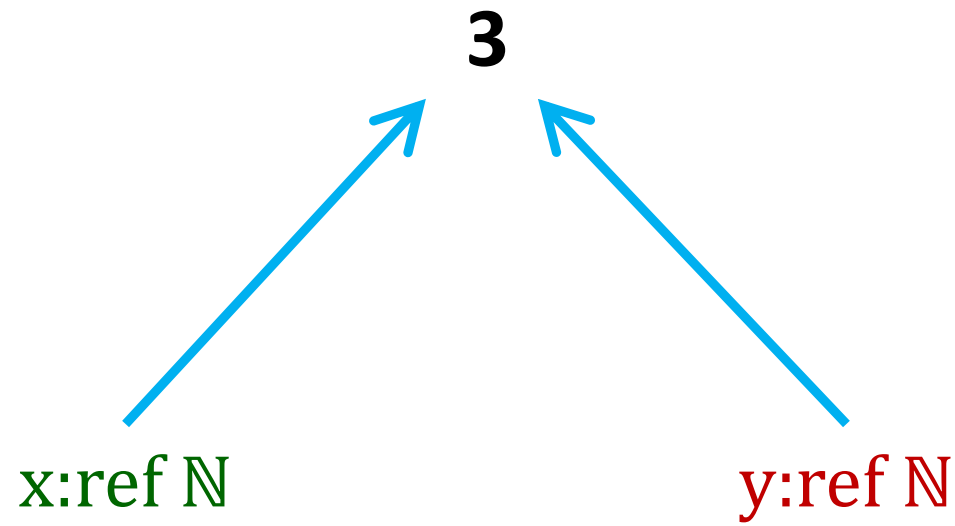
Rely-Guarantee References for Refinement Types over Aliased Mutable Data

Colin S. Gordon, Michael D. Ernst, and Dan Grossman
University of Washington

PLDI 2013

Static Verification Meets Aliasing




```
x := !x+1;
m(y);
static_assert(!x > 0);
```
← Pass or fail?

Static Program Verification: Mutation + Aliases

- Common approaches:
  - Restrict aliasing
    - Separation Logic, Linear & Uniqueness Types
  - Restrict mutation
    - Read-only access
    - Reference Immutability, Region & Effect Systems

Our Approach: Describe Mutation

- Arbitrary binary relations
- Explicitly characterize:
  - How data may change over time
    - Side effects, type state, protocols, invariants, monotonicity…
    - Lots of prior work, all working around aliasing
  - Safe assumptions in the presence of mutation through aliases
- Eye towards backwards compatibility:
  - Subsume standard references, reference immutability

Rely-Guarantee References

- New approach to static verification of imperative programs
- Formal core type system + proofs
  - Uses dependent types
- Implementation as Coq DSL
- Characterized proof burden for 4 examples
  - Roughly on par w/ pure functional equivalents

Outline

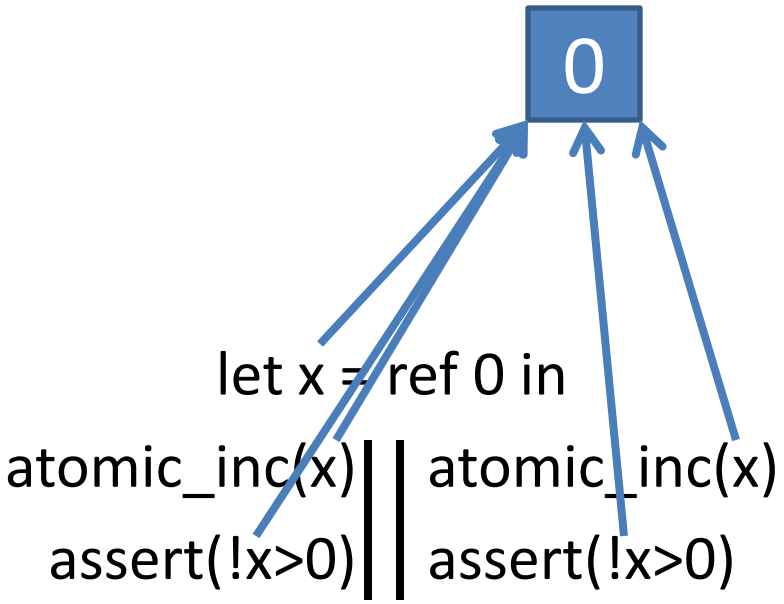
- Concurrent Reasoning for Aliases
- Typechecking Rely-Guarantee References
- Technical Challenge: Nested References
- Intuition for Soundness
- Conclusions

A Duality: Threads & Aliases

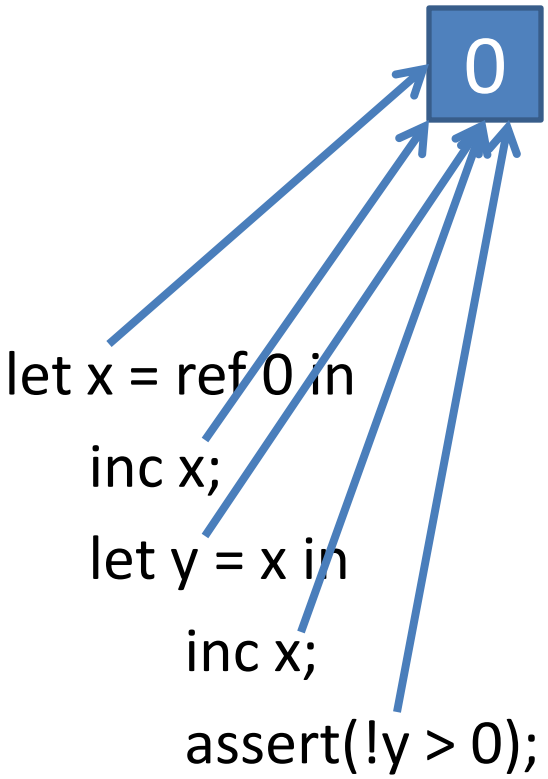
- Mutation by aliases ≈ thread interference
- Actions through aliases can be seen as concurrent
- Rely-Guarantee reasoning is good for threads
  - Summarizes possible interference
- We can adapt concurrent analyses to treat aliasing
  - A few differences to discuss later

Thread & Alias Interference

Thread Interference



Alias Interference



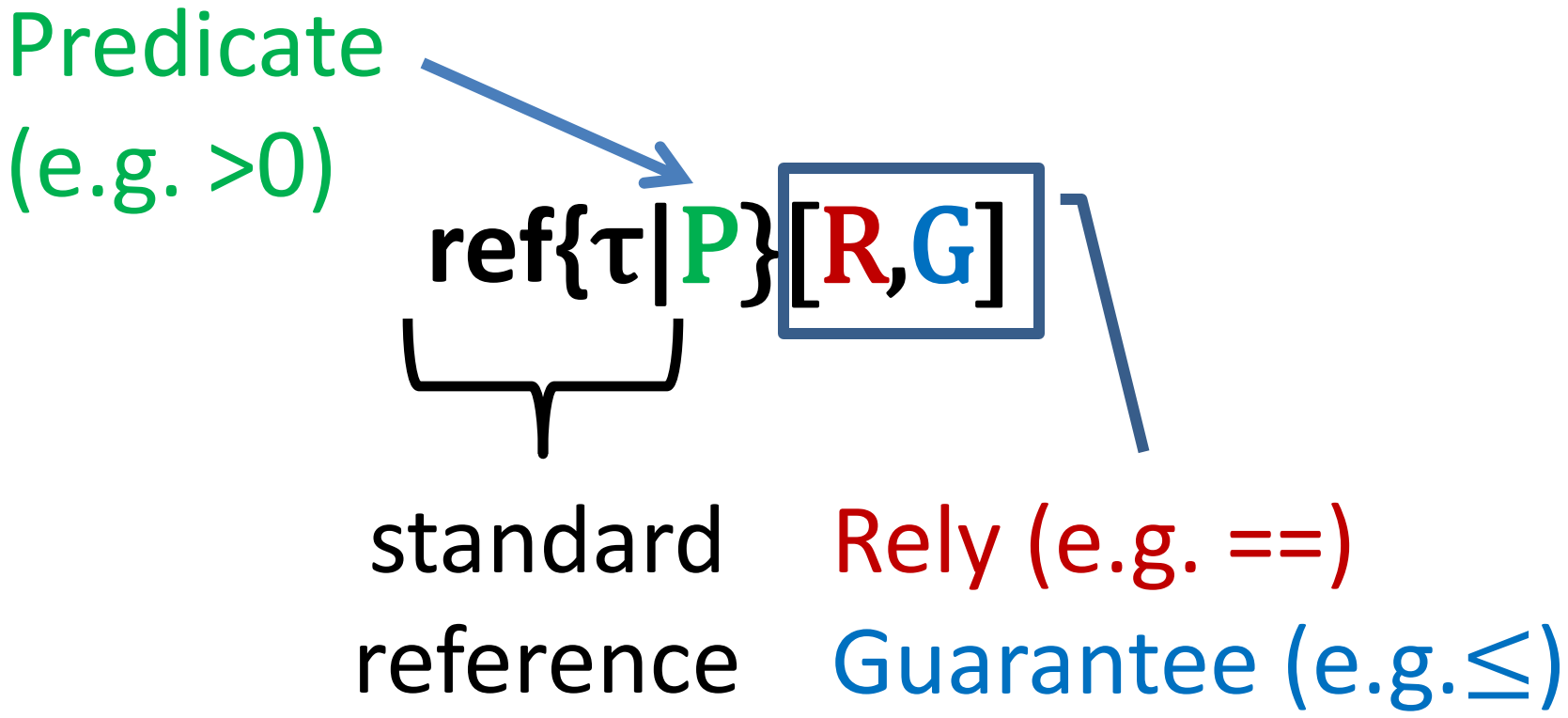
Rely-Guarantee for Threads

- Characterize thread interference:
  1. **Rely** summarizes expected interference
  2. **Guarantee** bounds thread actions
  3. **Stable** assertions are preserved by interference
  4. **Compatible** threads: each rely assumes at least the other's guarantee

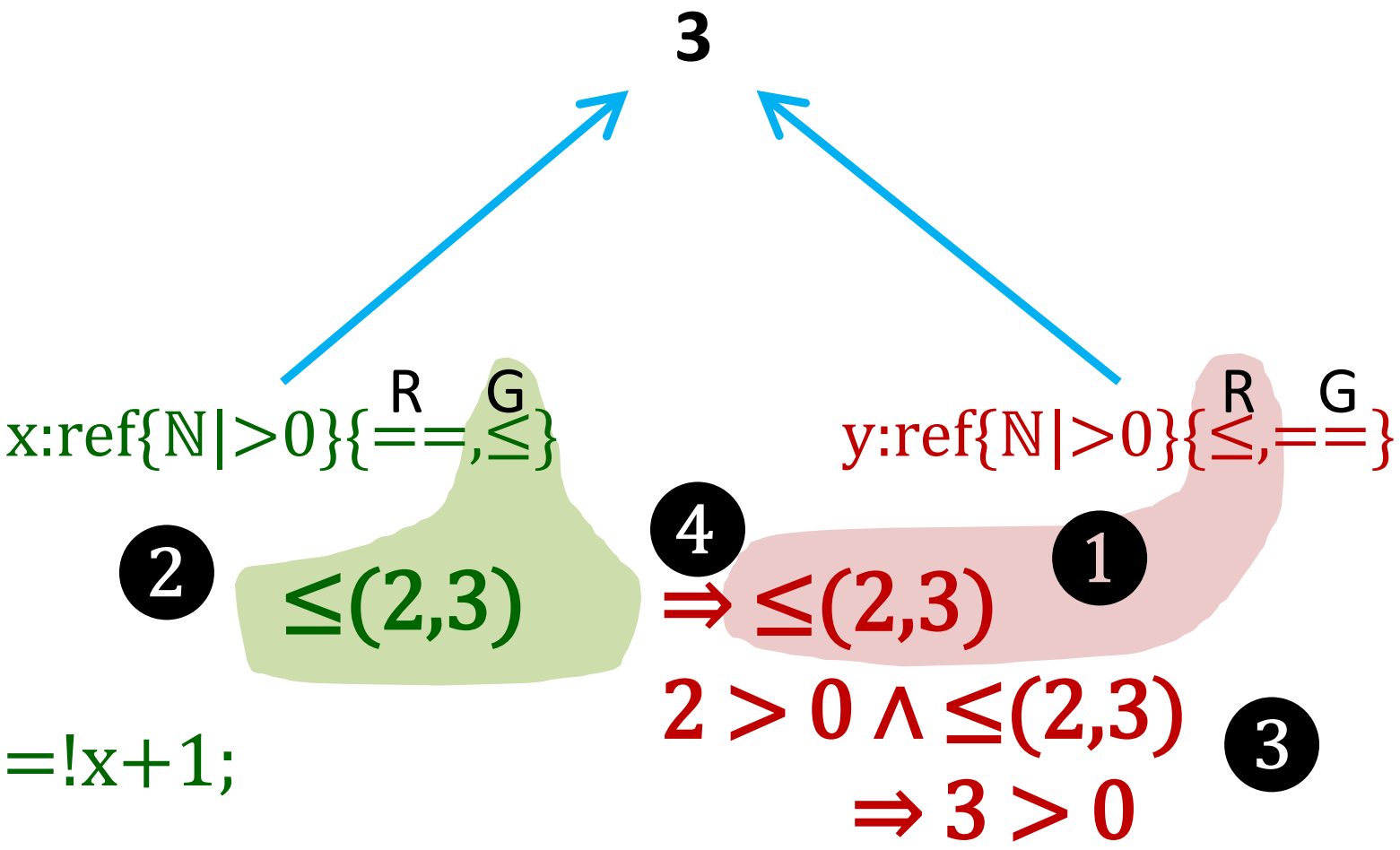
Rely-Guarantee for References

- Characterize alias interference:
  1. **Rely** summarizes alias interference
  2. **Guarantee** bounds actions through this alias
  3. **Stable** predicates preserved by interference
  4. **Compatible** aliases: if x == y, then
     $x.G \subseteq y.R \;\&\&\; y.G \subseteq x.R$
- Subsumes ML references! (OCaml, SML, etc.)

Rely-Guarantee Reference Type



Alias Interference Revisited



1 Rely, 2 Guarantee, 3 Stable, 4 Compatible

Splitting for Compatible References

- x:ref{nat|P}[≈,havoc] cannot be duplicated
  - Duplicates must be *compatible (#4)*
  - havoc ⋢ ≈
- Must track & restrict duplication:
  - let y = x in … could create
    - Two immutable refs (ref{nat|P}[≈, ≈])
    - Two unrestricted refs (ref{nat|any}[havoc, havoc])
    - Producer/consumer (ref{nat|any}[≥,≤] and ref{nat|any}[≤,≥])

Outline

- Concurrent Reasoning for Aliases
- Typechecking Rely-Guarantee References
- Technical Challenge: Nested References
- Intuition for Soundness
- Conclusions

A Coq DSL for Rely-Guarantee References

- Shallow DSL embedding in a proof assistant
- Satisfying proof obligations
  - Separated from program text
  - Semi-automatic
- Examples include
  - Monotonic Counter
    - Simple, but illustrative
    - Specify how data changes over time, not inc operation
  - Reference Immutability

Example: A Monotonic Counter

```
Definition increasing : hrel nat := (λ n n’ h h’. n <= n').
Definition counter := ref{nat|pos}[increasing,increasing].
Definition read (c:counter) : nat := !c.
Definition inc (p:counter) : unit :=
    [p]:= !p + 1.
Definition mkCounter (u:unit) : counter := Alloc 1.
Example test_counter (u:unit) : unit :=
 x <- mkCounter tt;
 inc x.
```

Proofs automatically discharged

Invalid Code: Decrement a Monotonic Counter

```
Definition counter :=
        ref{nat|pos}[increasing,increasing].
Definition dec (p:counter) : unit :=
        [p]:= !p - 1.
```

Unsatisfiable proof obligation:

$$\boldsymbol{increasing\ (!p)(!p-1) \leftrightarrow \forall n, n \leq n-1}$$

Reference Immutability via Rely-Guarantee References

- writable T $\stackrel{\text{def}}{=}$ ref{T|any}[havoc,havoc]
- readable T $\stackrel{\text{def}}{=}$ ref{T|any}[havoc,$\approx$]
- immutable T $\stackrel{\text{def}}{=}$ ref{T|any}[$\approx$, $\approx$]
- Suggests a spectrum:

$$\text{ML refs} \subseteq \text{RI} \subseteq \ldots \subseteq \text{RGref}$$

Outline

- Concurrent Reasoning for Aliases
- Typechecking Rely-Guarantee References
- Technical Challenge: Nested References
- Intuition for Soundness
- Conclusions

References to References

- Folding
  - If x.f is a ref, and x's type disallows mutation to anything, type of !x.f should, too
- Containment
  - ref{T|P}[R,G]: if T contains refs, R permits their interference as well
- Precision
  - P,R,G only depend on heap reachable from the T they apply to

Non-issues in concurrent program logics:

no "threads to threads"

Outline

- Concurrent Reasoning for Aliases
- Typechecking Rely-Guarantee References
- Technical Challenge: Nested References
- Intuition for Soundness
- Conclusions

How to Preserve Refinements

- Well-formed ref{τ|P}[R,G]
  (e.g. ref{int|>0}[$\leq$,$\leq$])
  - P is *stable* with respect to R (#3)
  - Enforce containment, precision
- Aliases as x:ref{τ|P}[R,G] and y:ref{τ|P'}[R',G']
  - Relies summarize guarantees (#4):
    G' ⊆ R, G ⊆ R'
  - Ensured by splitting semantics and folding
- Actions in x.G are included in alias y's y.R, and thus by stability preserves y.P

Outline

- Concurrent Reasoning for Aliases
- Typechecking Rely-Guarantee References
- Technical Challenge: Nested References
- Intuition for Soundness
- Conclusions

Future Work

- We've worked out a core system
- Route to a full system:
  - Non-atomic updates
  - Internalizing proof terms (in progress)
  - Better datatype definitions
  - Borrowing
  - Concurrency (in progress)
  - More examples / experience

Related Work

- Rely-guarantee program logics
  - Mostly concurrent
  - Explicit Stabilization (Wickerson'10) used for malloc
  - We apply RG at a much finer granularity
- Reference Immutability, Ownership
  - Notion of per-reference read-only
  - Tschantz'05, Zibin'07, Dietl'07, Zibin'10, Gordon'12
  - We generalize read-only to arbitrary relations
- Dependent types for imperative programs
  - Types depend only on immutable data (DML, etc.)
  - Or bake in a low-level program logic (Ynot / Hoare Type Theory)
  - Our types directly treat interference

Conclusion

- Rely-Guarantee References
  - Directly address alias interference
  - Key challenge: nested references
  - Apply concurrent verification insights to aliasing
    - We applied rely-guarantee
    - Other correspondences exist
- Promising early results
  - Modest proof burden
  - Backwards compatible with more traditional systems

https://github.com/csgordon/rgref/