Using Correctness-by-Construction to Derive Dead-zone Algorithms

Bruce Watson Loek Cleophas Derrick Kourie

FASTAR Research Group
Stellenbosch University & Pretoria University
South Africa

{bruce, loek, derrick}@fastar.org

Prague Stringology Conference, 1 September 2014

The journey is the reward

- *Derive* an iterative version of the dead-zone algorithm
  Give correctness proof
- Motivate for *correctness-by-construction* (CbC)
- Introduce CbC as a way of explaining algorithms
- Show how CbC can be used in *inventing* new one

Often in *Science of Computer Programming*, Elsevier Journal

Contents

fastar
RESEARCH GROUP

What *is* CbC?

1. Start with a *specification*
2. *Refine* the specification

   . . . in tiny steps

   . . . each of which is *correctness-preserving*
3. Stop when it's executable enough

What do we have at the end?

- Algorithm we can run
- *Derivation* showing how we got there
- Interwoven correctness proof
- 'Tiny' derivation steps give choices

  Family of algorithms

fastar RESEARCH GROUP

Problem statement

Single keyword exact pattern matching:

> *Given two strings $x, y \in \Sigma^*$ over an alphabet Σ (x is the pattern, y is the input text) find all occurrences of x as a contiguous substring of y.*

For convenience:

$$Match(x, y, j) \equiv (x = y_{[j, j+|x|)})$$

Now we have our postcondition:

$$\text{MS} = \bigcup_{j \in [0,|y|) : Match(x,y,j)} \{j\}$$

For example, $y = \texttt{abbaba}$ and $x = \texttt{ba}$ gives

$$\text{MS} = \{2, 4\}$$

Intuitive solution

Partition the indices in y — i.e. set $[0, |y|)$

1. MS — a match has already been found
2. Live_Todo — we know nothing

   still *live*.
3. $\neg$(MS $\cup$ Live_Todo) — we *know* no match occurs

1 and 3 together are the *dead-zone*

Intuitive solution (cont.)

Start with Live_Todo $= [0, |y|)$ (all are live) and MS $= \emptyset$
$\ldots$ reduce to Live_Todo $= \emptyset$ (all dead), i.e.

DO loops

What do we need to derive a loop?

Invariant:
- Predicate/assertion
- True before and after the loop
- True at the top and bottom of each iteration

Variant:
- Integer expression
- Often based on the loop control variable
- Decreasing each iteration, bounded below
- Gives us confidence it's not an infinite loop

Bertrand Meyer 2011 (rephrasing Edsger Dijkstra 1970)

"Publish no loop without its invariant"

See also Furia, Meyer, Velder: *Loop invariants: Analysis, Classification and Examples*, Computing Surveys 2014.

fastar RESEARCH GROUP

DO loops

For invariant I and variant expression V we get

$\{\ P\ \}$
$\{\ I\ \}$
do $G \rightarrow$
  $\{\ I \wedge G \wedge$ expression V has a particular value $\}$
  S_0
  $\{\ I \wedge$ expression V has decreased $\}$
od
$\{\ I \wedge \neg G\ \}$
$\{\ Q\ \}$

First algorithm

$\text{Live\_Todo} := [0, |y|);$
$\text{MS} := \emptyset;$
$\{\ \textbf{invariant:}\ (\forall\ j : j \in \textbf{MS} : \textbf{Match}(x, y, j))\ \ \}$
$\{\ \wedge(\forall\ j : j \notin (\textbf{MS} \cup \textbf{Live\_Todo}) : \neg\textbf{Match}(x, y, j))\ \ \}$
$\{\ \textbf{variant:}\ |\textbf{Live\_Todo}|\ \ \}$
S : Some kind of loop
$\{\ \textbf{invariant}\ \wedge |\textbf{Live\_Todo}| = 0\ \ \}$
$\{\ \textbf{post}\ \ \}$

Ranges of positions

Be cheap:
change Live_Todo to be a *pairwise disjoint set* of live ranges $[l, h)$

$\text{Live\_Todo} := \{[0, |y|)\};$
$\text{MS} := \emptyset;$
$\{\ \textbf{invariant:}\ (\forall\ j : j \in \textbf{MS} : \textbf{Match}(x, y, j))\ \ \}$
$\{\ \ \wedge (\forall\ j : j \notin (\textbf{MS} \cup \textbf{Live\_Todo}) : \neg\textbf{Match}(x, y, j))\ \ \}$
$\{\ \textbf{variant:}\ |\textbf{Live\_Todo}|\ \ \}$
do $\text{Live\_Todo} \neq \emptyset \rightarrow$
  Extract some $[l, h)$ from Live_Todo;

  S_1 : do some stuff to check matches in $[l, h)$ and update Live_Todo
od
$\{\ \textbf{invariant}\ \wedge |\textbf{Live\_Todo}| = 0\ \ \}$
$\{\ \textbf{post}\ \ \}$

Ranges of positions (stripped of invariant stuff)

$\text{Live\_Todo} := \{[0, |y|)\};$
$\text{MS} := \emptyset;$
do $\text{Live\_Todo} \neq \emptyset \rightarrow$
    Extract some $[l, h)$ from Live_Todo;

    S_1 : do some stuff to check matches in $[l, h)$ and update Live_Todo
od
$\{$ **post** $\}$

Ranges of positions (details)

Choose middle of a live range $\left\lfloor \frac{l+h}{2} \right\rfloor$
and check there (also exclude end):

$\mathsf{Live\_Todo} := \{[0, |y| - |x|)\};$
$\mathsf{MS} := \emptyset;$
$\mathbf{do}\ \mathsf{Live\_Todo} \neq \emptyset \rightarrow$
$\quad \text{Extract } [l, h) \text{ from } \mathsf{Live\_Todo};$
$\quad m := \left\lfloor \frac{l+h}{2} \right\rfloor;$
$\quad \mathbf{if}\ \mathit{Match}(x, y, m) \rightarrow$
$\qquad \mathsf{MS} := \mathsf{MS} \cup \{m\}$
$\quad \mathbf{fi};$
$\quad \mathsf{Live\_Todo} := \mathsf{Live\_Todo} \cup [l, m) \cup [m+1, h)$
$\mathbf{od}$
$\{\ \mathbf{post}\ \ \}$

What if we insert an empty range into Live_Todo??

Ranges of positions (details)

$$
\begin{array}{l}
\mathsf{Live\_Todo} := \{[0, |y| - |x|)\}; \\
\mathsf{MS} := \emptyset; \\
\textbf{do}\ \ \mathsf{Live\_Todo} \neq \emptyset \rightarrow \\
\quad \text{Extract } [l, h) \text{ from } \mathsf{Live\_Todo}; \\
\quad \textbf{if}\ \ l \geq h \rightarrow \{\ \text{empty range}\ \}\ \textbf{skip} \\
\quad [\!]\ \ l < h \rightarrow \\
\qquad\qquad m := \left\lfloor \frac{l+h}{2} \right\rfloor; \\
\qquad\qquad \textbf{if}\ \ Match(x, y, m) \rightarrow \\
\qquad\qquad\quad \mathsf{MS} := \mathsf{MS} \cup \{m\} \\
\qquad\qquad \textbf{fi}; \\
\qquad\qquad \mathsf{Live\_Todo} := \mathsf{Live\_Todo} \cup [l, m) \cup [m+1, h) \\
\quad \textbf{fi} \\
\textbf{od} \\
\{\ \textbf{post}\ \ \}
\end{array}
$$

Greater shifts

We can of course user *Match* (or other) information to make larger window shifts

$l', h' := m - shl, m + shr;$
$\text{Live\_Todo} := \text{Live\_Todo} \cup [l, l') \cup [h', h);$

Representing the ‘set’ of live-zones

- Live_Todo are pairwise disjoint... can be done in parallel

  Simone & Thierry have presented an algorithm with similar characteristics
- Live_Todo is a set

  Extracting $[l, h)$ gives an arbitrary pair

  Very poor performance with cache misses in y
- Live_Todo can easily be represented using a queue or stack

  Breadth- or depth-wise traversals of the ranges in y

  Queue: worst case size $|y|$, best case $\left\lceil \frac{|y|}{|x|} \right\rceil$

  Stack: worst case size $log_2|y|$

Live_Todo as a stack

```
Live_Todo := ⟨[0, |y| − |x|)⟩;
MS := ∅;
do Live_Todo ≠ ∅ →
    Pop [l, h) from Live_Todo;
    if l ≥ h → { empty range } skip
    [] l < h →
                m := ⌊(l+h)/2⌋;
                if Match(x, y, m) →
                    MS := MS ∪ {m}
                fi;
                l′, h′ := m − shl, m + shr;
                Push [h′, h) onto Live_Todo;
                Push [l, l′) onto Live_Todo
    fi
od
{ post }
```

Optimization: L-R deadness sharing

maintain integer z with invariant (such that)

$$(\forall\ i : 0 \leq i < z : i \text{ is dead})$$

and keep z maximal, giving:

$\vdots$

$z := 0;$

$\vdots$

do Live_Todo $\neq \emptyset \rightarrow$
  Pop $[l, h)$ from Live_Todo;
  $l := l \max z;$
  $z := l;$
  if $l \geq h \rightarrow \{$ empty range $\}$ **skip**
    $\vdots$

Concurrency: decouple match verification from shifting

Live_Todo $:= \langle [0, |y| - |x|) \rangle$;
MS $:= \emptyset$;
do Live_Todo $\neq \emptyset \rightarrow$
    Pop $[l, h)$ from Live_Todo;
    if $l \geq h \rightarrow$ { empty range } **skip**
    [] $l < h \rightarrow$
        $m := \lfloor \frac{l+h}{2} \rfloor$;
        Add m to queue $\mathsf{Attempt}_t$ for some thread t;
        $l', h' := m - shl, m + shr$;
        Push $[h', h)$ to Live_Todo;
        Push $[l, l')$ to Live_Todo
    fi
od
{ **post** }

Conclusions & ongoing work

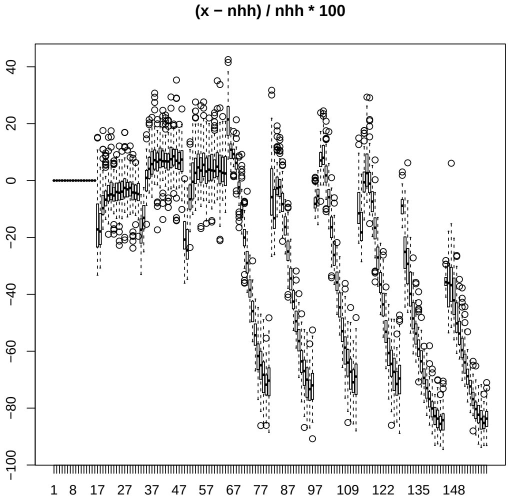
- Interesting new algorithm skeleton
- Performance is similar to *comparable* algorithms

  Not yet clear how to integrate advances in other algorithms
- CbC is robust and relatively easy

  Creativity is not hampered: new algorithms can be invented
- Useful methodology for bringing coherence to a field

  . . . and detecting unexplored parts

Performance



Data Sources: i7 / Wall plug / Sequential / * / * / Bible / Machine time