]})\,\backslash else(0)$$

Automation

Idea

Providing rules for patterns occuring when employing induction or loop invariants

Induction Base Case

$\backslash bSum\ x; (l; l; t) \rightsquigarrow 0$

Induction Step Case

$$\backslash bSum\ x; (l; u+1; t) \rightsquigarrow$$
$$\backslash bSum\ x; (l; u; t) + \backslash if\,(l \leq u)\,\backslash then(t^{[x/u]})\,\backslash else(0)$$

Automation

Idea

Providing rules for patterns occuring when employing induction or loop invariants

Induction Base Case

$\backslash bSum\, x; (l; l; t) \rightsquigarrow 0$

Induction Step Case

$$\backslash bSum\, x; (l; u+1; t) \rightsquigarrow$$
$$\backslash bSum\, x; (l; u; t) + \backslash if\, (l \leq u)\, \backslash then(t^{[x/u]})\, \backslash else(0)$$

Automation

Idea

Providing rules for patterns occuring when employing induction or loop invariants

Induction Base Case

$\backslash bSum\, x; (l; l; t) \rightsquigarrow 0$

Induction Step Case

$$\backslash bSum\, x; (l; u+1; t) \rightsquigarrow$$
$$\backslash bSum\, x; (l; u; t) + \backslash if\, (l \leq u)\, \backslash then(t^{[x/u]})\, \backslash else(0)$$

Automation – Example

Example Program

```
int s = 0;
for(int i=0; i<a.length; i++){   s += a[i];}
```

Loop Invariant

$\backslash bSum\ x; (0; i; a[x]) \doteq s$

Initially Valid ($i \doteq 0 \wedge s \doteq 0$)

$\backslash bSum\ x; (0; 0; a[x]) \doteq 0 \rightsquigarrow 0 \doteq 0 \rightsquigarrow true$

Loop Body preserves Invariant

$\backslash bSum\ x; (0; i; a[x]) \doteq s \rightarrow \langle \texttt{s += a[i]; i++;} \rangle (\backslash bSum\ x; 0; i; a[x]) \doteq s$
$\rightsquigarrow \backslash bSum\ x; (0; i; a[x]) \doteq s \rightarrow \backslash bSum\ x; (0; i+1; a[x]) \doteq s + a[i]$
$\rightsquigarrow \backslash bSum\ x; (0; i; a[x]) \doteq s \rightarrow \backslash bSum\ x; (0; i; a[x]) + a[i] \doteq s + a[i]$
$\rightsquigarrow true$

Automation – Example

Example Program

```
int s = 0;
for(int i=0; i<a.length; i++){   s += a[i];}
```

Loop Invariant

$\backslash bSum\ x; (0; i; a[x]) \doteq s$

Initially Valid ($i \doteq 0 \wedge s \doteq 0$)

$\backslash bSum\ x; (0; 0; a[x]) \doteq 0 \rightsquigarrow 0 \doteq 0 \rightsquigarrow true$

Loop Body preserves Invariant

$\backslash bSum\ x; (0; i; a[x]) \doteq s \rightarrow \langle \texttt{s += a[i]; i++;} \rangle (\backslash bSum\ x; 0; i; a[x]) \doteq s$
$\rightsquigarrow \backslash bSum\ x; (0; i; a[x]) \doteq s \rightarrow \backslash bSum\ x; (0; i+1; a[x]) \doteq s + a[i]$
$\rightsquigarrow \backslash bSum\ x; (0; i; a[x]) \doteq s \rightarrow \backslash bSum\ x; (0; i; a[x]) + a[i] \doteq s + a[i]$
$\rightsquigarrow true$

Automation – Example

Example Program

```
int s = 0;
for(int i=0; i<a.length; i++){   s += a[i];}
```

Loop Invariant

$\backslash bSum\ x; (0; i; a[x]) \doteq s$

Initially Valid ($i \doteq 0 \wedge s \doteq 0$)

$\backslash bSum\ x; (0; 0; a[x]) \doteq 0 \rightsquigarrow 0 \doteq 0 \rightsquigarrow true$

Loop Body preserves Invariant

$\backslash bSum\ x; (0; i; a[x]) \doteq s \rightarrow \langle \texttt{s += a[i]; i++;} \rangle (\backslash bSum\ x; 0; i; a[x]) \doteq s$
$\rightsquigarrow \backslash bSum\ x; (0; i; a[x]) \doteq s \rightarrow \backslash bSum\ x; (0; i+1; a[x]) \doteq s + a[i]$
$\rightsquigarrow \backslash bSum\ x; (0; i; a[x]) \doteq s \rightarrow \backslash bSum\ x; (0; i; a[x]) + a[i] \doteq s + a[i]$
$\rightsquigarrow true$

Automation – Example

Example Program

```
int s = 0;
for(int i=0; i<a.length; i++){   s += a[i];}
```

Loop Invariant

$\backslash bSum\ x; (0; i; a[x]) \doteq s$

Initially Valid ($i \doteq 0 \wedge s \doteq 0$)

$\backslash bSum\ x; (0; 0; a[x]) \doteq 0 \rightsquigarrow 0 \doteq 0 \rightsquigarrow true$

Loop Body preserves Invariant

$\backslash bSum\ x; (0; i; a[x]) \doteq s \rightarrow \langle \texttt{s += a[i]; i++;} \rangle (\backslash bSum\ x; 0; i; a[x]) \doteq s$
$\rightsquigarrow \backslash bSum\ x; (0; i; a[x]) \doteq s \rightarrow \backslash bSum\ x; (0; i+1; a[x]) \doteq s + a[i]$
$\rightsquigarrow \backslash bSum\ x; (0; i; a[x]) \doteq s \rightarrow \backslash bSum\ x; (0; i; a[x]) + a[i] \doteq s + a[i]$
$\rightsquigarrow true$

Automation – Example

Example Program

```
int s = 0;
for(int i=0; i<a.length; i++){   s += a[i];}
```

Loop Invariant

$\backslash bSum\ x; (0; i; a[x]) \doteq s$

Initially Valid ($i \doteq 0 \wedge s \doteq 0$)

$\backslash bSum\ x; (0; 0; a[x]) \doteq 0 \rightsquigarrow 0 \doteq 0 \rightsquigarrow true$

Loop Body preserves Invariant

$\backslash bSum\ x; (0; i; a[x]) \doteq s \rightarrow \langle \texttt{s += a[i]; i++;} \rangle (\backslash bSum\ x; 0; i; a[x]) \doteq s$

$\rightsquigarrow \backslash bSum\ x; (0; i; a[x]) \doteq s \rightarrow \backslash bSum\ x; (0; i+1; a[x]) \doteq s + a[i]$

$\rightsquigarrow \backslash bSum\ x; (0; i; a[x]) \doteq s \rightarrow \backslash bSum\ x; (0; i; a[x]) + a[i] \doteq s + a[i]$

$\rightsquigarrow true$

Automation – Example

Example Program

```
int s = 0;
for(int i=0; i<a.length; i++){   s += a[i];}
```

Loop Invariant

$\backslash bSum\ x; (0; i; a[x]) \doteq s$

Initially Valid ($i \doteq 0 \wedge s \doteq 0$)

$\backslash bSum\ x; (0; 0; a[x]) \doteq 0 \rightsquigarrow 0 \doteq 0 \rightsquigarrow true$

Loop Body preserves Invariant

$\backslash bSum\ x; (0; i; a[x]) \doteq s \rightarrow \langle \texttt{s += a[i]; i++;} \rangle (\backslash bSum\ x; 0; i; a[x]) \doteq s$
$\rightsquigarrow \backslash bSum\ x; (0; i; a[x]) \doteq s \rightarrow \backslash bSum\ x; (0; i+1; a[x]) \doteq s + a[i]$
$\rightsquigarrow \backslash bSum\ x; (0; i; a[x]) \doteq s \rightarrow \backslash bSum\ x; (0; i; a[x]) + a[i] \doteq s + a[i]$
$\rightsquigarrow true$

Automation – Example

Example Program

```
int s = 0;
for(int i=0; i<a.length; i++){   s += a[i];}
```

Loop Invariant

$\backslash bSum\ x; (0; i; a[x]) \doteq s$

Initially Valid ($i \doteq 0 \wedge s \doteq 0$)

$\backslash bSum\ x; (0; 0; a[x]) \doteq 0 \rightsquigarrow 0 \doteq 0 \rightsquigarrow true$

Loop Body preserves Invariant

$\backslash bSum\ x; (0; i; a[x]) \doteq s \rightarrow \langle \texttt{s += a[i]; i++;} \rangle (\backslash bSum\ x; 0; i; a[x]) \doteq s$
$\rightsquigarrow \backslash bSum\ x; (0; i; a[x]) \doteq s \rightarrow \backslash bSum\ x; (0; i+1; a[x]) \doteq s + a[i]$
$\rightsquigarrow \backslash bSum\ x; (0; i; a[x]) \doteq s \rightarrow \backslash bSum\ x; (0; i; a[x]) + a[i] \doteq s + a[i]$
$\rightsquigarrow true$

Automation – Example

Example Program

```
int s = 0;
for(int i=0; i<a.length; i++){   s += a[i];}
```

Loop Invariant

$\backslash bSum\ x; (0; i; a[x]) \doteq s$

Initially Valid ($i \doteq 0 \wedge s \doteq 0$)

$\backslash bSum\ x; (0; 0; a[x]) \doteq 0 \rightsquigarrow 0 \doteq 0 \rightsquigarrow true$

Loop Body preserves Invariant

$\backslash bSum\ x; (0; i; a[x]) \doteq s \rightarrow \langle \texttt{s += a[i]; i++;} \rangle (\backslash bSum\ x; 0; i; a[x]) \doteq s$
$\rightsquigarrow \backslash bSum\ x; (0; i; a[x]) \doteq s \rightarrow \backslash bSum\ x; (0; i+1; a[x]) \doteq s + a[i]$
$\rightsquigarrow \backslash bSum\ x; (0; i; a[x]) \doteq s \rightarrow \backslash bSum\ x; (0; i; a[x]) + a[i] \doteq s + a[i]$
$\rightsquigarrow true$

Demo

Demo