Semantyka i weryfikacja programów

Bartosz Klin

(slajdy Andrzeja Tarleckiego)

Instytut Informatyki
Wydział Matematyki, Informatyki i Mechaniki
Uniwersytet Warszawski

`http://www.mimuw.edu.pl/~klin` pok. 5680

`klin@mimuw.edu.pl`

Strona tego wykładu:

`http://www.mimuw.edu.pl/~klin/sem19-20.html`

Program Semantics & Verification

Bartosz Klin

(slides courtesy of Andrzej Tarlecki)

Institute of Informatics
Faculty of Mathematics, Informatics and Mechanics
University of Warsaw

`http://www.mimuw.edu.pl/~klin` office: 5680

`klin@mimuw.edu.pl`

This course: `http://www.mimuw.edu.pl/~klin/sem19-20.html`

Specification as a development task

Given precondition φ and postcondition ψ

develop a program S such that

$$\boxed{\{\varphi\}\, S\, \{\psi\}}$$

For instance

Find S such that

$$\{n \geq 0\}\ S\ \{rt^2 \leq n \wedge n < (rt+1)^2\}$$

One correct solution:

$$
\begin{array}{l}
\{n \geq 0\} \\
\quad rt := 0;\ sqr := 1; \\
\quad \textbf{while}\ sqr \leq n\ \textbf{do}\ rt := rt + 1;\ sqr := sqr + 2 * rt + 1 \\
\{rt^2 \leq n \wedge n < (rt+1)^2\}
\end{array}
$$

Hoare's logic: trouble #1

Another correct solution:

$$\{n \geq 0\}$$
$$\textbf{while true do skip}$$
$$\{rt^2 \leq n \wedge n < (rt+1)^2\}$$

since $\vdash$

$$\{n \geq 0\}$$
$$\textbf{while } \{\textbf{true}\} \textbf{ true do skip}$$
$$\{rt^2 \leq n \wedge n < (rt+1)^2\}$$

Partial correctness:
termination not guaranteed,
and hence not requested!

Total correctness

Total correctness $=$ partial correctness $+$ successful termination

Total correctness judgements:

$$[\varphi]\ S\ [\psi]$$

Intended meaning:

Whenever the program S starts in a state satisfying the precondition φ

then it terminates successfully in a final state that satisfies the postcondition ψ

Total correctness: semantics

$$\models [\varphi]\, S\, [\psi]$$

iff

$$\{\varphi\} \subseteq [\![S]\!]\ \{\psi\}$$

where for $S \in \mathbf{Stmt}$, $A \subseteq \mathbf{State}$:

$$[\![S]\!]\ A = \{s \in \mathbf{State} \mid \mathcal{S}[\![S]\!]\ s = a, \text{for some } a \in A\}$$

Spelling this out:

The total correctness judgement $[\varphi]\, S\, [\psi]$ holds, written $\models [\varphi]\, S\, [\psi]$, if for all states $s \in \mathbf{State}$

$$\text{if } \mathcal{F}[\![\varphi]\!]\ s = \mathbf{tt} \text{ then } \mathcal{S}[\![S]\!]\ s \in \mathbf{State} \text{ and } \mathcal{F}[\![\psi]\!]\ (\mathcal{S}[\![S]\!]\ s) = \mathbf{tt}$$

Total correctness: proof rules

$$\frac{}{[\varphi[x \mapsto e]]\ x := e\ [\varphi]}$$

$$\frac{}{[\varphi]\ \mathbf{skip}\ [\varphi]}$$

$$\frac{[\varphi]\ S_1\ [\theta] \qquad [\theta]\ S_2\ [\psi]}{[\varphi]\ S_1 ; S_2\ [\psi]}$$

$$\frac{[\varphi \wedge b]\ S_1\ [\psi] \qquad [\varphi \wedge \neg b]\ S_2\ [\psi]}{[\varphi]\ \mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2\ [\psi]}$$

$$\frac{???}{[???]\ \mathbf{while}\ b\ \mathbf{do}\ S\ [???]}$$

$$\frac{\varphi' \Rightarrow \varphi \qquad [\varphi]\ S\ [\psi] \qquad \psi \Rightarrow \psi'}{[\varphi']\ S\ [\psi']}$$

Total-correctness rule for loops

$$\frac{(nat(l) \wedge \varphi(l+1)) \Rightarrow b \qquad [nat(l) \wedge \varphi(l+1)]\; S\; [\varphi(l)] \qquad \varphi(0) \Rightarrow \neg b}{[\exists l. nat(l) \wedge \varphi(l)]\; \textbf{while } b \textbf{ do } S\; [\varphi(0)]}$$

where

- $\varphi(l)$ is a formula with a free variable l that does not occur in $\textbf{while } b \textbf{ do } S$,
- $nat(l)$ stands for $0 \leq l$, and
- $\varphi(l+1)$ and $\varphi(0)$ result by substituting, respectively, $l+1$ and 0 for l in $\varphi(l)$.

Informally: l is a *counter* that indicates the number of iterations of the loop body

Soundness

(of the proof rules for total correctness for the statements of TINY)

$$\text{if} \quad \boxed{\mathcal{TH}(\mathbf{Int}) \vdash [\varphi]\, S\, [\psi]} \quad \text{then} \quad \boxed{\models [\varphi]\, S\, [\psi]}$$

Proof: By induction on the structure of the proof tree: all the cases are as for partial correctness, except for the rule for loops.

loop rule: Consider $s \in \{nat(l) \wedge \varphi(l)\}$. By induction on $s(l)$ (which is a natural number) show that $\mathcal{S}[\![\mathbf{while}\ b\ \mathbf{do}\ S]\!]\ s = s'$ for some $s' \in \{\varphi(0)\}$ (easy!). To complete the proof, notice that if a variable x does not occur in a statement $S' \in \mathbf{Stmt}$ and two states differ at most on x, then whenever S' terminates successfully starting in one of them, then so it does starting in the other, and the result states differ at most on x.

Completeness

(of the proof system for total correctness for the statements of TINY)

It so happens that:

$$\mathcal{TH}(\mathbf{Int}) \vdash [\varphi]\, S\, [\psi] \quad \text{iff} \quad \models [\varphi]\, S\, [\psi]$$

Proof (idea): Only loops cause extra problems: here, for $\varphi(l)$ take the conjunction of the (partial correctness) loop invariant with the formula

"the loop terminates in exactly l iterations"

It so happens that the latter can indeed be expressed here (since finite tuples of integers and their finite sequences can be coded as natural numbers)!

For example

To prove:

$$[n \geq 0 \land rt = 0 \land sqr = 1]$$
$$\textbf{while } sqr \leq n \textbf{ do}$$
$$rt := rt + 1; sqr := sqr + 2 * rt + 1$$
$$[rt^2 \leq n \land n < (rt+1)^2]$$

use the following invariant with the iteration counter l:

$$sqr = (rt+1)^2 \land rt^2 \leq n \land l = \lfloor \sqrt{n} \rfloor - rt$$

Cheating here, of course:

"$l = \lfloor \sqrt{n} \rfloor - rt$" has to be captured by a first-order formula in the language of TINY

Luckily: this can be done!

Here, this is quite easy: $(rt+l)^2 \leq n < (rt+l+1)^2$

Well-founded relations

A relation $\succ \subseteq W \times W$ is *well-founded* if there is no infinite chain

$$a_0 \succ a_1 \succ \ldots \succ a_i \succ a_{i+1} \succ \ldots$$

Typical example:

$$\langle \mathbf{Nat}, > \rangle$$

A few other examples:

- $\mathbf{Nat}^n$ with component-wise (strict) ordering;
- A^* with proper prefix ordering;
- $\mathbf{Nat}^n$ with lexicographic (strict) ordering generated by the usual ordering on $\mathbf{Nat}$;
- any ordinal with the natural (strict) ordering; etc.

Total correctness = partial correctness + successful termination

Proof method

To prove

$$[\varphi]\ \mathbf{while}\ b\ \mathbf{do}\ S\ [\varphi \wedge \neg b]$$

- show "partial correctness": $[\varphi \wedge b]\ S\ [\varphi]$
- show "termination": find a set W with a well-founded relation $\succ \subseteq W \times W$ and a function $w\colon \mathbf{State} \to W$ such that for all states $s \in \{\varphi \wedge b\}$,

$$w(s) \succ w(\mathcal{S}[\![S]\!]\ s)$$

BTW: $w\colon \mathbf{State} \rightharpoonup W$ may be partial as long as it is defined on $\{\varphi\}$.

Example

Prove:

$$
\begin{array}{l}
[x \geq 0 \wedge y \geq 0] \\
\quad \textbf{while } x > 0 \textbf{ do} \\
\qquad\qquad \textbf{if } y > 0 \textbf{ then } y := y - 1 \textbf{ else } (x := x - 1; y := f(x)) \\
[\textbf{true}]
\end{array}
$$

where f yields a natural number for any natural argument.

- If one knows nothing more about f, then the previous proof rule for the total correctness of loops is useless here.
- BUT: termination can be proved easily using the function $w : \textbf{State} \to \textbf{Nat} \times \textbf{Nat}$, where $w(s) = \langle s\,x, s\,y \rangle$: after each iteration of the loop body the value of w decreases w.r.t. the (well-founded) lexicographic order on pairs of natural numbers.

A fully specified program

$[x \geq 0 \wedge y \geq 0]$

$\mathbf{while}\ [x \geq 0 \wedge y \geq 0]\ x > 0\ \mathbf{do}\ \textsf{\textbf{decr}}\ \langle x, y\rangle\ \textsf{\textbf{in}}\ \mathbf{Nat} \times \mathbf{Nat}\ \textsf{\textbf{wrt}}\ \succ$

$\quad \mathbf{if}\ y > 0\ \mathbf{then}\ y := y - 1\ \mathbf{else}\ (x := x - 1; y := f(x))$

$[\mathbf{true}]$

. . . with various notational variants
assuming some external definitions for
the well-founded set and function into it

Hoare's logic: trouble #2

Find S such that

$$\{n \geq 0\}\, S\, \{rt^2 \leq n \wedge n < (rt+1)^2\}$$

Another correct solution:

$$\begin{array}{l} \{n \geq 0\} \\ \quad rt := 0;\, n := 0 \\ \{rt^2 \leq n \wedge n < (rt+1)^2\} \end{array}$$

A number of techniques to avoid this:

- variables that are required not to be used in the program;
- binary postconditions;
- meta-expressions such as `old(_)` in the Java Modeling Language (JML)
- various forms of algorithmic/dynamic logic, with program modalities.

Parsing

Parsing

The process of mapping a string of tokens (lexems) into a syntactic tree

```
foo+17/(bar-4)
```

is parsed to

```
Plus(Id(foo),Div(Num(17),Sub(Id(bar),Num(4))))
```

- At this stage, context-free properties of programs are interpreted.
- Parsing is preceded by lexing, where regular properties are interpreted. So really:

```
foo+17/(bar-4)
```

is lexed to

```
Id(foo), Plus, Num(17), Div, LPar, Id(bar), Sub, Num(4), RPar
```

and this list of lexems is then parsed.

Complexity of parsing

- The Cocke-Younger-Kasami (CYK) algorithm checks membership in a given context-free language in $\mathcal{O}(n^3)$ time
- It is not hard to modify the algorithm so that it produces a parse tree
- Using Valiant's ideas, the exponent can be brought down to ω, the matrix multiplication exponent.

Currently $\omega \approx 2.373$

- This complexity is a bit unsatisfactory.
- To do better, one restricts attention to deterministic context-free languages.

So e.g. the language of palindromes is excluded

- Then membership checking and parsing can be done in linear time.

Main approaches

1. Top-down parsing
   - Start from the top level of the syntax tree (i.e. the starting nonterminal of the grammar)
   - Try to go down the syntax tree by replacing nonterminals with right-hand sides of productions
   - Example algorithm: LL parsing
   - Tools: Coco/R, JavaCC, Parsec
2. Bottom-up parsing
   - Start from the bottom-left end of the syntax tree
   - Try to build the syntax tree from the left, replacing right-hand sides of productions by nonterminals
   - Example algorithm: LR parsing
   - Tools: yacc, bison, Happy

LL(k) parsing

- The word is processed from the left
- The leftmost derivation is produced

The process simulates a pushdown automaton with one state:

- Stack alphabet $\Gamma = N \cup T$, intial stack: the starting nonterminal, acceptance by empty stack
- If top of the stack is $a \in T$:
  - if a on input: pop it from the stack, read the next symbol
  - otherwise, error: a was expected
- If top of the stack is $A \in N$:
  - choose a production $A \to w$
  - replace A with w on the stack

How to choose the production?

First, Follow and Select sets

For $w \in (T \cup N)^*$, $k \in \mathbb{N}$, $A \in N$:

$$
\begin{aligned}
w|_k &= \begin{cases} a_1 \cdots a_k & \text{if} \quad w = a_1 \cdots a_k v \\ w\# & \text{if} \quad |w| < k \end{cases} \\
\mathsf{First}_k(w) &= \{v|_k : w \to^* v,\ v \in T^*\} \\
\mathsf{Follow}_k(A) &= \{v|_k : S \to^* wAv,\ v \in T^*\} \\
\mathsf{Select}_k(A \to w) &= \mathsf{First}_k(w \cdot \mathsf{Follow}_k(A))
\end{aligned}
$$

- A grammar is (strongly) LL(k) if $\mathsf{Select}_k(A \to w)$ and $\mathsf{Select}_k(A \to v)$ are disjoint for every two productions from the same nonterminal A.
- In the parsing process, a production $A \to w$ is chosen if the next k symbols of input are in $\mathsf{Select}_k(A \to w)$.

LL(1)

- For $k = 1$, we omit the subscripts in First_k, Follow_k and Select_k
- The definitions get simpler:

$$\mathsf{Select}(A \to w) = \begin{cases} (\mathsf{First}(w) \setminus \{\#\}) \cup \mathsf{Follow}(A) & \text{if} \quad w \to^* \epsilon \\ \mathsf{First}(w) & \text{if} \quad w \not\to^* \epsilon \end{cases}$$

- The sets $\mathsf{First}(w)$ and $\mathsf{Follow}(A)$ can be computed by a simple fixpoint procedure, approximating from below

This is parser generation, not parsing!

Limitations of LL(1)

Typical reasons for a grammar to not be LL(1):

- Left ambiguity: $A \to wv \mid wu$

  This is easy to fix by left factorization:

$$\begin{array}{rcl} A & \to & wB \\ B & \to & v \mid u \end{array}$$

- Left recursion: $A \to Aw \mid v$

  Here a fix is:

$$\begin{array}{rcl} A & \to & vB \\ B & \to & wB \mid \epsilon \end{array}$$

This messes up the parse tree to some extent...