Deputy (Dependent Types)
Key Idea: Verify bounds checking code that programmer already wrote. Try to use the already available code from program.

```c
void mycopy ( int *dest, int *src, int n)
// Here ‘int n’ is the bounds information.
{
    int i;
    for (i=0; i<n; i++)
    {
        dest[i] = src[i];
    }
}
```

```c
void mycopy ( int *count (n) dest, int *count (n) src, int n)
{
    int i;
    for (i=0; i<n; i++)
    {
        dest[i] = src[i];
    }
}
```

A valid example:

```c
void foo (void)
{
    int A[11], B[7];
    mycopy (A, B, 6);  // valid
}
```
Deputy treats \([\text{int *} \text{COUNT} (n)]\) as one type.

Array

\[
A : \text{ptr}(11) \text{ int} \\
\text{dest} : \text{ptr}(n) \text{ int} \quad // n \text{ is extra variable}
\]

\text{mycopy} (A, B, 6) // what compiler checks..???

\[
\begin{align*}
A : \text{ptr}(11) \text{ int} \\
& A : \text{ptr}(6) \text{ int}
\end{align*}
\]

Rule :

\[
\begin{align*}
\text{p} : \text{ptr}(n) \text{ int}, & \quad n \geq m \\
\text{P} : \text{ptr}(m) \text{ int}
\end{align*}
\]

Here, compiler will outsource the problem to theorem, which will check “11\(\geq\)6” and return true.

**Contract Programming**

It is an approach for designing software. It prescribes that software designers should define formal, precise and verifiable interface specifications for software components, which extend the ordinary definition of abstract data types with preconditions, post conditions and invariants. These specifications are referred to as "contracts".

**Factors to be considered:**

1. **Performance:**
   
   Compiler is able to check bounds at compile time, without inserting extra code, resulting in same binary. So, zero runtime overhead.
   
   **Example:** if \(n\) is input at runtime.
   
   If deputy cannot check at compile time, then it inserts runtime checks. This are very few.

2. **Binary Compatibility:**
   
   Deputy do not change interface of function but may insert some code in function body.
   
   - Without using any metadata.
   - Code will run same even if those annotations are removed.
3. No Code Change:
If there is no annotation, deputy assumes COUNT(1). (which may not be the intention.)
This is a setback and requires code change.
Code changes happen but they are good because, you don’t introduce bugs and binaries remain same.

4. Separate Compilation:
Supported.

Note: With proper implementation of Deputy, there will be no spatial bugs, but there will be temporal bugs.

```plaintext
src : ptr(n) int       i : int       0<=i<n

src[i] : int

src : ptr(n) int       // got from program
i : int                // got from program
0<=i<n                 // ????

If compiler cannot proce 0<=i<n, then it will introduce bounds check.

void mycopy ( int *count (n) dest, int *count (n) src, int n)
{
    int i;
    for (i=0; i<n; i++)
    {
        assert (0<=i && i<n);
        dest[i] = src[i];
    }
}
```
Suppose,

```
p = &i;       // Line 1

for (i=a; i<b; i+=c)
{
  --   // ------------------------------------- {a<i<b}   FACT-01
  *
p+=7;
}
```

Here, Line 1 is basically modifying i. This is called **aliasing problem**.

Fact-01 requires two conditions:

- Check that i is not modified in the loop.
- “&I” is never used in the program.

So, it checks the fact \(0<=i<n\) implies “assert \((0<=i && i<n)\)”, which eliminates the need of assert statement.

**Hybrid Type Checking**: Bounds checking driven by type checking rules and what programmer does.
**Limitation:**

If i is part of struct, then without taking address of i (&i), we can have it altered.

```c
//ignore any printf bugs here

int myprint (int * COUNT(n) p, int n)
{
    while(n>0);
    {
        printf("%d", *p);
        p++;
        n--;
        // parallel assignment
    }
}

p : ptr(n) int
```

![](image)

```
p : ptr(n) int       n : int       n>0
--------------------------------------------------------
(p+1) : ptr(n-1) int
```