Meanings of Annotations in VCC

Deepak D'Souza

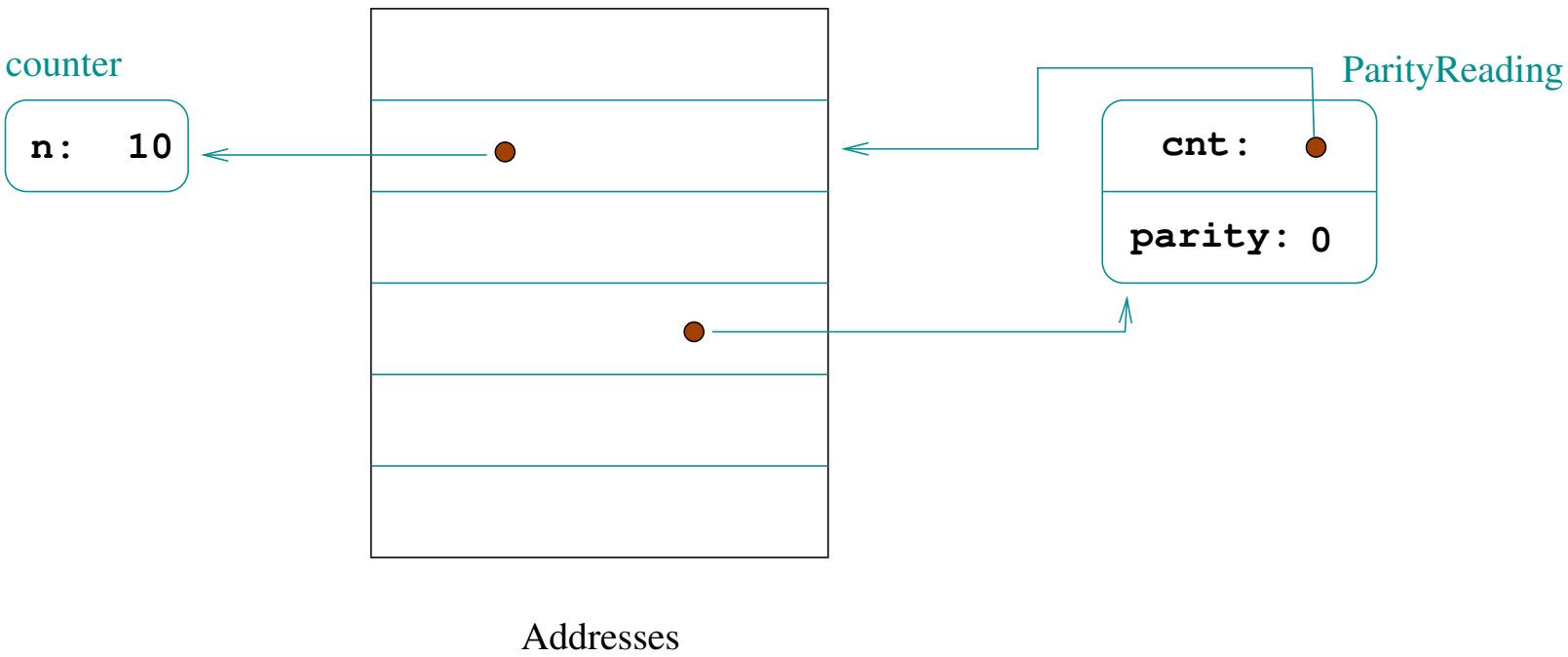
Department of Computer Science and Automation
Indian Institute of Science, Bangalore.

2 Mar 2020

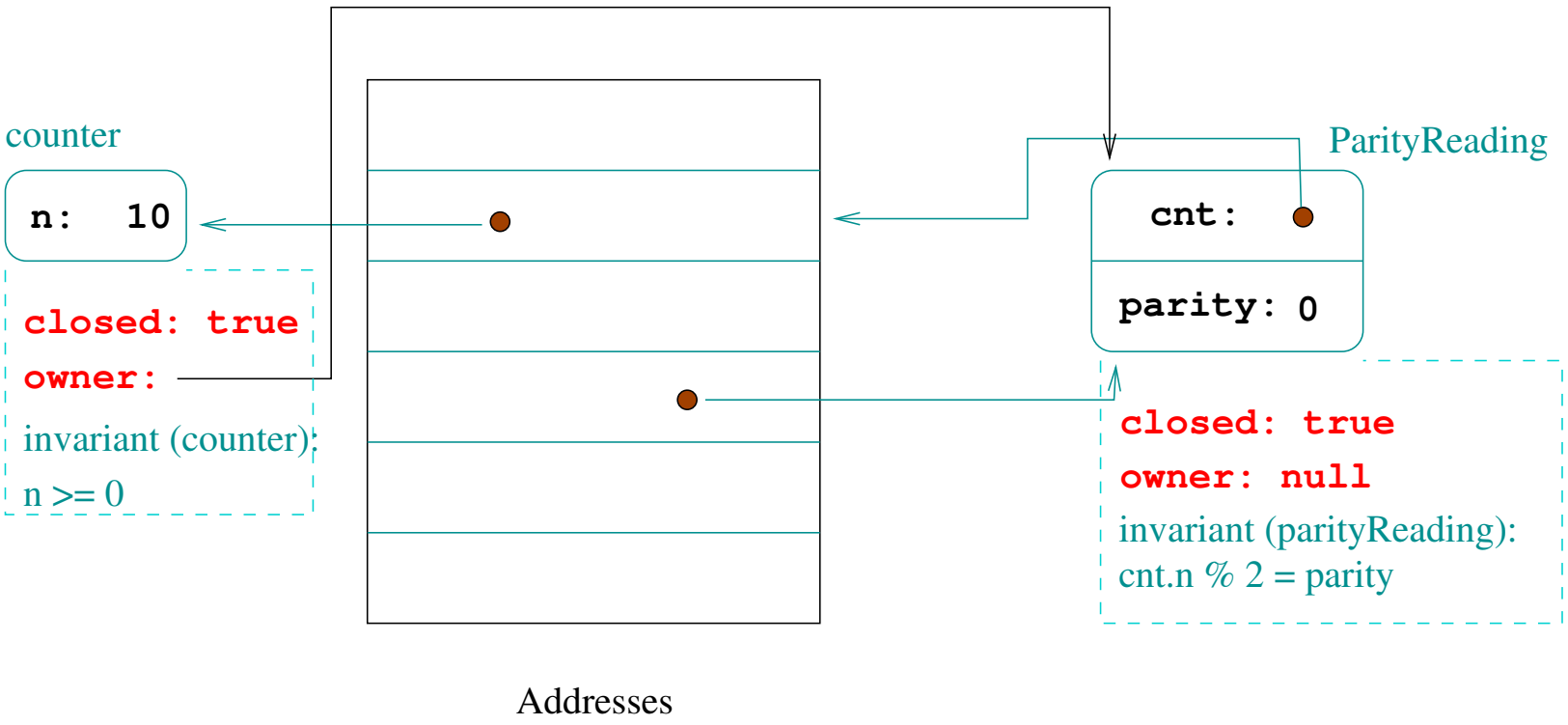
Outline

Heap state

- Heap state (set of objects at certain addresses in memory).



Augmented Heap state



VCC also adds other auxiliary ("ghost") fields like "**owns**" (the set of objects declared to be owned by this object, typically in its invariants); and a boolean "**valid**" (true if the object is part of the set of "real" objects in the current state).

Overview of "local" verification

- Aim is to prove that a program is safe (object invariants are satisfied in each reachable state).
- Standard inductive argument: show initial state is safe, and show that each program statement leads from a safe pre-state to a safe post-state.
- Arguing this may not be easy due to object invariants spanning several sub-objects.
- Instead, first show that object invariants are admissible (an invariant-preserving update to a sub-object does not break the invariant of the super-object); and then argue "locally" that each statement is legal (updates to an object preserves its invariant).
- In other words:

$$\textit{admissible invariants} + \textit{legal updates} = \textit{safety}$$

Object (ghost) fields added by VCC

- **valid**: Boolean field that is true iff the object is part of the set of "real" objects in the current heap state. In particular, p cannot be NULL.
- **closed**: Boolean field which is true iff this object is "closed" (is valid and satisfies its invariants).
- **owner**: pointer to the object (or thread) that currently owns this object.
- **owns**: set of objects declared to be owned by this object, typically in its invariants

Meaning of annotations (predicates)

- **thread_local(p)**: true iff (the object pointed to by) p is valid and owned by current thread.
- **mutable(p)**: true iff **thread_local**(p) and is not closed.
- **wrapped(p)**: true iff (the object pointed to by) p is closed and owned by the current thread.
- **writeable(p)**: true iff (the object pointed to by) p is part of the \writes set of the function. (i.e. the function has permission to write to p).

Meaning of annotations (declarations)

- **\writes p**: p is part of the \writes set of the function (i.e. the objects that the function might possibly modify). Also requires that $p->\mathit{owner}$ is me.

Meaning of annotations (expressions)

- **span(p)**: the set of pointers to the members of the object pointed to by p, unioned with $\{p\}$ itself.
- **embedding(o)**: if o is a pointer to a primitive field (like `int`) of an object p, returns p.

Meaning of annotations (statements)

- _**(unwrap p)**: Add all member objects to **writes** set of the function. In particular, transfers ownership of sub-objects from p to the current thread.

  Important: the function does not have to (and should not) report that it **writes span(p)**, as this would lead to contradictory assumptions about ownership of p's sub-objects. Instead, the function should just say it **writes p**.

Example illustrating writes p and writes span(p)

```
typedef struct counter {
  _(ghost \natural count),
  unsigned hi, lo;
  ...
} counter;
```

```
void init(counter *p)
_(writes \span(p))
_(ensures \wrapped(p))
{
  _(ghost p->count = 0);
  p->hi = 0;
  p->lo = 0;
  _(wrap p)
}
```

```
void inc(counter *p)
_(writes p)
_(requires \wrapped(p))
_(requires p->count < 256*255 + 255)
_(ensures \wrapped(p))
{
  _(unwrap p)
  _(ghost p->count = p->count + 1)
  if (p->lo < 255)
    p->lo++;
  else {
    p->lo = 0;
    p->hi++;
  }
  _(wrap p)
}
```