

An Introduction to Region Inference

David Walker

(with slides stolen from Mads Tofte)

Living in a Civilized World

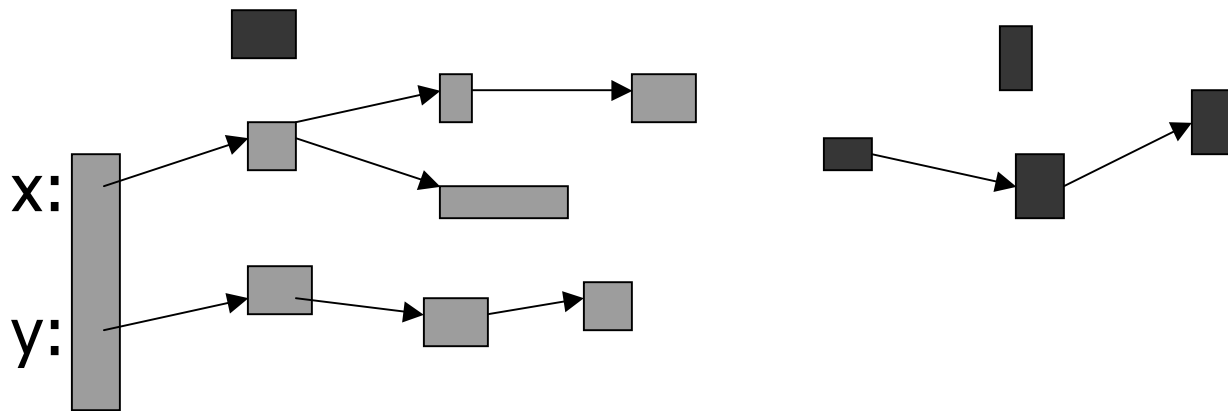
- civilized languages give us:
 - software engineering advantages
 - error checking via type safety
 - eg: Java, ML
 - security
 - provably sound enforcement mechanisms
 - Java virtual machine, TAL
- civilized languages require automatic memory management

Garbage Collection

- Strengths:
 - Relieves the programmer of (most of) the burden of managing memory
 - Effective in many applications

But ...

- Is garbage collection **THE** solution?
- No. GC is an **approximation**:



- Liveness is approximated by reachability
- True liveness is undecidable

Semantic Disadvantages

- Deallocation is not evident in the program text
 - How can we reason accurately about space?
- Garbage Collection takes time
 - How can we reason accurately about time?

(see Blelloc & Greiner PLDI '96, Minamide HOOTs '99)

Practical Problems

- Foxnet (CMU): a web server written in ML
 - last rebooted: 1 day, 3 hours, 9 minutes
 - plagued by space leaks (cite: jgm)
- Ensemble
 - required hacking the O’Caml runtime to allow it to do its own management of buffers
- In general, embedded systems need real-time guarantees
 - even real-time collectors have millisecond pauses

Stack-Based Memory Management

- Well-studied, rigorous semantics
 - Algol, Reynolds & O'Hearn, Stack-based TAL
- Predictability
 - Every point of allocation is matched by a point of deallocation *and the programmer knows where these are!*
- Eager re-use of memory
 - efficient in space & time
 - many compilers attempt stack-allocation optimizations (eg: Marmot, Java)

Limitations of the Stack

- Some lifetimes just do not follow a stack discipline
 - eg: "escaping" objects (ie: pairs, function closures returned from functions)
- Some values grow dynamically
 - eg: lists, trees

(trumpets sound now)

Region-Based MM

- Combines Stack-based + Heap-based MM
- Type-directed algorithm determines when to deallocate
- Liveness = Is an object accessed in the future?

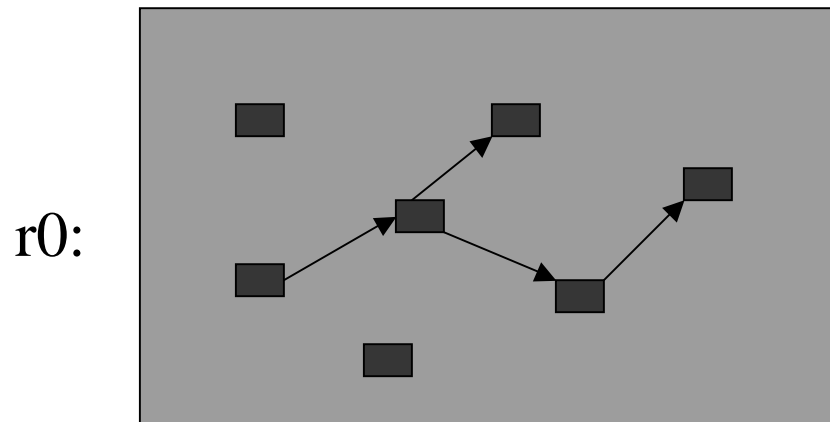
- Original theory by Tofte & Talpin
- Implemented by Tofte and others for SML
- Study continued by Tofte, Birkedal, Elsman, Aiken, me, ...

Outline

- Regions: what are they?
 - how do regions combine stack & heap allocation?
- Region Inference
 - how do we place region annotations on programs to direct allocation/deallocation
- Region Type System
 - how do we verify that deallocation doesn't happen too early?

Regions

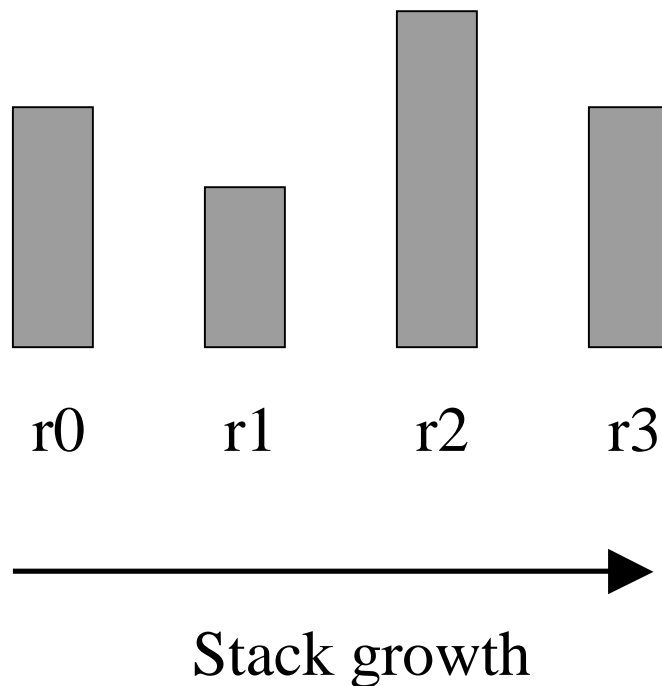
- A region is an area of memory:



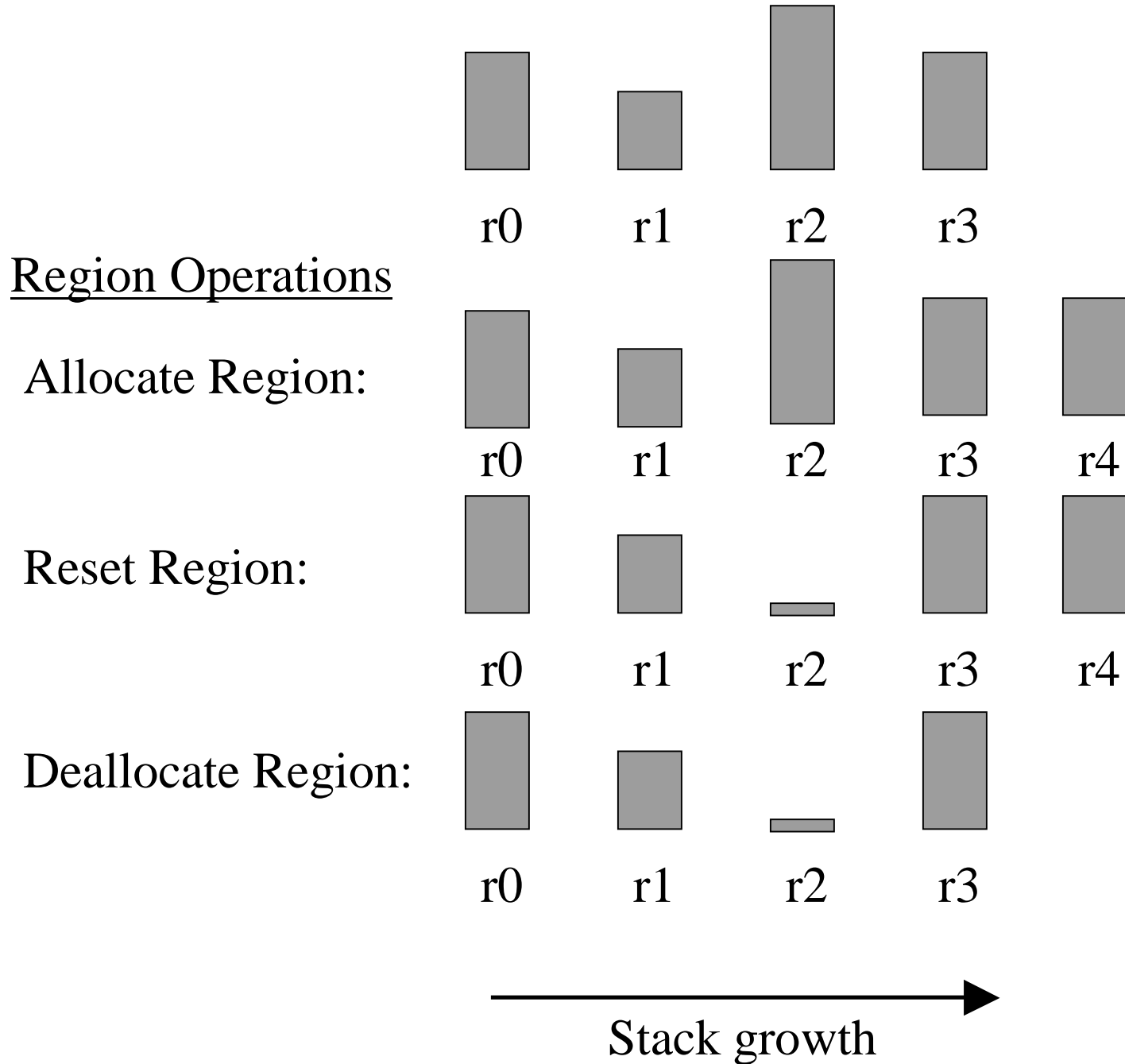
- All values are put into regions
- Some regions have known size, others do not

The Region Stack

- Regions are organized on a stack:

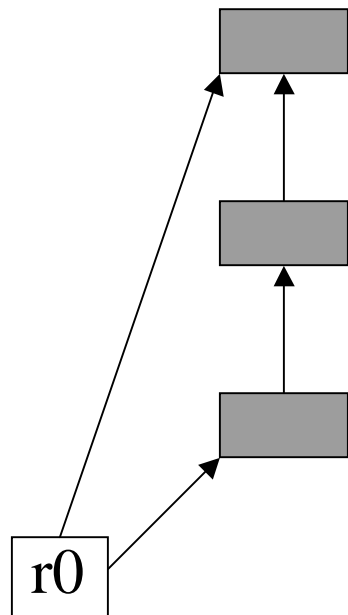


- Values can be put into any live region, not just the region on top



Region Implementation

- A region is a linked list of pages:



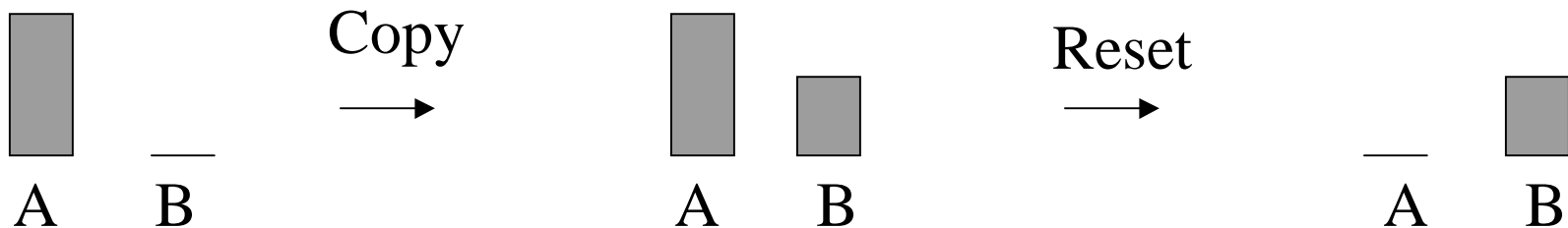
- Pointer to head of list, pointer to allocation point
- Runtime maintains free page list
- All region operations are constant-time
- Good for reasoning about time behaviour

Special Cases

- Stack-allocation
 - All regions have statically known size
 - No region resetting
- A single allocation space with no garbage collection
 - One region, unbounded size

Special Cases (Cont)

- A (synchronous) copying garbage collector
 - Two regions, A & B:
 - allocate into A
 - copy reachable data into B
 - reset A
 - allocate into B ...



Anecdotal Experience

- Anno Domini Project (DIKU)
 - finds Y2K bugs in Cobol programs
 - 60,000+ lines of code
 - space leaks in initial coding
 - Mads Tofte spent 2 days debugging and eliminated them all
 - Currently a marketed product
- How far will region inference really scale?

Summary

- Region inference is an alternative memory-management strategy
 - combines beneficial elements of heap and stack allocation strategies
 - makes it easier to reason about time and space
 - a type inference algorithm inserts region annotations into SML programs automatically
 - a type system ensures safety