Introduction to Separation Logic

Rafal Kolanski

October 2008

Australian Government
Department of Broadband, Communications and the Digital Economy
Australian Research Council

NICTA Members

NICTA Partners

Overview

NICTA

In this talk:

- background on pointers
- separation logic

Why Reason about Pointers?

Pointers are everywhere!

- operating system kernels (Linux)
- device drivers
- network code (TCP/IP)
- web servers (Apache)
- anything involving C/C++
- even Java and ML have references

The Problem with Pointers

$$\{\ \text{valid}\ p \wedge \text{valid}\ q\ \}$$

$$*q = 42;$$

$$*p = 7;$$

$$\{\ *p = 7 \wedge *q = ?\ \}$$

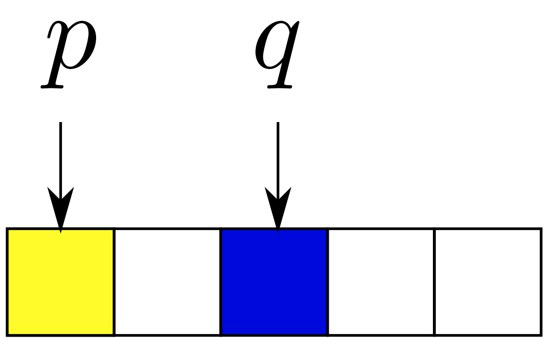
The Problem with Pointers

$$\{ \text{ valid } p \wedge \text{valid } q \text{ } \}$$

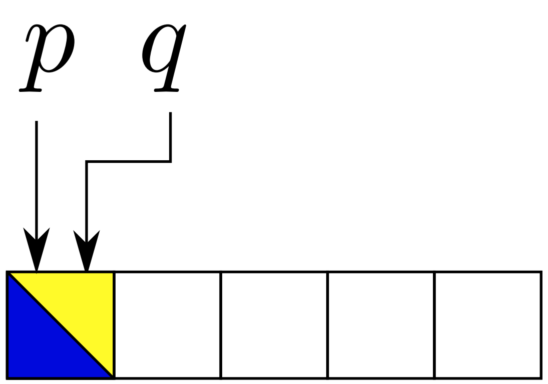
$$*q = 42;$$

$$*p = 7;$$

$$\{ *p = 7 \wedge *q = ? \}$$



$$\Rightarrow *q = 42$$



$$\Rightarrow *q = 7$$

Some Simpler Approaches

A Simple Plan

$$\textbf{datatype}\ \text{ref} = \text{Ref int} \mid \text{Null}$$

$$\textbf{types}\ \text{heap} = \text{int} \Rightarrow \text{val}$$

$$\textbf{datatype}\ \text{val} = \text{Int int} \mid \text{Bool bool} \mid \text{Struct\_x int int bool} \mid \ldots$$

- hp :: heap, p :: ref
- pointer access: *p = the_Int (hp (the_addr p))
- pointer update: *p :== v   = hp :== hp ((the_addr p) := v)

- a bit klunky
- gets worse with structs
- lots of value extraction (the_Int) in spec and program

Burstall '72 / Bornat '00

A linked list struct with next pointer and element:

$$\textbf{datatype}\ \mathrm{ref} = \mathrm{Ref\ int} \mid \mathrm{Null}$$

$$\textbf{types}\ \mathrm{next\_hp} = \mathrm{int} \Rightarrow \mathrm{ref}$$

$$\textbf{types}\ \mathrm{elem\_hp} = \mathrm{int} \Rightarrow \mathrm{int}$$

- next :: next_hp, elem :: elem_hp, p :: ref
- pointer access: p->next = next (the_addr p)
- pointer update: p->next :== v = next :== next ((the_addr p) := v)

- a separate heap for each struct field
- p->next and p->elem can't alias
- assumes a type-safe language
- p1->next and p2->next can still alias

Separation Logic

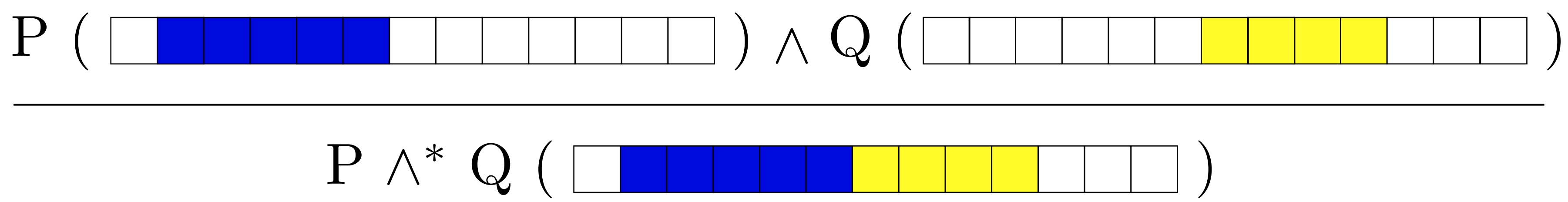
The Heap

$$\textbf{types}\ \text{heap} = \text{“nat} \rightharpoonup \text{nat”}$$

The heap represents computer memory

- partial map: allocated and unallocated regions
- emp: a heap with no allocated regions
- we'll use a simple version based on natural numbers
- and steal 0 to mean the null pointer

Separating Conjunction



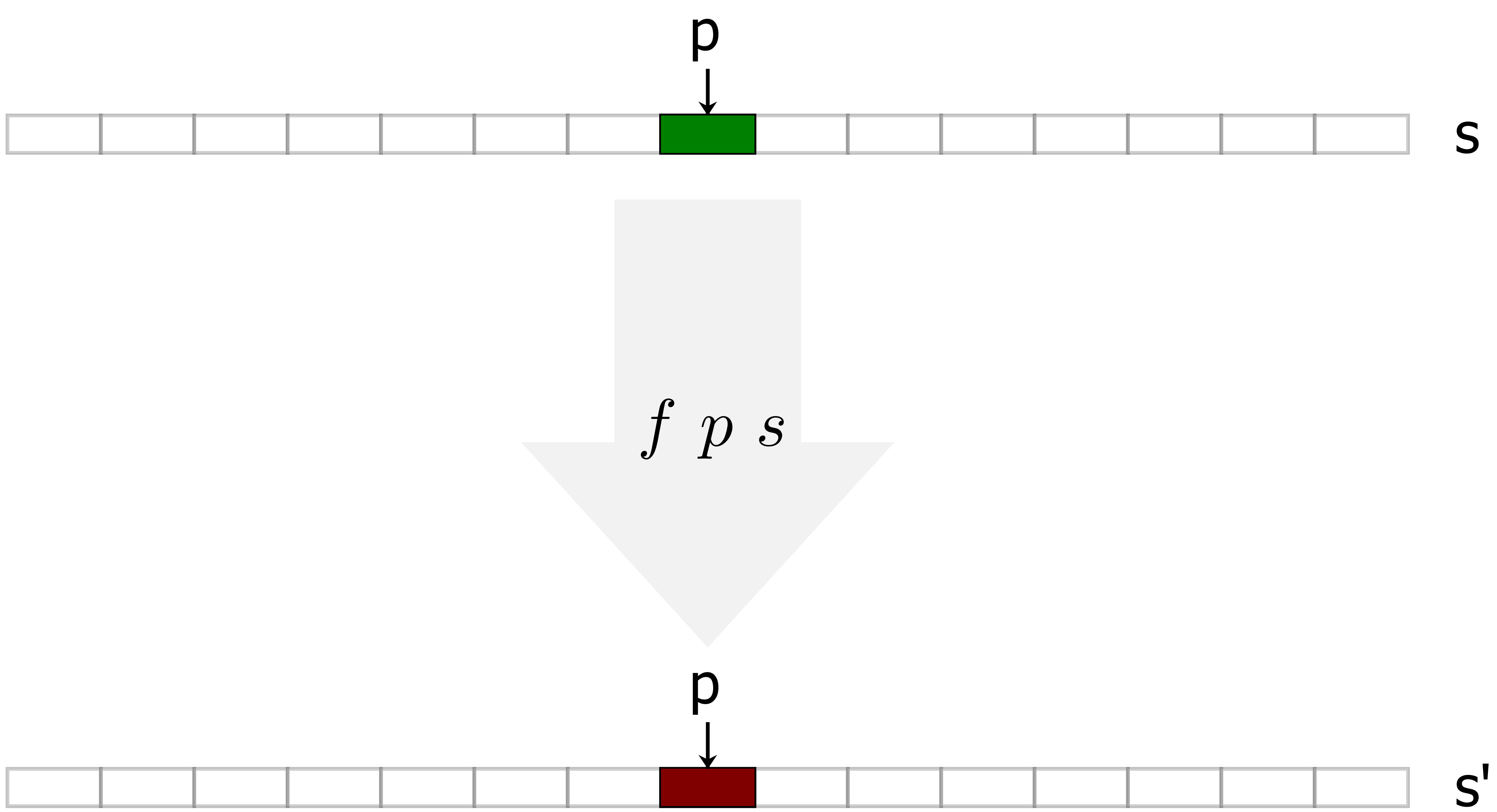
Primary mechanism of separation logic

- assign resources (e.g. heap) to predicates
- predicates consume resources
- no resource sharing across separating conjunction

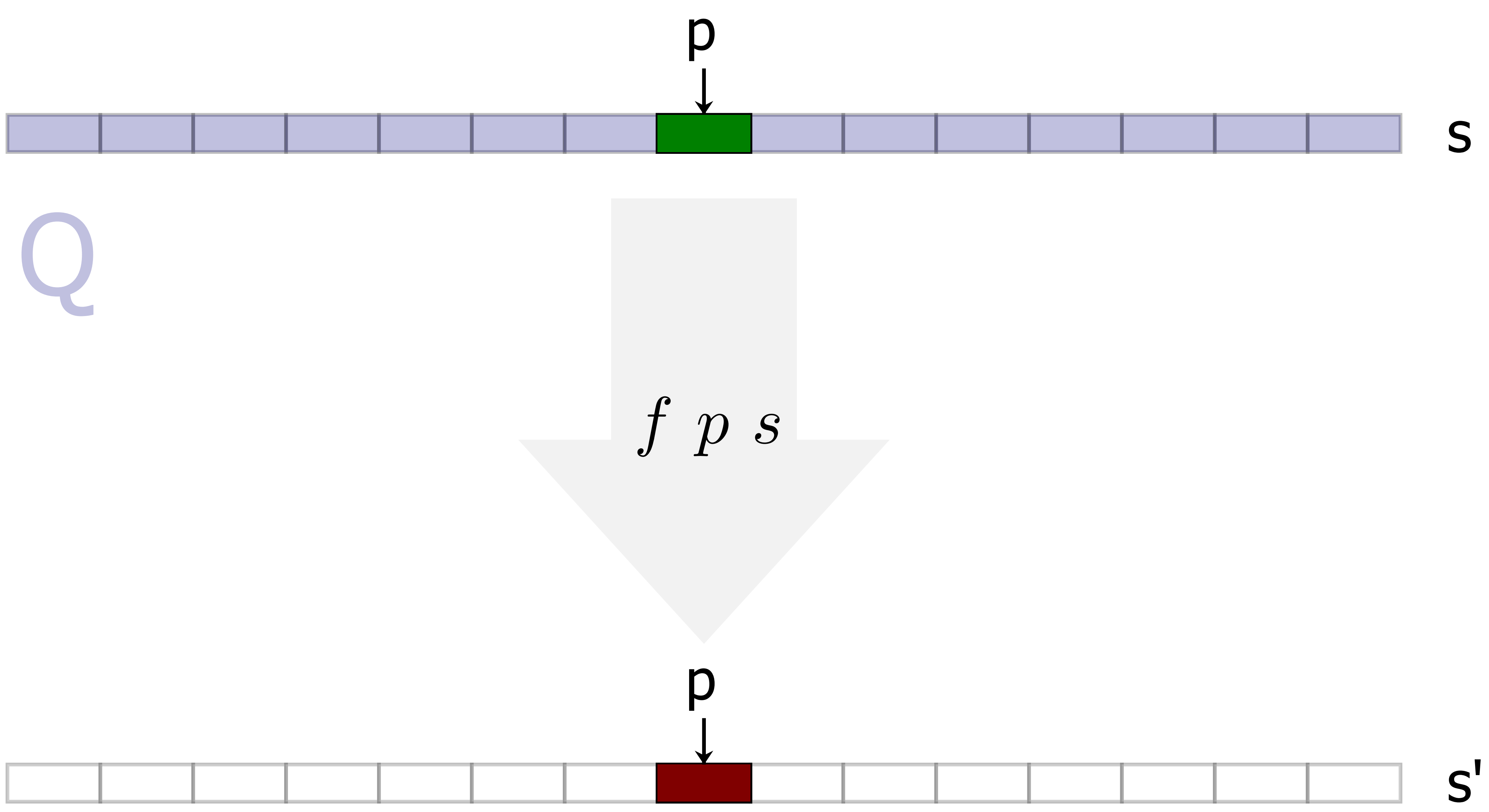
$$h_0 \perp h_1 \equiv \mathrm{dom}\ h_0 \cap \mathrm{dom}\ h_1 = \{\}$$

$$(\mathrm{P} \wedge^* \mathrm{Q})\ \mathrm{h} \equiv \exists h_0\ h_1.\ h = h_0 \mathbin{++} h_1 \wedge h_0 \perp h_1 \wedge \mathrm{P}\ h_0 \wedge \mathrm{Q}\ h_1$$

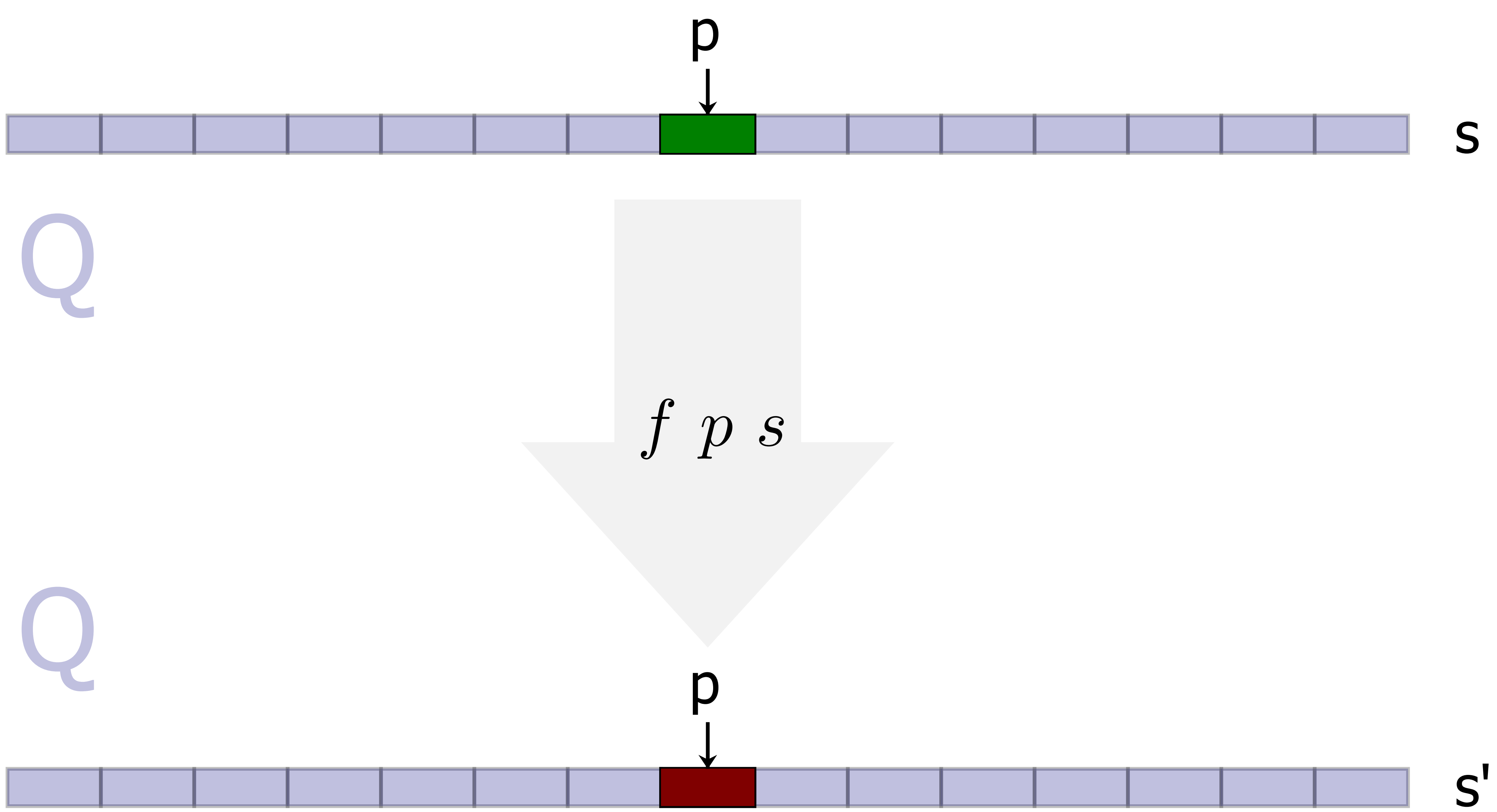
Benefits of Local Reasoning



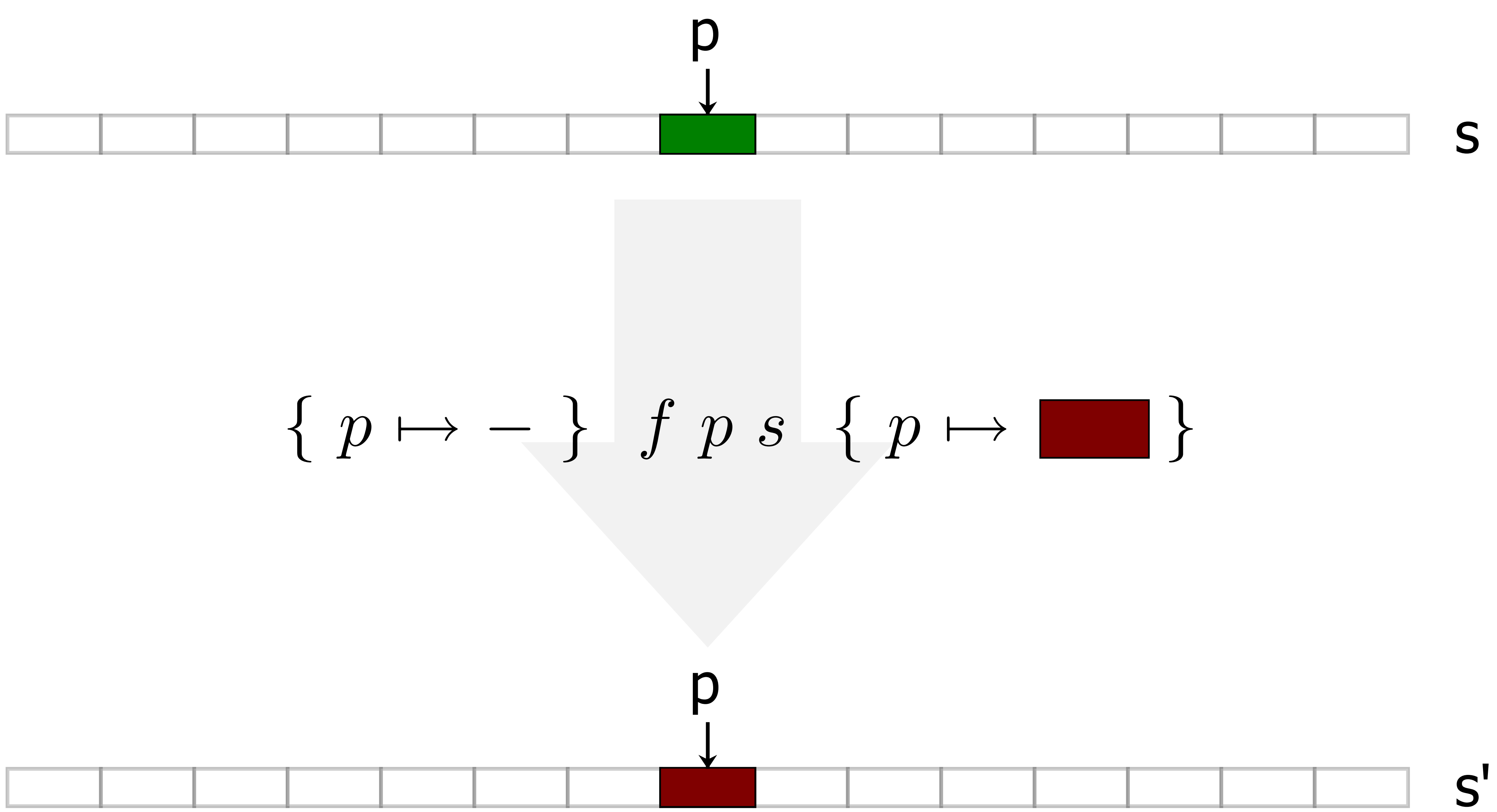
Benefits of Local Reasoning



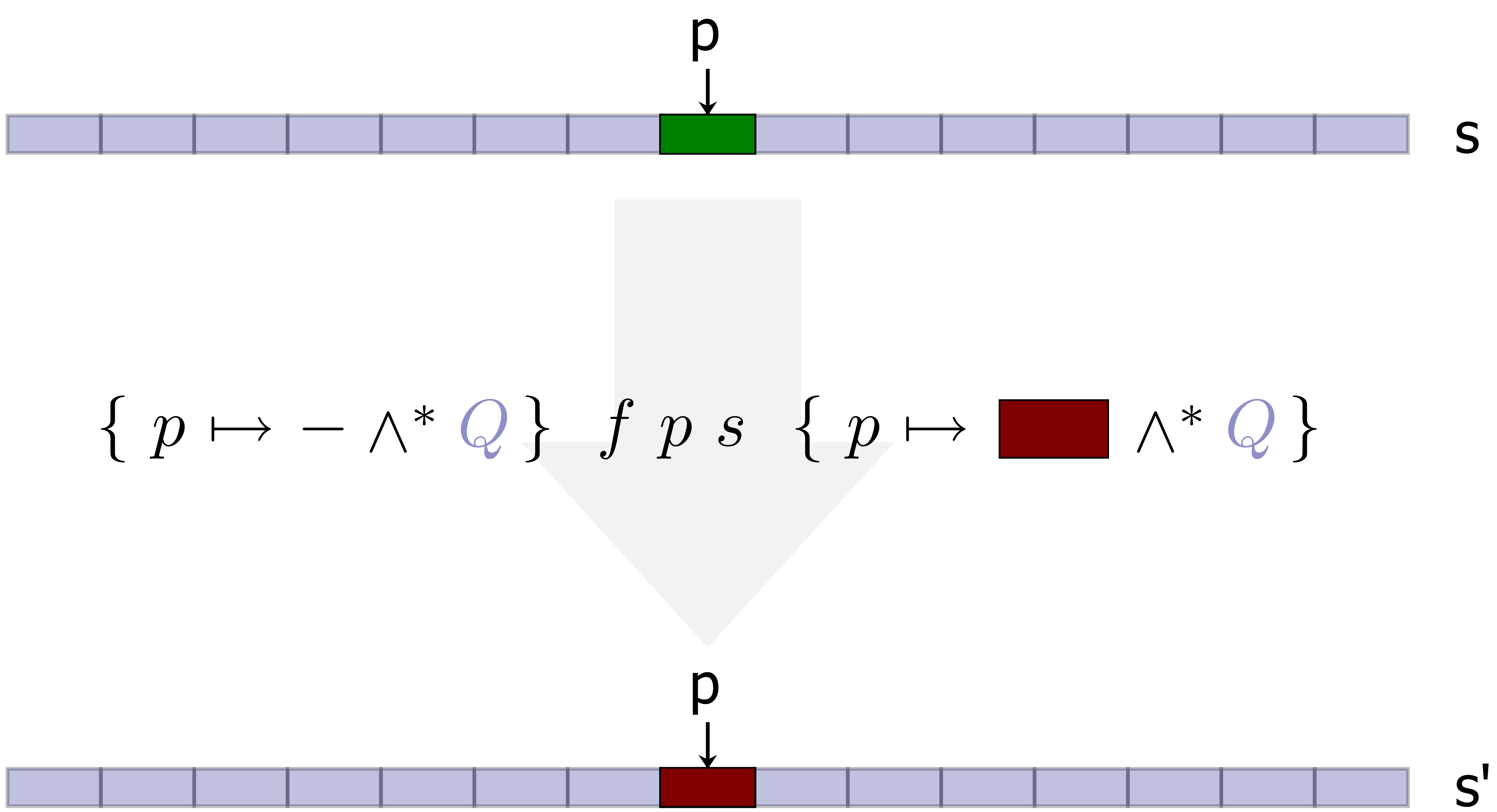
Benefits of Local Reasoning



Benefits of Local Reasoning



Benefits of Local Reasoning



$$\{\, p \mapsto - \wedge^* Q \,\}\ \ f\ p\ s\ \ \{\, p \mapsto \blacksquare \wedge^* Q \,\}$$

Benefits of Local Reasoning

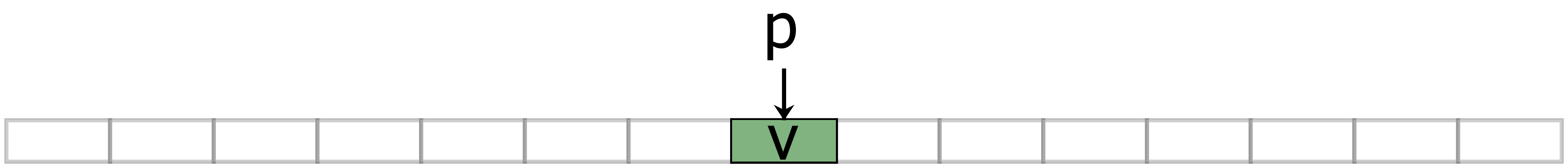
$$\frac{\{\ p \mapsto -\ \}\ \ f\ p\ s\ \ \{\ p \mapsto \blacksquare\ \}}{\{\ p \mapsto -\ \wedge^*\ Q\ \}\ \ f\ p\ s\ \ \{\ p \mapsto \blacksquare\ \wedge^*\ Q\ \}}$$

Benefits of Local Reasoning

The Frame Rule

$$\frac{\{\ P\ \}\ \ stmt\ \ \{\ P'\ \}}{\{\ P \wedge^{*} Q\ \}\ \ stmt\ \ \{\ P' \wedge^{*} Q\ \}}$$

Precise Mapping Predicates



$$(p \mapsto v)\ h \equiv (h\ p = \mathrm{Some}\ v \wedge \mathrm{dom}\ h = \{p\})$$

$$(p \mapsto -) \equiv \exists v.\ (p \mapsto v)$$

The maps-to predicate defines a heap

- with only one valid pointer
- combine with other mappings to make bigger heaps
- remember to use separating conjunction!

Defining a Linked List

Demo

A Programming Model

What's old:

- local variables used for calculations
- the usual constructs: SKIP, IF, WHILE, ";"
- and local variable assignment
- with identical Hoare rules

What's new:

- a variable representing the heap
- want precise specification of assignment to pointer
- need a way to allocate/free memory

Allocation and Disposal

Allocation rule:

$$\{\ \text{emp}\ \}\ \text{alloc}\ x\ [e_1, e_2, \ldots, e_n]\ \{\ x \mapsto e_1 \wedge^* \ldots \wedge^*\ x + n \mapsto e_n\ \}$$

Disposal rule:

$$\{\ x \mapsto -\ \}\ \text{dispose}\ x\ \{\ \text{emp}\ \}$$

Assignment

The normal, local assignment rule:

$$\{\ x \mapsto -\ \}\ [x] := v\ \{\ x \mapsto v\ \}$$

Using the magic wand (separating implication):

$$(\mathrm{P} \longrightarrow^* \mathrm{Q})\ \mathrm{h} \equiv \forall h'\ .\ h' \perp h \wedge \mathrm{P}\ h' \longrightarrow \mathrm{Q}\ (h ++ h')$$

we can make it a backwards-reasoning rule:

$$\{\ x \mapsto - \wedge^* (x \mapsto v \longrightarrow^* \mathrm{P})\ \}\ [x] := v\ \{\ \mathrm{P}\ \}$$

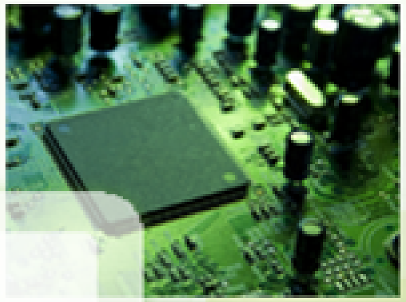
Reversing a Linked List

Demo

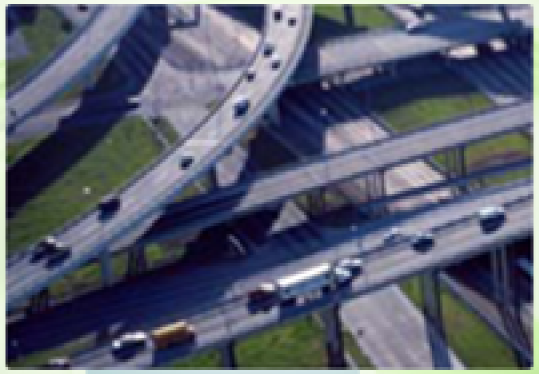
Conclusion

Separation Logic

- is a nice way to reason about pointers
- doesn't need specification of what doesn't change



NICTA



From **imagination** to **impact**