The Resource Bound Problem for Imperative Languages

Quentin Carbonneaux

April 21, 2015

An Example Program

Suppose we have n consecutive stack operations.

```
push();
popall();
push();
push();
⋮
```

$$\overbrace{x_1, x_2, \ldots, x_s}^{s}.$$

popall costs s, push costs 1.
The stack starts empty.

We can rephrase the setting using non-determinism and a loop.

```
while n > 0
    if *
        push();
    else
        popall();
    n = n - 1;
```

We can rephrase the setting using non-determinism and a loop.

```
while n > 0
    if *
        push();
    else
        popall();
    n = n - 1;
```

What is the worst-case?

Answer: $2n$

The number of elements pushed cannot be bigger than n. Thus,

- the combined cost for all the pops is $\leq n$,
- and the combined cost for all pushes is $\leq n$.

Hence $2n$.

Is it a Proof?

This is nice reasoning, but informal. It's unclear how to scale this to real programs. We need

- Formal Cost Semantics
- Formal Logical Reasoning
- Automation

My work brings some answers to these 3 points.

Potential Method and Quantitative Logic

A Good Start: Tarjan's Idea

Tarjan proposes to find a *potential function*.

- σ, σ' are program states.
- C_l is the cost of one loop iteration.
- $\Phi : \text{State} \to \mathbb{Z}$ is a potential function iff

$$\Phi(\sigma) \geq \begin{cases} 0 & \text{if } \sigma(n) \leq 0 \\ C_l + \Phi(\sigma') & \text{if } \sigma(n) > 0 \end{cases} .$$

Why?

$$\begin{aligned}
\Phi(\sigma_0) &\geq C_l + \Phi(\sigma_1) \\
\Phi(\sigma_1) &\geq C_l + \Phi(\sigma_2) \\
\Phi(\sigma_2) &\geq C_l + \Phi(\sigma_3) \\
&\vdots \\
\Phi(\sigma_f) &\geq 0
\end{aligned}$$

$\sum C_l$ is the total program cost.

Claim: Taking $\Phi = 2n + s$ Works

push case ($C_l = 1$):

$$\begin{aligned}\Phi(n, s) = 2n + s &= 1 + 2(n-1) + s + 1 \\ &\geq C_l + \Phi(n-1, s+1).\end{aligned}$$

popall case ($C_l = s$):

$$\begin{aligned}\Phi(n, s) = 2n + s &= s + 2(n-1) + 1 \\ &\geq C_l + \Phi(n-1, 0).\end{aligned}$$

base case: $\Phi(0, s) = s \geq 0$

What did we prove?

$\Phi(n, 0) = 2n$ bounds the number of stack ops.

What did we prove?

$\Phi(n, 0) = 2n$ bounds the number of stack ops.

We proved more! If the stack is not initially empty, $\Phi(n, s)$ is a correct bound. We had to introduce s in Φ for the *induction*.

Potential Function $\implies$ Compositional

Using triples, we get compositional resource bound proofs:

Write $\{\Phi\}S\{\Phi'\}$ to mean $\Phi(\sigma) \geq C_S + \Phi'(\sigma')$.

Potential Function $\implies$ Compositional

Using triples, we get compositional resource bound proofs:

Write $\{\Phi\}S\{\Phi'\}$ to mean $\Phi(\sigma) \geq C_S + \Phi'(\sigma')$.

If $\{\Phi\}S_1\{\Phi'\}$ and $\{\Phi'\}S_2\{\Phi''\}$ then

$$\{\Phi\}S_1; S_2\{\Phi''\}.$$

This is telescoping, because $C_{S_1;S_2} = C_{S_1} + C_{S_2}$.

$$\frac{\{\Phi\}S_1\{\Phi'\} \qquad \{\Phi'\}S_2\{\Phi''\}}{\{\Phi\}S_1; S_2\{\Phi''\}}\,(\text{Q:}\textsc{Seq})$$

$$\frac{\{X \wedge \Phi\}S\{\Phi\}}{\{\Phi\}\textsf{while } X \textsf{ do } S\{\neg X \wedge \Phi\}}\ (\text{Q:}\textsc{Loop})$$

Combining Logic and Potential

Logic Assertions	Potential Functions
State $\to \mathbb{B}$	State $\to \mathbb{Q}_0^+ \cup \{\infty\}$
$\top$	$0, 1, 1.5, 2, \ldots$
$\bot$	∞
$\wedge$	$+$
$\vee$	min

Example: $\top \vee \bot = \top$ translates to $\min(0, \infty) = 0$.

Complete Logic for Clight

$$\frac{}{\Delta; B; R \vdash_L \{Q\}\, \mathsf{skip}\, \{Q\}}\ (\text{L:Skip}) \qquad \frac{}{\Delta; B; R \vdash_L \{B\}\, \mathsf{break}\, \{Q\}}\ (\text{L:Break})$$

$$\frac{n < 0 \implies Q \geqslant 0}{\Delta; B; R \vdash_L \{Q\}\, \mathsf{tick}(n)\, \{Q - n\}}\ (\text{L:Tick}) \qquad \frac{}{\Delta; B; R \vdash_L \{R\,(\sigma(x))\}\, \mathsf{return}\ x\, \{Q\}}\ (\text{L:Return})$$

$$\frac{}{\Delta; B; R \vdash_L \{\mathsf{istrue}\ [\![e]\!]_\sigma \implies Q\}\, \mathsf{assert}\ e\, \{Q\}}\ (\text{L:Assert}) \qquad \frac{}{\Delta; B; R \vdash_L \{\lambda\sigma.\, Q\,\sigma[x \mapsto [\![e]\!]_\sigma]\}\, x \leftarrow e\, \{Q\}}\ (\text{L:Update})$$

$$\frac{I \geqslant Q \qquad \Delta; Q; R \vdash_L \{I\}\, S\, \{I\}}{\Delta; B; R \vdash_L \{I\}\, \mathsf{loop}\ S\, \{Q\}}\ (\text{L:Loop}) \qquad \frac{\begin{array}{c}\Delta; B; R \vdash_L \{P\}\, S_1\, \{Q'\} \\ \Delta; B; R \vdash_L \{Q'\}\, S_2\, \{Q\}\end{array}}{\Delta; B; R \vdash_L \{P\}\, S_1; S_2\, \{Q\}}\ (\text{L:Seq})$$

$$\frac{\Delta; B; R \vdash_L \{\mathsf{istrue}\ [\![e]\!]_\sigma + P\}\, S_1\, \{Q\} \qquad \Delta; B; R \vdash_L \{\mathsf{isfalse}\ [\![e]\!]_\sigma + P\}\, S_2\, \{Q\}}{\Delta; B; R \vdash_L \{P\}\, \mathsf{if}(e)\ S_1\ \mathsf{else}\ S_2\, \{Q\}}\ (\text{L:If})$$

$$\frac{\Delta(f) = \forall z\, \vec{v}\, v.(P_f\, z\, \vec{v}, Q_f\, z\, v) \qquad P \geqslant P_f\, y\,(\sigma(\vec{x})) + A \qquad \forall v.\,(Q_f\, y\, v + A \geqslant \lambda\sigma.\, Q\,\sigma[r \mapsto v])}{\Delta; B; R \vdash_L \{P\}\, r \leftarrow f(\vec{x})\, \{Q\}}\ (\text{L:Call})$$

$$\frac{\begin{array}{c}\Delta \cup \Delta'; B; R \vdash_L \{P\}\, S\, \{Q\} \qquad P_f \geqslant 0 \\ \forall f\, P_f\, Q_f.\, \Delta'(f) = \forall z\, \vec{v}\, v.(P_f\, z\, \vec{v}, Q_f\, z\, v) \to \forall y\, \vec{v}.\,(\Delta \cup \Delta'; \bot; Q_f\, y \vdash_L \{P_f\, y\, \vec{v}\}\, S_f\, \{\bot\})\end{array}}{\Delta; B; R \vdash_L \{P\}\, S\, \{Q\}}\ (\text{L:Extend})$$

$$\frac{\begin{array}{c}\Delta; B'; R' \vdash_L \{P'\}\, S\, \{Q'\} \\ P \geqslant P' \quad Q' \geqslant Q \quad B' \geqslant B \quad \forall v.\,(R'\, v \geqslant R\, v)\end{array}}{\Delta; B; R \vdash_L \{P\}\, S\, \{Q\}}\ (\text{L:Weaken}) \qquad \frac{\Delta; B; R \vdash_L \{P\}\, S\, \{Q\} \qquad x \in \mathbb{Q}_0^+}{\Delta; B + x; R + x \vdash_L \{P + x\}\, S\, \{Q + x\}}\ (\text{L:Frame})$$

Figure 13: Rules of the Quantitative Hoare Logic

Example Applications

- Linear stack bound for Fib function.
- Linear stack bound for factorial.
- Logarithmic stack bound for binary search.
- Logarithmic stack bound for Quicksort.
- Backend for automated stack bounds tool.
- A stack bound for full CertiKOS.
- A powerful semantic tool.
- A common language for tools and proofs.

Cost Semantics

Back to the Technique: What is C_S?

We need a precise definition of the "resource cost" of a program.

$$
\begin{aligned}
S := x \leftarrow E \mid \mathsf{skip} \mid S; S \\
\mid \mathsf{while}\ E\ \mathsf{do}\ S \\
\mid \mathsf{if}\ E\ \mathsf{then}\ S\ \mathsf{else}\ S
\end{aligned}
$$

Classic Small-step Semantics

Define configurations as: (S, K, H).

$$K := \mathsf{KLoop}\ E\ S\ K \mid \mathsf{KSeq}\ S\ K \mid \mathsf{KEmpty}$$

Define rules like:

$$\frac{}{(S_1; S_2, K, H) \to (S_1, \mathsf{KSeq}\ S_2\ K, H)}\ (\text{S:}\textsc{Seq}1)$$

$$\frac{}{(\mathsf{skip}, \mathsf{KSeq}\ S\ K, H) \to (S, K, H)}\ (\text{S:}\textsc{Seq}2)$$

$$\frac{[\![E]\!]_H = n}{(x \leftarrow E, K, H) \to (\mathsf{skip}, K, H[x \leftarrow n])}\ (\text{S:}\textsc{Set})$$

$$\frac{[\![E]\!]_H = 0}{(\mathsf{while}\ E\ \mathsf{do}\ S, K, H) \to (\mathsf{skip}, K, H)}\ (\text{S:}\textsc{While1})$$

$$\frac{[\![E]\!]_H \neq 0}{(\mathsf{while}\ E\ \mathsf{do}\ S, K, H) \to (S, \mathsf{KLoop}\ E\ S\ K, H)}\ (\text{S:}\textsc{While2})$$

$$\frac{}{(\mathsf{skip}, \mathsf{KLoop}\ E\ S\ K, H) \to (\mathsf{while}\ E\ \mathsf{do}\ S, K, H)}\ (\text{S:}\textsc{While3})$$

Example

$$
\begin{aligned}
&(\mathsf{while}\ x < 1\ \mathsf{do}\ x \leftarrow x+1, \mathsf{KE}, \{x \leftarrow 0\}) \\
&\to (x \leftarrow x+1, \mathsf{KL}\ (x{<}1)\ (x \leftarrow x+1)\ \mathsf{KE}, \{x \leftarrow 0\}) \\
&\to (\mathsf{skip}, \mathsf{KL}\ (x{<}1)\ (x \leftarrow x+1)\ \mathsf{KE}, \{x \leftarrow 1\}) \\
&\to (\mathsf{while}\ x < 1\ \mathsf{do}\ x \leftarrow x+1, \mathsf{KE}, \{x \leftarrow 1\}) \\
&\to (\mathsf{skip}, \mathsf{KE}, \{x \leftarrow 1\})
\end{aligned}
$$

S:WHILE2, S:SET, S:WHILE3, S:WHILE1

Cost Semantics

Program configurations are now: (S, K, H, c). This c is *resource counter*, it changes as the program executes.

$$S := \ldots \mid \mathsf{tick}(n)$$

$$\frac{}{(\mathsf{tick}(n), K, H, c) \to (\mathsf{skip}, K, H, c - n)} \text{(S:Tick)}$$

All the rules get a side condition $c \geq 0$.

Stuck Configurations

A configuration is *stuck* if it cannot execute further. Different kind of stuckness exist:

- Memory error.
- Divisions by 0.
- Resource crashes.

We recognize resource crashes as configurations (S, K, H, c) where $c < 0$.

Resource Safety

Identifying crashes lets us define safety:

A configuration C is *safe* for n steps if any execution sequence of n steps *or less* starting in C does not end as a resource crash.

This predicate is formally defined inductively using the small-step semantics.

Why Indexing the Definition?

- It lets us talk about diverging executions. (Think about stack usage.)
- It makes the soundness proof of our logic possible. (In the `while` case.)

Resource Cost of a Program

$$C_S = \inf\{c \mid \forall n.\ safe_n(S, \mathsf{KEmpty}, H_0, c)\}$$

Resource Cost of a Program

$$C_S = \inf\{c \mid \forall n.\, \mathit{safe}_n(S, \mathsf{KEmpty}, H_0, c)\}$$

Problems with this notion:

- inf is a promess for trouble.
- Explicit mention of KEmpty and H_0.
- In short: non compositional!

Soundness of the Logic

Revisit Semantic Validity

Remember $\{\Phi\}S\{\Phi'\} \equiv \Phi(\sigma) \geq C_S + \Phi'(\sigma')$.
Let's be more thorough.

Revisit Semantic Validity

Remember $\{\Phi\}S\{\Phi'\} \equiv \Phi(\sigma) \geq C_S + \Phi'(\sigma')$.
Let's be more thorough.

A continuation K is safe_n for a potential Φ' if
$\forall H\, c.\ \Phi'(H) \leq c \implies \mathit{safe}_n(\mathsf{skip}, K, H, c)$.
We write $\mathit{safeK}_n(\Phi', K)$.

Revisit Semantic Validity

Remember $\{\Phi\}S\{\Phi'\} \equiv \Phi(\sigma) \geq C_S + \Phi'(\sigma')$.
Let's be more thorough.

A continuation K is safe_n for a potential Φ' if
$\forall H\ c.\ \Phi'(H) \leq c \implies \mathit{safe}_n(\mathsf{skip}, K, H, c)$.
We write $\mathit{safeK}_n(\Phi', K)$.

A triple $\{\Phi\}S\{\Phi'\}$ is *valid* for n steps if
$\forall K\ H\ c\ k{\leq}n.\ \mathit{safeK}_k(\Phi', K) \wedge \Phi(H) \leq c$
$\implies \mathit{safe}_k(S, K, H, c)$.

One Remark

We have $\forall \Phi' \, n. \, safeK_n(\Phi', \mathsf{KEmpty})$.

So, $\{\Phi\} S \{\Phi'\}$ valid for every n implies:

$$\forall n. \, safe_n(S, \mathsf{KEmpty}, H_0, \Phi(H_0)).$$

That is, $C_S \leq \Phi(H_0)$.

Our semantic validity of triples is connected to the intuitive resource cost of a program.

Example Proof: The Sequence

$$\frac{\{\Phi\}S_1\{\Phi'\} \qquad \{\Phi'\}S_2\{\Phi''\}}{\{\Phi\}S_1;S_2\{\Phi''\}} \text{(Q:SEQ)}$$

If the two premisses are valid for n steps, the conclusion will be too.

Let K a continuation safe for $k \leq n$ steps, let $\Phi(H) \leq c$. We want $safe_k(S_1;S_2,K,H,c)$.

Proving $safe_k(S_1; S_2, K, H, c)$

- By definition of $safe_k$, we must show $safe_{k-1}(S_1, \textsf{KSeq}\ S_2\ K, H, c)$.

Proving $safe_k(S_1; S_2, K, H, c)$

- By definition of $safe_k$, we must show $safe_{k-1}(S_1, \mathsf{KSeq}\ S_2\ K, H, c)$.
- By the first premiss, we now have to show $safeK_{k-1}(\Phi', \mathsf{KSeq}\ S_2\ K)$.

Proving $safe_k(S_1; S_2, K, H, c)$

- By definition of $safe_k$, we must show $safe_{k-1}(S_1, \mathsf{KSeq}\ S_2\ K, H, c)$.
- By the first premiss, we now have to show $safeK_{k-1}(\Phi', \mathsf{KSeq}\ S_2\ K)$.
- By definition of $safeK_{k-1}$, we must show $safe_{k-1}(\mathsf{skip}, \mathsf{KSeq}\ S_2\ K, H', c')$ for $\Phi'(H') \leq c'$.

Proving $safe_k(S_1; S_2, K, H, c)$

- By definition of $safe_k$, we must show $safe_{k-1}(S_1, \mathsf{KSeq}\ S_2\ K, H, c)$.
- By the first premiss, we now have to show $safeK_{k-1}(\Phi', \mathsf{KSeq}\ S_2\ K)$.
- By definition of $safeK_{k-1}$, we must show $safe_{k-1}(\mathsf{skip}, \mathsf{KSeq}\ S_2\ K, H', c')$ for $\Phi'(H') \leq c'$.
- By definition of $safe_{k-1}$, we must show $safe_{k-2}(S_2, K, H, c')$, for $\Phi'(H') \leq c'$.

Proving $safe_k(S_1; S_2, K, H, c)$

- By definition of $safe_k$, we must show $safe_{k-1}(S_1, \mathsf{KSeq}\ S_2\ K, H, c)$.
- By the first premiss, we now have to show $safeK_{k-1}(\Phi', \mathsf{KSeq}\ S_2\ K)$.
- By definition of $safeK_{k-1}$, we must show $safe_{k-1}(\mathsf{skip}, \mathsf{KSeq}\ S_2\ K, H', c')$ for $\Phi'(H') \leq c'$.
- By definition of $safe_{k-1}$, we must show $safe_{k-2}(S_2, K, H, c')$, for $\Phi'(H') \leq c'$.
- The second premiss finishes the proof, since $safeK_{k-2}(\Phi'', K)$ (after weakening).

This Seemed Tricky?

It's not! The Coq proof we have for the logic might be the smallest soundness proof for a program logic for C in Coq! (400 lines)

Very much recommended for novices and teaching!

In Coq With No Tactic Fu

```
Proof with try (intros; ksgn).
unfold valid at 3, safe; intros.
assert (CNNEG: 0 <= c).
{ eapply (valid_nneg n B R P Q' s1 x XSGN PRE1)...
  eapply (valid_nneg n B R Q' Q s2 x XSGN PRE2)...
  eassumption.
}
split; [ exact CNNEG | step ].
apply PRE1 with (x := x); try (omega || assumption).
clear INI.
unfold safek, safe; intuition;
  try step; try ksgn.
+ apple SAFEK; assumption.
+ simpl; apple SAFEK; auto.
+ eapply (valid_nneg n B R Q' Q s2 x XSGN PRE2)...
  eassumption.
+ eapply PRE2 with (x := x); try (omega || apply INI).
  simpl; apple SAFEK; now auto.
+ eapply (valid_nneg n B R Q' Q s2 x XSGN PRE2)...
  eassumption.
Qed.
```

The previous proof in the full Clight context in Coq. Arguably very short!

Automatic Derivation of Resource Bounds

Automating the Proof Search

A potential function can be any function.
What if we only look at *a few* of them?

Automating the Proof Search

A potential function can be any function.
What if we only look at *a few* of them?

$$\Phi(H) = k_0 + \sum_{x,y} k_{xy} \cdot |[H(x), H(y)]|$$

where $k_{\_} \in \mathbb{Q}_0^+$ and $|[a, b]| = \max(b - a, 0)$.

Motivation for Intervals

```
for (x = 0; x < y-1; x += 2) {
  tick(1);
}
```

Motivation for Intervals

```
for (x = 0; x < y-1; x += 2) {
  tick(1);
}
```

$$\Phi(H) = \frac{1}{2}|[0, H(y)]|$$

Indices

To succintly refer to a potential function, we use *indices*.

$$\mathcal{I} = \{0\} \cup \{xy \mid x, y \in \text{Vars}\}$$

Indices

To succintly refer to a potential function, we use *indices*.

$$\mathcal{I} = \{0\} \cup \{xy \mid x, y \in \text{Vars}\}$$

If $f_0 = \lambda\_.\, 1$ and $f_{xy} = \lambda H.\, |[H(x), H(y)]|$

$$\Phi(H) = \sum_{I \in \mathcal{I}} k_I \cdot f_I(H).$$

f_I is a *base function*. (Linear algebra.)

Rules for Potential

Idea: Reuse the logic's rules for soundness and constrain potential functions on their coefficients.

- For example: Ensure $\Phi > \Phi'$ by $\forall I$, $k_I > k'_I$.
- Reuse all syntax directed rules as-is.
- Add a little more work for statemtents modifying the heap.

Increments of Variables

Consider the increment program $x \leftarrow x + 1$.

The logic rule is notoriously unhelpful:

$$\frac{}{\{\lambda H.\, \Phi(H[x \leftarrow [\![E]\!]_H])\} x \leftarrow E \{\Phi\}} \text{(Q:S}\textsc{et}\text{)}$$

We need to understand how $\Phi = \sum_I k_I \cdot f_I$ is changed.

Constraints for $\{\Phi\}x \leftarrow x + 1\{\Phi'\}$

Only $[y, x]$ and $[x, y]$ will change. We write x' for the new value of x.

Constraints for $\{\Phi\}x \leftarrow x + 1\{\Phi'\}$

Only $[y, x]$ and $[x, y]$ will change. We write x' for the new value of x.

- Consider $\Phi = k_0 + k_{yx} \cdot |[y, x]|$,
  we have $|[y, x']| = |[y, x]| + 1$, so
  $\Phi' = (k_0 - k_{yx}) + k_{yx} \cdot |[y, x]|$.

Constraints for $\{\Phi\}x \leftarrow x + 1\{\Phi'\}$

Only $[y, x]$ and $[x, y]$ will change. We write x' for the new value of x.

- Consider $\Phi = k_0 + k_{yx} \cdot |[y, x]|$,
  we have $|[y, x']| = |[y, x]| + 1$, so
  $\Phi' = (k_0 - k_{yx}) + k_{yx} \cdot |[y, x]|$.

- Suppose $x' \in [x, y]$ and consider
  $\Phi = k_0 + k_{xy} \cdot |[x, y]|$,
  we have $|[x', y]| = |[x, y]| - 1$, so
  $\Phi' = (k_0 + k_{xy}) + k_{xy} \cdot |[x, y]|$.

Constraints for $\{\Phi\}x \leftarrow x + 1\{\Phi'\}$

Only $[y, x]$ and $[x, y]$ will change. We write x' for the new value of x.

- Consider $\Phi = k_0 + k_{yx} \cdot |[y, x]|$,
  we have $|[y, x']| = |[y, x]| + 1$, so
  $\Phi' = (k_0 - k_{yx}) + k_{yx} \cdot |[y, x]|$.

- Suppose $x' \in [x, y]$ and consider
  $\Phi = k_0 + k_{xy} \cdot |[x, y]|$,
  we have $|[x', y]| = |[x, y]| - 1$, so
  $\Phi' = (k_0 + k_{xy}) + k_{xy} \cdot |[x, y]|$.

In both cases $\Phi(H) = \Phi'(H[x \leftarrow x + 1])$.

Full Automatic System for Clight

$$\frac{}{B; R; (\Gamma, Q) \vdash \mathsf{skip} \dashv (\Gamma, Q)}\ \text{(Q:Skip)} \qquad \frac{}{(\Gamma, Q_B); R; (\Gamma, Q_B) \vdash \mathsf{break} \dashv (\Gamma', Q')}\ \text{(Q:Break)}$$

$$\frac{n < 0 \implies Q \geqslant 0}{B; R; (\Gamma, Q) \vdash \mathsf{tick}(n) \dashv (\Gamma, Q{-}n)}\ \text{(Q:Tick)} \qquad \frac{P = Q_R[ret/x] \quad \Gamma = \Gamma_R[ret/x] \quad \forall i \in \mathsf{dom}(P).\, p_i = q_i}{B; (\Gamma_R, Q_R); (\Gamma, Q) \vdash \mathsf{return}\ x \dashv (\Gamma', Q')}\ \text{(Q:Return)}$$

$$\frac{\forall u.(q_{yu} = q'_{xu} + q'_{yu} \wedge q_{uy} = q'_{ux} + q'_{uy})}{B; R; (\Gamma[x/y], Q{+}M_u{+}M_e(y)) \vdash x \leftarrow y \dashv (\Gamma, Q')}\ \text{(Q:Update)} \qquad \frac{Q \geqslant Q' \quad (\Gamma', Q'); R; (\Gamma, Q) \vdash S \dashv (\Gamma, Q)}{B; R; (\Gamma, Q) \vdash \mathsf{loop}\ S \dashv (\Gamma', Q')}\ \text{(Q:Loop)}$$

$$\frac{\begin{array}{c} q'_{0y} = q_{0y} - \sum_u \max(q_{ux}, -q_{xu}) \\ q'_{y0} = q_{y0} - \sum_u \max(q_{xu}, -q_{ux}) \end{array}}{B; R; (\Gamma[x/x{+}y], Q) \vdash x \leftarrow x + y \dashv (\Gamma, Q')}\ \text{(Q:Inc)} \qquad \frac{\begin{array}{c} q'_{0y} = q_{0y} - \sum_u \max(q_{xu}, -q_{ux}) \\ q'_{y0} = q_{y0} - \sum_u \max(q_{ux}, -q_{xu}) \end{array}}{B; R; (\Gamma[x/x{-}y], Q) \vdash x \leftarrow x - y \dashv (\Gamma, Q')}\ \text{(Q:Dec)}$$

$$\frac{\begin{array}{c} B; R; (\Gamma \wedge e, Q) \vdash S_1 \dashv (\Gamma', Q') \\ B; R; (\Gamma \wedge \neg e, Q) \vdash S_2 \dashv (\Gamma', Q') \end{array}}{B; R; (\Gamma, Q) \vdash \mathsf{if}(e)\ S_1\ \mathsf{else}\ S_2 \dashv (\Gamma', Q')}\ \text{(Q:If)} \qquad \frac{\begin{array}{c} B; R; (\Gamma, Q) \vdash S_1 \dashv (\Gamma', Q') \\ B; R; (\Gamma', Q') \vdash S_2 \dashv (\Gamma'', Q'') \end{array}}{B; R; (\Gamma, Q) \vdash S_1; S_2 \dashv (\Gamma'', Q'')}\ \text{(Q:Seq)}$$

$$\frac{\begin{array}{c} (\Gamma_f, Q_f, \Gamma'_f, Q'_f) \in \Delta(f) \qquad \mathrm{Loc} = \mathrm{Locals}(Q) \\ \forall i \neq j.\, x_i \neq x_j \quad c \in \mathbb{Q}_0^+ \quad Q = P + S \quad Q' = P' + S \quad U = Q_f[\vec{args}/\vec{x}] \quad U' = Q'_f[ret/r] \\ \forall i \in \mathsf{dom}(U).\, p_i = u_i \quad \forall i \in \mathsf{dom}(U').\, p'_i = u'_i \quad \forall i \notin \mathsf{dom}(U').\, p'_i = 0 \quad \forall i \notin \mathrm{Loc}.\, s_i = 0 \end{array}}{B; R; (\Gamma_f[\vec{args}/\vec{x}] \wedge \Gamma_{\mathrm{Loc}}, Q{+}c) \vdash r \leftarrow f(\vec{x}) \dashv (\Gamma'_f[ret/r] \wedge \Gamma_{\mathrm{Loc}}, Q'{+}c)}\ \text{(Q:Call)}$$

$$\frac{}{B; R; (\Gamma, Q) \vdash \mathsf{assert}\ e \dashv (\Gamma \wedge e, Q)}\ \text{(Q:Assert)} \qquad \frac{\begin{array}{c} \Sigma f = (\vec{y}, S_f) \quad Q_f \geqslant 0 \quad Q'_f \geqslant 0 \\ B; (\Gamma'_f, Q'_f); (\Gamma_f[\vec{args}/\vec{y}], Q_f[\vec{args}/\vec{y}]) \vdash S_f \dashv (\Gamma', Q') \end{array}}{(\Gamma_f, Q_f, \Gamma'_f, Q'_f) \in \Delta(f)}\ \text{(Q:Extend)}$$

$$\frac{\Gamma_1 \models \Gamma_2 \quad Q_1 \succeq_{\Gamma_1} Q_2 \quad B; R; (\Gamma_2, Q_2) \vdash S \dashv (\Gamma'_2, Q'_2) \quad \Gamma'_2 \models \Gamma'_1 \quad Q'_2 \succeq_{\Gamma'_2} Q'_1}{B; R; (\Gamma_1, Q_1) \vdash S \dashv (\Gamma'_1, Q'_1)}\ \text{(Q:Weak)}$$

$$\frac{\begin{array}{c} \mathcal{L} = \{xy \mid \exists l_{xy} {\in} \mathbb{N}\,.\,\Gamma \models l_{xy} \leqslant |[x, y]|\} \quad \mathcal{U} = \{xy \mid \exists u_{xy} {\in} \mathbb{N}\,.\,\Gamma \models |[x, y]| \leqslant u_{xy}\} \quad \forall i.\, p_i, r_i \in \mathbb{Q}_0^+ \\ \forall i \in \mathcal{U}.\, q'_i \geqslant q_i - r_i \quad \forall i \in \mathcal{L}.\, q'_i \geqslant q_i + p_i \quad \forall i \notin \mathcal{U} {\cup} \mathcal{L} {\cup} \{0\}.\, q'_i \geqslant q_i \quad q'_0 \geqslant q_0 + \sum_{i \in \mathcal{U}} u_i r_i - \sum_{i \in \mathcal{L}} l_i p_i \end{array}}{Q' \succeq_\Gamma Q}\ \text{(Relax)}$$

Figure 4: Inference rules of the quantitative analysis.

Example Derivation

$\{\cdot;\, 0 + T\cdot|[x, y]|\}$
```
while (x < y) {
```
$\quad\{x < y;\ 0 + T\cdot|[x, y]|\}$
```
    x = x + 1;
```
$\quad\{x \leq y;\, T + T\cdot|[x, y]|\}$
```
    tick(T);
```
$\quad\{x \leq y;\, 0 + T\cdot|[x, y]|\}$
```
}
```
$\{x \geq y; 0 + T\cdot|[x, y]|\}$

Tarjan's Example From 50 Slides Ago

```
{·; 2 · |[0, n]| + |[0, s]|}
while (n > 0) {
    {n > 0; 2 · |[0, n]| + |[0, s]|}
    n−−;
    {·; 2 + 2 · |[0, n]| + |[0, s]|}
    if (*)
        s++;  {·; 1 + 2 · |[0, n]| + |[0, s]|}
        tick(1);  {·; 2 · |[0, n]| + |[0, s]|}
    else
        {·; 0 + 2 · |[0, n]| + |[0, s]|}
        while (s > 0)
            {s > 0; 2 · |[0, n]| + |[0, s]|}
            s−−;  {·; 1 + 2 · |[0, n]| + |[0, s]|}
            tick(1);  {·; 2 · |[0, n]| + |[0, s]|}
}
```

How to Automate?

Remark: All the constraints generated are *linear*.

- Apply the rules with dummy names for $(k_I)_I$.
- Collect all the constraints.
- Feed them the an LP solver.
- A solution is a proof certificate.
- No solution, report an error.

Implemented in C^4B, validated by the PLDI AEC 2015.

Future Work and Demo

What Now?

- (P) Automation for polynomial bounds.
- (TP) Extend automation to handle memory.
- (T) Prove logic completeness.
- (T) Integrate with contextual refinement.
- (P) Apply to real-time systems.

T is for theory, P is for practice.