Automatic Parallelization and Optimization of Programs by Proof Rewriting

Clément Hurlin

INRIA Sophia Antipolis - Méditerranée, France
Twente Universiteit, The Netherlands

January 8th 2008

Parallelism and Security (ParSec) meeting

Hongseok Yang:

“When you prove a program, you prove much more than you think”

Hongseok Yang:

"When you prove a program, you prove much more than you think"

In separation logic:

- Formulas denote heaps.
- $\Xi \star \Theta$ denotes the conjunction of two disjoint heaps Ξ and Θ.
- Proofs show how subheaps are used (and unused) by commands.

Hongseok Yang:

"When you prove a program, you prove much more than you think"

In separation logic:

- Formulas denote heaps.
- $\Xi \star \Theta$ denotes the conjunction of two disjoint heaps Ξ and Θ.
- Proofs show how subheaps are used (and unused) by commands.

↳ Unused subheaps are (Frame)d.

$$\frac{\{\Xi_a\}C\{\Xi_{a'}\}}{\{\Xi_a \star \Xi_f\}C\{\Xi_{a'} \star \Xi_f\}}\ (\text{Frame } \Xi_f)$$

↳ Command C does not access heap Ξ_f during execution.

In this work:

- We parallelize and optimize proven programs.
- We use the proof as an analysis:
  - Useless data discovery — (Frame) rule
  - Useful data discovery — antiframes
  - Alias analysis — $\star$ operator

In this work:

- We parallelize and optimize proven programs.
- We use the proof as an analysis:
  - Useless data discovery — (Frame) rule
  - Useful data discovery — antiframes
  - Alias analysis — $\star$ operator

$$\frac{\{\Xi_a\}C\{\Xi_{a'}\}}{\{\Xi_a \star \Xi_f\}C\{\Xi_{a'} \star \Xi_f\}}\ (\text{Frame } \Xi_f)$$

In this work:

- We parallelize and optimize proven programs.
- We use the proof as an analysis:
  - Useless data discovery — (Frame) rule
  - Useful data discovery — antiframes
  - Alias analysis — $\star$ operator

$$\frac{\{\Xi_a\}C\{\Xi_{a'}\}}{\{\Xi_a \star \Xi_f\}C\{\Xi_{a'} \star \Xi_f\}}\ (\text{Frame } \Xi_f)$$

- Optimizations are expressed with a rewrite system between proof trees.
- Proof trees are derivations of Hoare triplets.

Next slide: $\boxed{\text{an example}}$

- $x \mapsto [f : n]$ has a dual meaning:
  - $x.f$ contains value n.
  - Permission to write and read $x.f$.

Next slide: $\boxed{\text{an example}}$

- $x \mapsto [f : n]$ has a dual meaning:
  - $x.f$ contains value n.
  - Permission to write and read $x.f$.

- The example is ugly.
- But look at the $\boxed{\text{commands at the root of the trees.}}$

$$\boxed{\Lambda^{n,m,k}_{x,y,z} \triangleq x \mapsto [f:n] \star y \mapsto [f:m] \star z \mapsto [f:k]}$$

$$\dfrac{\dfrac{\dfrac{}{\{\Lambda^{-}_{x}\}x \longrightarrow f = n\{\Lambda^{n}_{x}\}}\,\text{(Mutate)}}{\{\Lambda^{-,-,-}_{x,y,z}\}x \longrightarrow f = n\{\Lambda^{n,-,-}_{x,y,z}\}}\,(\text{Fr }\Lambda^{-,-}_{y,z}) \qquad \dfrac{\dfrac{\dfrac{}{\{\Lambda^{-}_{y}\}y \longrightarrow f = m\{\Lambda^{m}_{y}\}}\,\text{(Mutate)}}{\{\Lambda^{n,-,-}_{x,y,z}\}y \longrightarrow f = m\{\Lambda^{n,m,-}_{x,y,z}\}}\,(\text{Fr }\Lambda^{n,-}_{x,z}) \quad \dfrac{\dfrac{}{\{\Lambda^{-}_{z}\}z \longrightarrow f = k\{\Lambda^{k}_{z}\}}\,\text{(Mutate)}}{\{\Lambda^{n,m,-}_{x,y,z}\}z \longrightarrow f = k\{\Lambda^{n,m,k}_{x,y,z}\}}\,(\text{Fr }\Lambda^{n,m}_{x,y})}{\{\Lambda^{n,-,-}_{x,y,z}\}y \longrightarrow f = m; z \longrightarrow f = k\{\Lambda^{n,m,k}_{x,y,z}\}}\,\text{(Seq)}}{\{\Lambda^{-,-,-}_{x,y,z}\}x \longrightarrow f = n; y \longrightarrow f = m; z \longrightarrow f = k\{\Lambda^{n,m,k}_{x,y,z}\}}\,\text{(Seq)}$$

$$\downarrow$$

$$\dfrac{\dfrac{}{\{\Lambda^{-}_{x}\}x \longrightarrow f = n\{\Lambda^{n}_{x}\}}\,\text{(Mutate)} \qquad \dfrac{\dfrac{}{\{\Lambda^{-}_{y}\}y \longrightarrow f = m\{\Lambda^{m}_{y}\}}\,\text{(Mutate)} \quad \dfrac{}{\{\Lambda^{-}_{z}\}z \longrightarrow f = k\{\Lambda^{k}_{z}\}}\,\text{(Mutate)}}{\{\Lambda^{-,-}_{y,z}\}y \longrightarrow f = m \parallel z \longrightarrow f = k\{\Lambda^{m,k}_{y,z}\}}\,\text{(Parallel)}}{\{\Lambda^{-,-,-}_{x,y,z}\}x \longrightarrow f = n \parallel (y \longrightarrow f = m \parallel z \longrightarrow f = k)\{\Lambda^{n,m,k}_{x,y,z}\}}\,\text{(Parallel)}$$

$$\boxed{\Lambda^{n,m,k}_{x,y,z} \triangleq x \mapsto [f : n] \star y \mapsto [f : m] \star z \mapsto [f : k]}$$

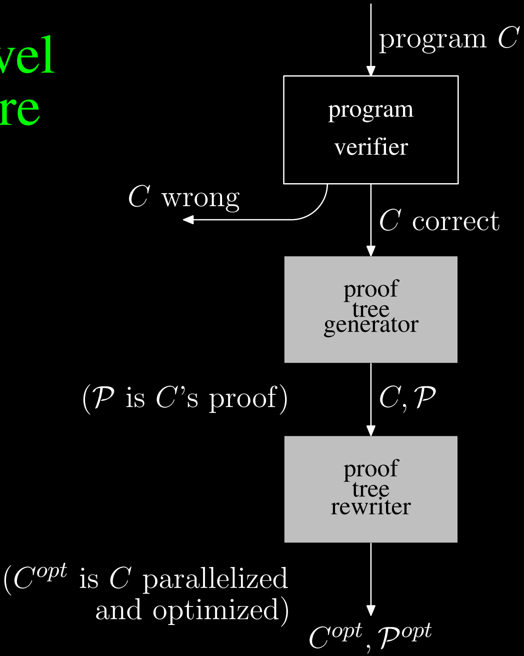
$$\dfrac{\dfrac{\dfrac{}{\{\Lambda^{-}_{x}\}x \longrightarrow f = n\{\Lambda^{n}_{x}\}}\,\text{(Mutate)}}{\{\Lambda^{-,-,-}_{x,y,z}\}x \longrightarrow f = n\{\Lambda^{n,-,-}_{x,y,z}\}}\,(\text{Fr } \Lambda^{-,-}_{y,z}) \qquad \dfrac{\dfrac{\dfrac{}{\{\Lambda^{-}_{y}\}y \longrightarrow f = m\{\Lambda^{m}_{y}\}}\,\text{(Mutate)}}{\{\Lambda^{n,-,-}_{x,y,z}\}y \longrightarrow f = m\{\Lambda^{n,m,-}_{x,y,z}\}}\,(\text{Fr } \Lambda^{n,-}_{x,z}) \qquad \dfrac{\dfrac{}{\{\Lambda^{-}_{z}\}z \longrightarrow f = k\{\Lambda^{k}_{z}\}}\,\text{(Mutate)}}{\{\Lambda^{n,m,-}_{x,y,z}\}z \longrightarrow f = k\{\Lambda^{n,m,k}_{x,y,z}\}}\,(\text{Fr } \Lambda^{n,m}_{x,y})}{\{\Lambda^{n,-,-}_{x,y,z}\}y \longrightarrow f = m; z \longrightarrow f = k\{\Lambda^{n,m,k}_{x,y,z}\}}\,\text{(Seq)}}{\{\Lambda^{-,-,-}_{x,y,z}\}x \longrightarrow f = n; y \longrightarrow f = m; z \longrightarrow f = k\{\Lambda^{n,m,k}_{x,y,z}\}}\,\text{(Seq)}$$

$$\downarrow$$

$$\dfrac{\dfrac{}{\{\Lambda^{-}_{x}\}x \longrightarrow f = n\{\Lambda^{n}_{x}\}}\,\text{(Mutate)} \qquad \dfrac{\dfrac{}{\{\Lambda^{-}_{y}\}y \longrightarrow f = m\{\Lambda^{m}_{y}\}}\,\text{(Mutate)} \qquad \dfrac{}{\{\Lambda^{-}_{z}\}z \longrightarrow f = k\{\Lambda^{k}_{z}\}}\,\text{(Mutate)}}{\{\Lambda^{-,-}_{y,z}\}y \longrightarrow f = m \parallel z \longrightarrow f = k\{\Lambda^{m,k}_{y,z}\}}\,\text{(Parallel)}}{\{\Lambda^{-,-,-}_{x,y,z}\}x \longrightarrow f = n \parallel (y \longrightarrow f = m \parallel z \longrightarrow f = k)\{\Lambda^{n,m,k}_{x,y,z}\}}\,\text{(Parallel)}$$

- Hypothesis: the lhs is a valid proof tree.
- Soundness follows from the inclusion of the rhs's leafs in the lhs's leafs.

High-Level Procedure



The rewrite system modifies programs but preserves specifications:

$$\frac{\mathscr{P}}{\{\Xi\}C\{\Theta\}}$$

$$\downarrow$$

$$\frac{\mathscr{Q}}{\{\Xi\}C'\{\Theta\}}$$

- The program and the the proof are modified: $\mathscr{P}, C \to \mathscr{Q}, C'$.
- But specifications are preserved: $\Xi, \Theta \to \Xi, \Theta$.

- The (Frame) rule is the central ingredient of our procedure.
- Problem: Existing program verifiers (e.g. smallfoot) do not make frames explicit.

- The (Frame) rule is the central ingredient of our procedure.
- Problem: Existing program verifiers (e.g. smallfoot) do not make frames explicit.

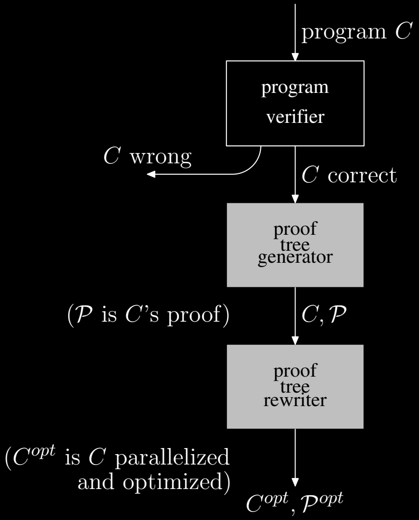
- Formulas Ξ, Θ are couples of a pure formula Π and a spatial formula Σ.
  - Π is a $\wedge$-conjoined list of variable equalities/inequalities.
  - Σ is $\star$-conjoined list of $\mapsto$ predicates (and more complex, irrelevant, formulas).

- The (Frame) rule is the central ingredient of our procedure.
- Problem: Existing program verifiers (e.g. smallfoot) do not make frames explicit.

$$\frac{\begin{array}{l} \ldots \\ x' \text{ fresh} \\ \Pi \vdash F = E \quad \{\ldots \wedge \Pi[x'/x] \mid (\Sigma \star F \mapsto [\rho])[x'/x]\} C \{\Pi' \mid \Sigma'\} \end{array}}{\{\Pi \mid (\Sigma \star F \mapsto [\rho])\} x := E \to f; C \{\Pi' \mid \Sigma'\}} \text{ (Lookup)}$$

- Substitutions x'/x affect the whole state: no explicit frame.
- Π is “too big”: there exists a “smaller” antiframe Π_s s.t. $\Pi_s \vdash F = E$.

Recall the big picture ?



In the proof tree generator:

- Proof rules with explicit frames.
- But still usage of the program's verifier normal rules for verification.

Next slide:

Proof rules with explicit {anti-,} frames

Berdine, Calcagno, and O'Hearn "Symbolic Execution with Separation Logic"

$$\frac{\begin{array}{l}\ldots \\ x' \text{ fresh} \\ \Pi \vdash F = E \qquad \{\ldots \wedge \Pi[x'/x] \mid (\Sigma \star F \mapsto [\rho])[x'/x]\}C\{\Pi' \mid \Sigma'\}\end{array}}{\{\Pi \mid \Sigma \star F \mapsto [\rho]\}x := E \to f;\, C\{\Pi' \mid \Sigma'\}} \text{(Lookup)}$$

With explicit frames and antiframes

$$\frac{\dfrac{\dfrac{\begin{array}{l} x' \text{ fresh} \qquad \ldots \\ \Pi_a \vdash F = E \\ \Xi = \Pi_a[x'/x] \wedge \ldots \mid (\Sigma_a \star F \mapsto [\rho])[x'/x] \end{array}}{\{\Pi_a \mid \Sigma_a \star F \mapsto [\rho]\}x := E \to f\{\Xi\} \qquad x \notin \Xi_f} \text{(Lookup)}}{\{(\Pi_a \mid \Sigma_a \star F \mapsto [\rho]) \star \Xi_f\}x := E \to f\{\Xi \star \Xi_f\}} \text{(Frame } \Xi_f\text{)} \qquad \{\Xi \star \Xi_f\}C\{\Xi'\}}{\{\underbrace{(\Pi_a \mid \Sigma_a \star F \mapsto [\rho])}_{\text{antiframe needed to prove } E \mapsto [\rho] \text{ and affected by } [x'/x]} \star \underbrace{\Xi_f}_{\text{frame unaffected by } [x'/x]}\}x := E \to f;\, C\{\Xi'\}} \text{(Seq}$$

Berdine, Calcagno, and O'Hearn "Symbolic Execution with Separation Logic"

$$\frac{x' \text{ fresh} \quad \{x = E[x'/x] \wedge \Pi[x'/x] \mid \Sigma[x'/x]\} C \{\Pi' \mid \Sigma'\}}{\{\Pi \mid \Sigma\} x := E; C \{\Pi' \mid \Sigma'\}} \text{ (Assign)}$$

- Same problem: substitutions $[x'/x]$ affect the whole state.

With explicit frames and antiframes

$$\dfrac{\dfrac{\dfrac{x' \text{ fresh}}{\{\Xi_a\} x := E \{\Xi_a[x'/x]\}}\text{ (Assign)} \quad x \notin \Xi_f}{\{\Xi_a \star \Xi_f\} x := E \{\Xi_a[x'/x] \star \Xi_f\}}\text{ (Frame } \Xi_f\text{)} \quad \{\Xi_a[x'/x] \star \Xi_f\} C \{\Xi'\}}{\{\underbrace{\Xi_a}_{\text{antiframe affected by [x'/x]}} \star \underbrace{\Xi_f}_{\text{frame unaffected by [x'/x]}}\} x := E; C \{\Xi'\}}\text{ (Seq)}$$

Berdine, Calcagno, and O'Hearn "Symbolic Execution with Separation Logic"

$$\frac{\Pi \vdash \bot}{\{\Pi \mathbin{¦} \Sigma\}C\{\Theta\}}\ \text{(Inconsistent)}$$

With explicit frames and antiframes

$$\frac{\dfrac{\Pi_a \vdash \bot}{\{\Pi_a \mathbin{¦} \mathtt{emp}\}C\{\Theta\}}\ \text{(Inconsistent)}}{\{\underbrace{\Pi_a \mathbin{¦} \mathtt{emp}}_{\text{sufficient antiframe to prove } \bot} \star \underbrace{\Pi_f \mathbin{¦} \Sigma_f}_{\text{frame}}\}C\{\Theta\}}\ (\text{Frame } \Pi_f \mathbin{¦} \Sigma_f)$$

- $\mathtt{emp} \triangleq$ the heap is empty.

- The proof tree generator implements rules with explicit frames and antiframes.
- Written as an extension of the program verifier (= smallfoot).

$\boxed{\text{Problem: proof trees generated with these rules have a special shape}}$

For successive atomic commands A, trees have the following shape:

$$\dfrac{\dfrac{\dots}{\{\dots\}A_0\{\dots\}}\,(\text{Frame}) \qquad \dfrac{\dfrac{\dots}{\{\dots\}A_1\{\dots\}}\,(\text{Frame}) \qquad \dfrac{\dfrac{\dots}{\{\dots\}A_2\{\dots\}}\,(\text{Frame}) \qquad \dfrac{}{\dots\vdash\dots}\,(\text{Empty})}{\{\dots\}A_2\{\dots\}}\,(\text{Seq})}{\{\dots\}A_1;A_2\{\dots\}}\,(\text{Seq})}{\{\dots\}A_0;A_1;A_2\{\dots\}}\,(\text{Seq})$$

- A (Frame) at each atomic command.

↳ Problem: Frames are redundant

For successive atomic commands A, trees have the following shape:

$$\dfrac{\dfrac{\dots}{\{\dots\}A_0\{\dots\}}\,(\text{Frame}) \qquad \dfrac{\dfrac{\dots}{\{\dots\}A_1\{\dots\}}\,(\text{Frame}) \qquad \dfrac{\dfrac{\dots}{\{\dots\}A_2\{\dots\}}\,(\text{Frame}) \qquad \dfrac{}{\dots \vdash \dots}\,(\text{Empty})}{\{\dots\}A_2\{\dots\}}\,(\text{Seq})}{\{\dots\}A_1;A_2\{\dots\}}\,(\text{Seq})}{\{\dots\}A_0;A_1;A_2\{\dots\}}\,(\text{Seq})$$

- A (Frame) at each atomic command.

↳ Problem: Frames are redundant

$$\dfrac{\dfrac{\dfrac{}{\{\Lambda^{-}_{x}\}x \to f = n\{\Lambda^{n}_{x}\}}\,(\text{Mutate})}{\{\Lambda^{-,-,-}_{x,y,z}\}x \to f = n\{\Lambda^{n,-,-}_{x,y,z}\}}\,(\text{Fr }\Lambda^{-,-}_{y,z}) \qquad \dfrac{\dfrac{\dfrac{}{\{\Lambda^{-}_{y}\}y \to f = m\{\Lambda^{m}_{y}\}}\,(\text{Mutate})}{\{\Lambda^{n,-,-}_{x,y,z}\}y \to f = m\{\Lambda^{n,m,-}_{x,y,z}\}}\,(\text{Fr }\Lambda^{n,-}_{x,z}) \qquad \dfrac{\dfrac{}{\{\Lambda^{-}_{z}\}z \to f = k\{\Lambda^{k}_{z}\}}\,(\text{Mutate})}{\{\Lambda^{n,m,-}_{x,y,z}\}z \to f = k\{\Lambda^{n,m,k}_{x,y,z}\}}\,(\text{Fr }\Lambda^{n,m}_{x,y})}{\{\Lambda^{n,-,-}_{x,y,z}\}y \to f = m;\, z \to f = k\{\Lambda^{n,m,k}_{x,y,z}\}}\,(\text{Seq})}{\{\Lambda^{-,-,-}_{x,y,z}\}x \to f = n;\, y \to f = m;\, z \to f = k\{\Lambda^{n,m,k}_{x,y,z}\}}\,(\text{Seq})$$

- We rewrite proof trees to frame multiple commands (= frame factorization).

- We rewrite proof trees to frame multiple commands (= frame factorization).
- Below, Ξ_c is the factorized frame.

Guard: $\Xi_f \Leftrightarrow \Xi_{f_0} \star \Xi_c$ and $\Theta_f \Leftrightarrow \Theta_{f_0} \star \Xi_c$

$$\dfrac{\dfrac{\{\Xi_a\}C\{\Xi_p\}}{\{\Xi_a \star \Xi_f\}C\{\Xi_p \star \Xi_f\}}\text{(Fr } \Xi_f\text{)} \quad \dfrac{\dfrac{\{\Theta_a\}C'\{\Theta_p\}}{\{\Theta_a \star \Theta_f\}C'\{\Theta_p \star \Theta_f\}}\text{(Fr } \Theta_f\text{)} \quad \{\Theta_p \star \Theta_f\}C''\{\Xi'\}}{\{\Theta_a \star \Theta_f\}C'; C''\{\Xi'\}}\text{(Seq)}}{\{\Xi_a \star \Xi_f\}C; C'; C''\{\Xi'\}}\text{(Seq)}$$

$\downarrow$ <u>FactorizeFrames</u>

$$\dfrac{\dfrac{\dfrac{\dfrac{\{\Xi_a\}C\{\Xi_p\}}{\{\Xi_a \star \Xi_{f_0}\}C\{\Xi_p \star \Xi_{f_0}\}}\text{(Fr } \Xi_{f_0}\text{)} \quad \dfrac{\dfrac{\{\Theta_a\}C'\{\Theta_p\} \quad \dfrac{}{\Theta_p \vdash \Theta_p}\text{(Empty)}}{\{\Theta_a\}C'\{\Theta_p\}}\text{(Seq)}}{\{\Theta_a \star \Theta_{f_0}\}C'\{\Theta_p \star \Theta_{f_0}\}}\text{(Fr } \Theta_{f_0}\text{)}}{\{\Xi_a \star \Xi_{f_0}\}C; C'\{\Theta_p \star \Theta_{f_0}\}}\text{(Seq)}}{\{\Xi_a \star \Xi_f\}C; C'\{\Theta_p \star \Theta_f\}}\text{(Fr } \Xi_c\text{)} \quad \{\Theta_p \star \Theta_f\}C''\{\Xi'\}}{\{\Xi_a \star \Xi_f\}C; C'; C''\{\Xi'\}}\text{(Seq)}$$

- Example of frame factorization:

$$
\dfrac{
\dfrac{\dfrac{}{\{\Lambda^{-}_{x}\}x \to f = n\{\Lambda^{n}_{x}\}}\,(\text{Mutate})}{\{\Lambda^{-,-,-}_{x,y,z}\}x \to f = n\{\Lambda^{n,-,-}_{x,y,z}\}}\,(\text{Fr }\Lambda^{-,-}_{y,z})
\quad
\dfrac{
\dfrac{\dfrac{}{\{\Lambda^{-}_{y}\}y \to f = m\{\Lambda^{m}_{y}\}}\,(\text{Mutate})}{\{\Lambda^{n,-,-}_{x,y,z}\}y \to f = m\{\Lambda^{n,m,-}_{x,y,z}\}}\,(\text{Fr }\Lambda^{n,-}_{x,z})
\quad
\dfrac{\dfrac{}{\{\Lambda^{-}_{z}\}z \to f = k\{\Lambda^{k}_{z}\}}\,(\text{Mutate})}{\{\Lambda^{n,m,-}_{x,y,z}\}z \to f = k\{\Lambda^{n,m,k}_{x,y,z}\}}\,(\text{Fr }\Lambda^{n,m}_{x,y})
}{\{\Lambda^{n,-,-}_{x,y,z}\}y \to f = m; z \to f = k\{\Lambda^{n,m,k}_{x,y,z}\}}\,(\text{Seq})
}{\{\Lambda^{-,-,-}_{x,y,z}\}x \to f = n; y \to f = m; z \to f = k\{\Lambda^{n,m,k}_{x,y,z}\}}\,(\text{Seq})
$$

$$\downarrow$$

$$
\dfrac{
\dfrac{\dfrac{}{\{\Lambda^{-}_{x}\}x \to f = n\{\Lambda^{n}_{x}\}}\,(\text{Mutate})}{\{\Lambda^{-,-,-}_{x,y,z}\}x \to f = n\{\Lambda^{n,-,-}_{x,y,z}\}}\,(\text{Fr }\Lambda^{-,-}_{y,z})
\quad
\dfrac{
\dfrac{
\dfrac{\dfrac{}{\{\Lambda^{-}_{y}\}y \to f = m\{\Lambda^{m}_{y}\}}\,(\text{Mutate})}{\{\Lambda^{-,-}_{y,z}\}y \to f = m\{\Lambda^{m,-}_{y,z}\}}\,(\text{Fr }\Lambda^{-}_{z})
\quad
\dfrac{\dfrac{}{\{\Lambda^{-}_{z}\}z \to f = k\{\Lambda^{k}_{z}\}}\,(\text{Mutate})}{\{\Lambda^{m,-}_{y,z}\}z \to f = k\{\Lambda^{m,k}_{y,z}\}}\,(\text{Fr }\Lambda^{m}_{y})
}{\{\Lambda^{-,-}_{y,z}\}y \to f = m; z \to f = k\{\Lambda^{n,m}_{y,z}\}}\,(\text{Seq})
}{\{\Lambda^{n,-,-}_{x,y,z}\}y \to f = m; z \to f = k\{\Lambda^{n,m,k}_{x,y,z}\}}\,(\text{Fr }\Lambda^{n}_{x})
}{\{\Lambda^{-,-,-}_{x,y,z}\}x \to f = n; y \to f = m; z \to f = k\{\Lambda^{n,m,k}_{x,y,z}\}}\,(\text{Seq})
$$

With factorized frames: optimizations are (quite) simple to express

$$\dfrac{\dfrac{\{\Xi\}C\{\Theta\}}{\{\Xi\star\Xi'\}C\{\Theta\star\Xi'\}}\;(\text{Frame } \Xi') \qquad \dfrac{\dfrac{\{\Xi'\}C'\{\Theta'\}}{\{\Theta\star\Xi'\}C'\{\Theta\star\Theta'\}}\;(\text{Frame } \Theta) \qquad \{\Theta\star\Theta'\}C''\{\Xi''\}}{\{\Theta\star\Xi'\}C';\,C''\{\Xi''\}}}{\{\Xi\star\Xi'\}C;\,C';\,C''\{\Xi''\}}\;(\text{Seq})$$

$$\downarrow\underline{\text{Parallelize}}$$

$$\dfrac{\dfrac{\{\Xi\}C\{\Theta\} \qquad \{\Xi'\}C'\{\Theta'\}}{\{\Xi\star\Xi'\}C \parallel C'\{\Theta\star\Theta'\}}\;(\text{Parallel}) \qquad \{\Theta\star\Theta'\}C''\{\Xi''\}}{\{\Xi\star\Xi'\}C \parallel C';\,C''\{\Xi''\}}\;(\text{Seq})$$

With factorized frames: optimizations are (quite) simple to express

$$\dfrac{\dfrac{\{\Xi\}C\{\Theta\}}{\{\Xi\star\Xi'\}C\{\Theta\star\Xi'\}}\,(\text{Frame } \Xi') \qquad \dfrac{\dfrac{\{\Xi'\}C'\{\Theta'\}}{\{\Theta\star\Xi'\}C'\{\Theta\star\Theta'\}}\,(\text{Frame } \Theta) \qquad \{\Theta\star\Theta'\}C''\{\Xi''\}}{\{\Theta\star\Xi'\}C';\, C''\{\Xi''\}}}{\{\Xi\star\Xi'\}C;\, C';\, C''\{\Xi''\}}\,(\text{Seq})$$

$$\downarrow\underline{\text{Parallelize}}$$

$$\dfrac{\dfrac{\{\Xi\}C\{\Theta\} \qquad \{\Xi'\}C'\{\Theta'\}}{\{\Xi\star\Xi'\}C \parallel C'\{\Theta\star\Theta'\}}\,(\text{Parallel}) \qquad \{\Theta\star\Theta'\}C''\{\Xi''\}}{\{\Xi\star\Xi'\}C \parallel C';\, C''\{\Xi''\}}\,(\text{Seq})$$

- $\{\Theta\star\Theta'\}C''\{\Xi''\}$ can be "dummy" i.e., a single application of (Empty).

↳ Or C'' can be a "normal" continuation.

- This rule matches the two cases.

Optimizations include:

- parallelization (previous slide)
- early disposal and late allocation (omitted in this talk)
- early lock releasing and late lock acquirement (next 2 slides)
- improvement of temporal locality (3^{th} slide)

Locks in Separation Logic

- Each lock guards a part of the heap called the lock's resource invariant.
- Resource invariants are exchanged between locks and threads:
  - When a lock is acquired, it lends its resource invariant to the acquiring thread.
  - When a lock is released, it claims back its resource invariant from the releasing thread.

Locks in Separation Logic

- Each lock guards a part of the heap called the lock's resource invariant.
- Resource invariants are exchanged between locks and threads:
  - When a lock is acquired, it lends its resource invariant to the acquiring thread.
  - When a lock is released, it claims back its resource invariant from the releasing thread.

Formally (where r_Θ means Θ is r's resource invariant):

$$\frac{\{\Xi \star \Theta\} C \{\Xi' \star \Theta\}}{\{\Xi\}\texttt{with}\ r_\Theta\ \texttt{do}\ C\ \texttt{endwith}\{\Xi'\}}\ \text{(Region)}$$

- Intuition: execute as much code as possible outside critical regions,

↳ by releasing locks as soon as possible

$$
\dfrac{
  \dfrac{
    \dfrac{
      \{\Xi \star \Theta\} C \{\Xi' \star \Theta\}
      \qquad
      \dfrac{\{\Xi'\} C' \{\Xi''\}}{\{\Xi' \star \Theta\} C' \{\Xi'' \star \Theta\}}\;(\text{Frame } \Theta)
    }{
      \{\Xi \star \Theta\} C; C' \{\Xi'' \star \Theta\}
    }\;(\text{Seq})
  }{
    \{\Xi\}\mathtt{with}\ r_{\Theta}\ \mathtt{do}\ C;\ C'\ \mathtt{endwith}\{\Xi''\}
  }\;(\text{Region})
  \qquad
  \{\Xi''\} C'' \{\Xi'''\}
}{
  \{\Xi\}\mathtt{with}\ r_{\Theta}\ \mathtt{do}\ C;\ C'\ \mathtt{endwith};\ C'' \{\Xi'''\}
}\;(\text{Seq})
$$

↓ <u>EarlyLockReleasing</u>

$$
\dfrac{
  \dfrac{
    \dfrac{
      \{\Xi \star \Theta\} C \{\Xi' \star \Theta\}
    }{
      \{\Xi\}\mathtt{with}\ r_{\Theta}\ \mathtt{do}\ C\ \mathtt{endwith}\{\Xi'\}
    }\;(\text{Region})
    \qquad
    \{\Xi'\} C' \{\Xi''\}
  }{
    \{\Xi\}\mathtt{with}\ r_{\Theta}\ \mathtt{do}\ C\ \mathtt{endwith};\ C' \{\Xi''\}
  }\;(\text{Seq})
  \qquad
  \{\Xi''\} C'' \{\Xi'''\}
}{
  \{\Xi\}\mathtt{with}\ r_{\Theta}\ \mathtt{do}\ C\ \mathtt{endwith};\ C';\ C'' \{\Xi'''\}
}\;(\text{Seq})
$$

- Intuition: execute as much code as possible outside critical regions,

↳ by acquiring locks as late as possible

$$\dfrac{\dfrac{\dfrac{\dfrac{\{\Xi\}C\{\Xi'\}}{\{\Xi\star\Theta\}C\{\Xi'\star\Theta\}}\,(\text{Frame }\Theta) \qquad \{\Xi'\star\Theta\}C'\{\Xi''\star\Theta\}}{\{\Xi\star\Theta\}C;\,C'\{\Xi''\star\Theta\}}\,(\text{Seq})}{\{\Xi\}\texttt{with}\ r_\Theta\ \texttt{do}\ C;\,C'\ \texttt{endwith}\{\Xi''\}}\,(\text{Region}) \qquad \{\Xi''\}C''\{\Xi'''\}}{\{\Xi\}\texttt{with}\ r_\Theta\ \texttt{do}\ C;\,C'\ \texttt{endwith};\,C''\{\Xi'''\}}\,(\text{Seq})$$

$\downarrow$ <u>LateLockAcquirement</u>

$$\dfrac{\dfrac{\{\Xi\}C\{\Xi'\} \qquad \dfrac{\{\Xi'\star\Theta\}C'\{\Xi''\star\Theta\}}{\{\Xi'\}\texttt{with}\ r_\Theta\ \texttt{do}\ C'\ \texttt{endwith}\{\Xi''\}}\,(\text{Region})}{\{\Xi\}C;\,\texttt{with}\ r_\Theta\ \texttt{do}\ C'\ \texttt{endwith}\{\Xi''\}} \qquad \{\Xi''\}C''\{\Xi'''\}}{\{\Xi\}C;\,\texttt{with}\ r_\Theta\ \texttt{do}\ C'\ \texttt{endwith};\,C''\{\Xi'''\}}\,(\text{Seq})$$

- temporal locality $\stackrel{\Delta}{=}$ time between two accesses to the same heap cell

↳ the smaller the better (no need to free/load processors's caches)

- Intuition below: C and C'' access the same part of the heap

↳ Execute them successively

$$
\dfrac{
  \dfrac{
    \dfrac{\{\Xi\}C\{\Xi'\}}{\{\Xi \star \Theta\}C\{\Xi' \star \Theta\}}\,(\text{Fr } \Theta)
    \qquad
    \dfrac{\{\Theta\}C'\{\Theta'\}}{\{\Xi' \star \Theta\}C'\{\Xi' \star \Theta'\}}\,(\text{Fr } \Xi')
  }{\{\Xi \star \Theta\}C;\, C'\{\Xi' \star \Theta'\}}\,(\text{Seq})
  \qquad
  \dfrac{\{\Xi'\}C''\{\Xi''\}}{\{\Xi' \star \Theta'\}C''\{\Xi'' \star \Theta'\}}\,(\text{Fr } \Theta')
}{\{\Xi \star \Theta\}C;\, C';\, C''\{\Xi'' \star \Theta'\}}\,(\text{Seq})
$$

$\downarrow$ <u>TemporalLocality</u>

$$
\dfrac{
  \dfrac{
    \dfrac{\{\Xi\}C\{\Xi'\} \qquad \{\Xi'\}C''\{\Xi''\}}{\{\Xi\}C;\, C''\{\Xi''\}}\,(\text{Seq})
  }{\{\Xi \star \Theta\}C;\, C''\{\Xi'' \star \Theta\}}\,(\text{Fr } \Theta)
  \qquad
  \dfrac{\{\Theta\}C'\{\Theta'\}}{\{\Xi'' \star \Theta\}C'\{\Xi'' \star \Theta'\}}\,(\text{Fr } \Xi'')
}{\{\Xi \star \Theta\}C;\, C'';\, C'\{\Xi'' \star \Theta'\}}\,(\text{Seq})
$$

Implementation:

- The rewrite rules have been implemented in Java+ tom .
- tom extends Java to pattern match against tom/user-defined Java objects.
- The implementation is approximately 2775 lines long.
- Each rewrite rule is less than 58 lines of code (i.e. manageable).
- Use of tom's strategies to fine tune optimizations.

Demo

```
requires tree(t); ensures emp;
disp_tree(t) {
 local i,j;
 if (t = nil) {} else {
  i := t→l; j := t→r;
  disp_tree(i); disp_tree(j);
  dispose(t);
 }
}
```

→

```
requires tree(t); ensures emp;
disp_tree(t) {
 local i,j;
 if (t = nil) {} else {
  i := t→l; j := t→r;
  dispose(t) ∥
  (disp_tree(i) ∥ disp_tree(j));
 }
}
```

I’m looking for a post doc starting approx. September 2009

Conclusion

- A entirely new technique for parallelizing and optimizing programs
- No ad-hoc analyses: separation logic proofs are taken as analyses.
- Can parallelize any code (i.e. not restricted to loops)
- Soundness is still doable ($\neq$ *classical* parallelizers)
- Prototype implementation

Future Work

1. Formalize the shape of proof trees (is there a normal form ?)
2. Release the implementation (cleanup + missing optimizations)
3. Extension to fractional permissions (more parallelization possible)
4. Extension to Parkinson's per-field $\star$ splitting (more parallelization possible)
5. Extension to object orientation (subsumes (4))(to deal with abstraction)
6. More optimizations

But:

- Points (3), (4), and (5) require an (open-source) program verifier with these features.