Contract Visualization for VeriFast: General Approach and Concepts

Pauline Hergersberg
Hochschule für Technik und Wirtschaft (HTW) Berlin
Department of Engineering I

Talk @ KeY Symposium 2023
Høgskulen på Vestlandet (HVL), Bergen, Norway
August 8 - 10, 2023

Hochschule für Technik und Wirtschaft Berlin
University of Applied Sciences

Outline

Problem and Objective

Problem and Objective

Problem

- contracts are bound to textual format
- textual format is detailed but not neccessarily clear
- especially beginners to VeriFast struggle with the complexity of contracts

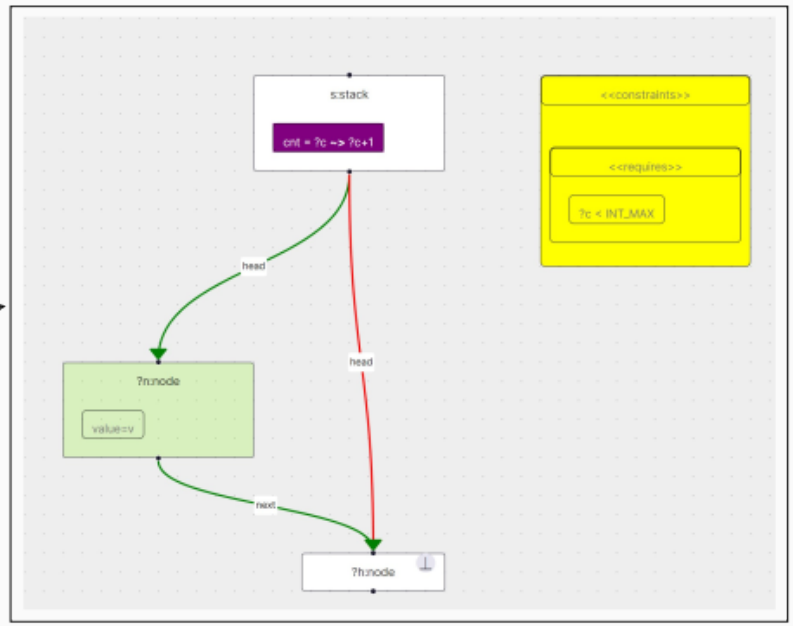
```
/*@ requires s->head |-> ?h &*&
             s->cnt |-> ?c &*& c < INT_MAX;
@*/
/*@ ensures  s->head |-> ?n &*& s->cnt |-> c+1 &*&
             n->value |-> v &*& n->next |-> h &*&
             malloc_block_node(n);
@*/
```

Problem and Objective

Objective

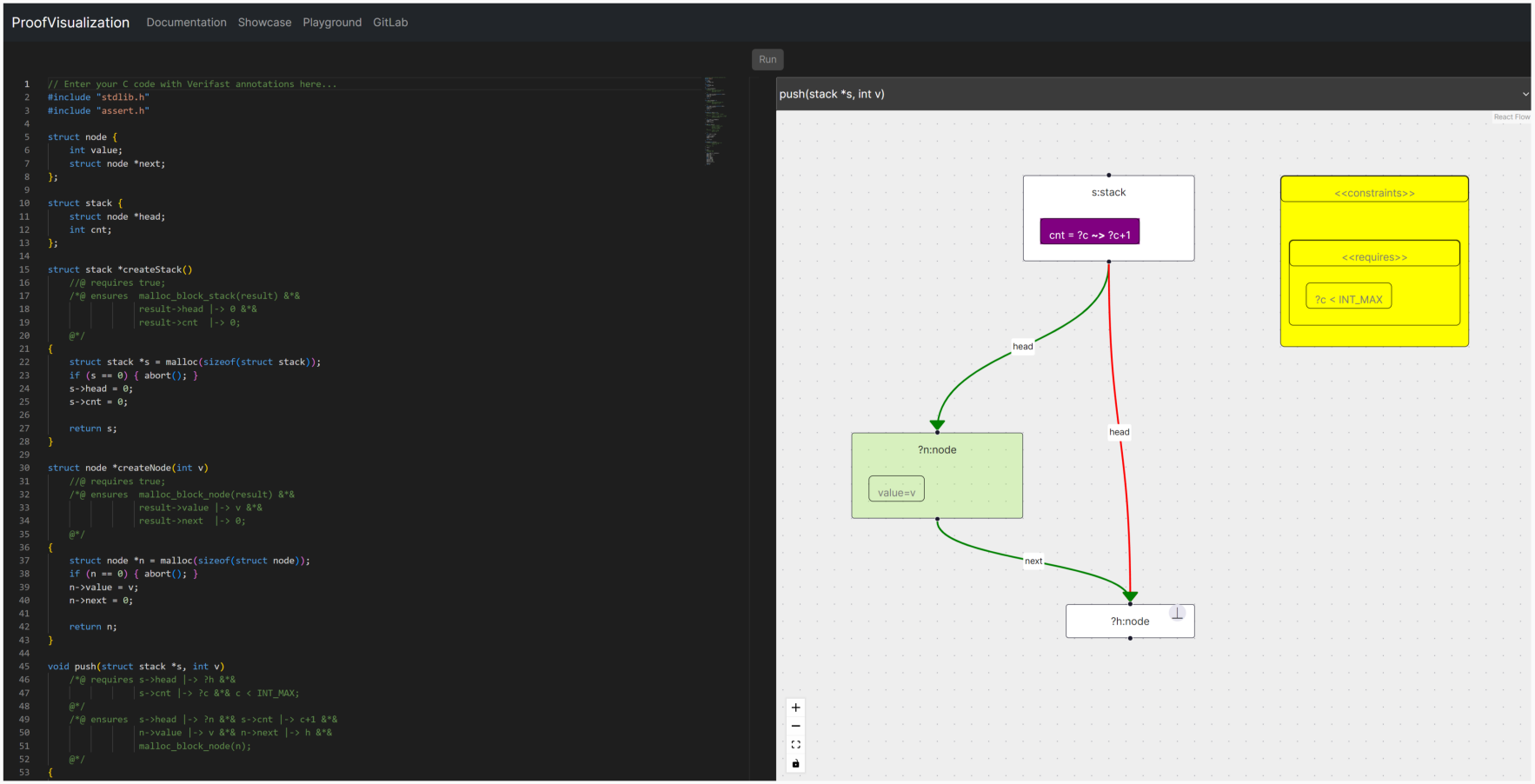
- process information from contracts
- automatically convert textual to visual format while preserving all information

```
/*@ requires s->head |-> ?h &*&
             s->cnt |-> ?c &*& c < INT_MAX;
@*/
/*@ ensures  s->head |-> ?n &*& s->cnt |-> c+1 &*&
             n->value |-> v &*& n->next |-> h &*&
             malloc_block_node(n);
@*/
```

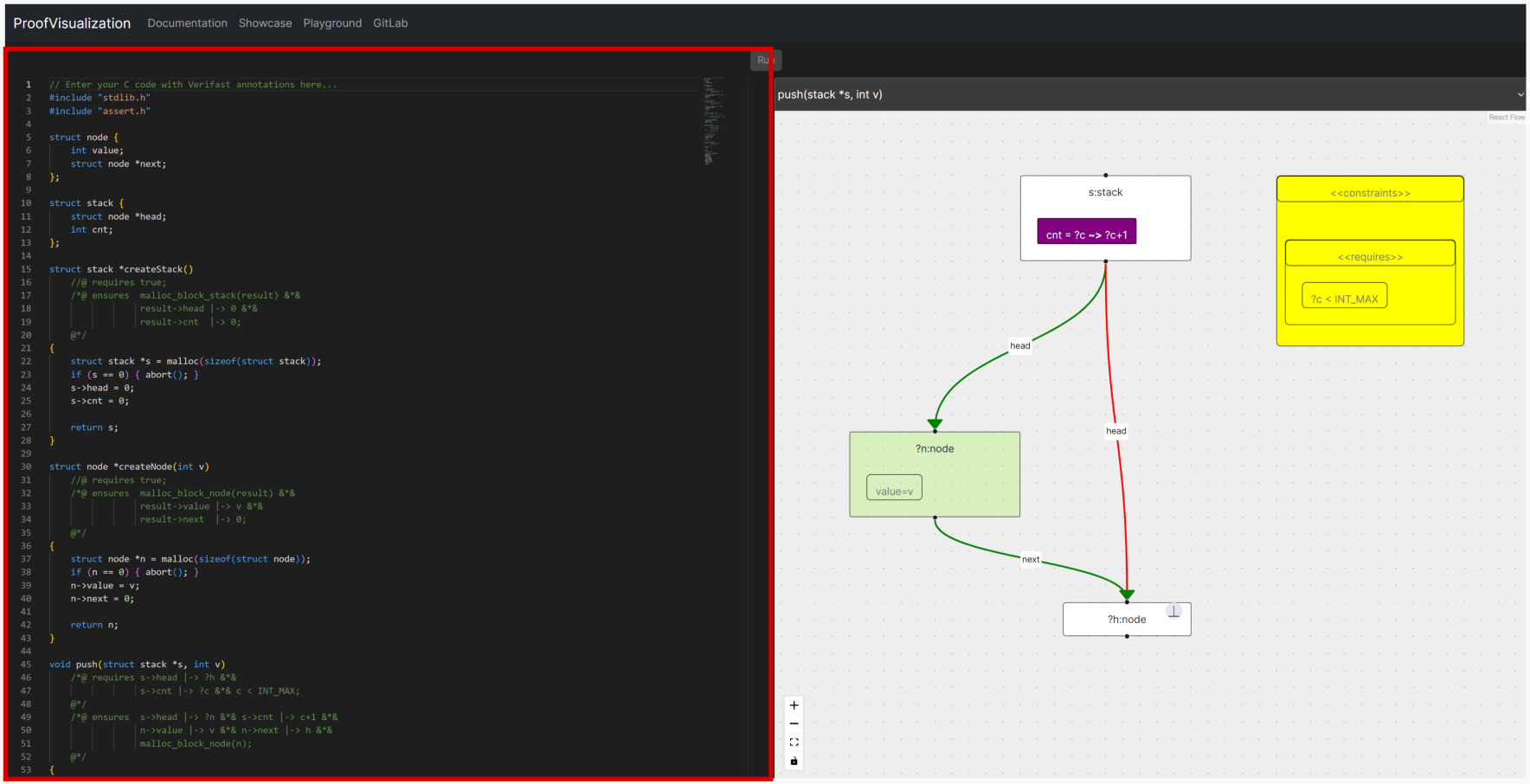


First Look

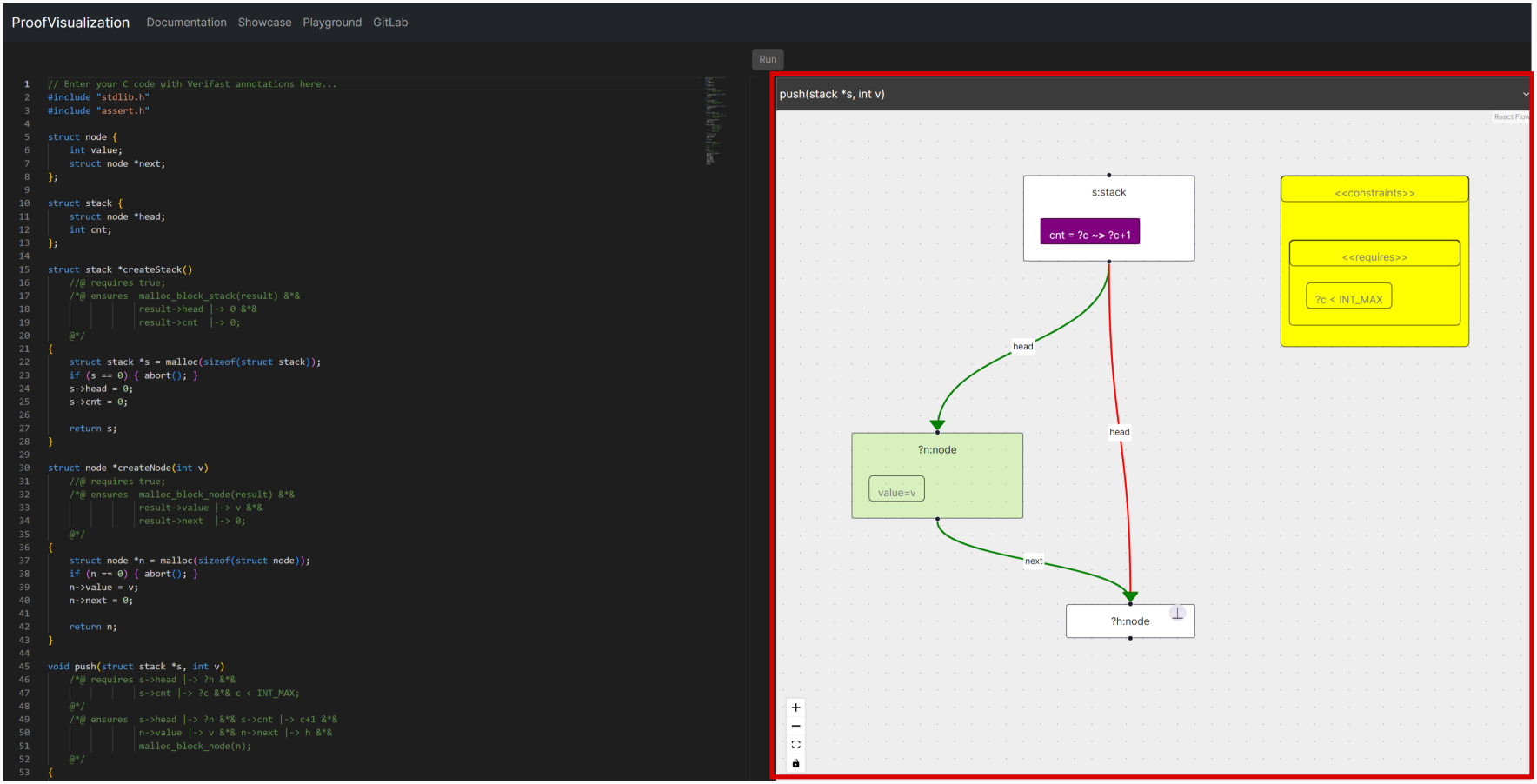
First Look


ProofVisualization
Documentation
Showcase
Playground
GitLab
Run
// Enter your C code with Verifast annotations here...
#include "stdlib.h"
#include "assert.h"
struct node {
int value;
struct node *next;
};
struct stack {
struct node *head;
int cnt;
};
struct stack *createStack()
//@ requires true;
/*@ ensures malloc_block_stack(result) &*&
result->head |-> 0 &*&
result->cnt |-> 0;
@*/
{
struct stack *s = malloc(sizeof(struct stack));
if (s == 0) { abort(); }
s->head = 0;
s->cnt = 0;
return s;
}
struct node *createNode(int v)
//@ requires true;
/*@ ensures malloc_block_node(result) &*&
result->value |-> v &*&
result->next |-> 0;
@*/
{
struct node *n = malloc(sizeof(struct node));
if (n == 0) { abort(); }
n->value = v;
n->next = 0;
return n;
}
void push(struct stack *s, int v)
/*@ requires s->head |-> ?h &*&
s->cnt |-> ?c &*& c < INT_MAX;
@*/
/*@ ensures s->head |-> ?n &*& s->cnt |-> c+1 &*&
n->value |-> v &*& n->next |-> h &*&
malloc_block_node(n);
@*/
{
push(stack *s, int v)
React Flow
s:stack
cnt = ?c ~> ?c+1
<<constraints>>
<<requires>>
?c < INT_MAX
head
head
?n:node
value=v
next
?h:node

First Look


ProofVisualization
Documentation
Showcase
Playground
GitLab
// Enter your C code with Verifast annotations here...
#include "stdlib.h"
#include "assert.h"
struct node {
int value;
struct node *next;
};
struct stack {
struct node *head;
int cnt;
};
struct stack *createStack()
//@ requires true;
/*@ ensures malloc_block_stack(result) &*&
result->head |-> 0 &*&
result->cnt |-> 0;
@*/
{
struct stack *s = malloc(sizeof(struct stack));
if (s == 0) { abort(); }
s->head = 0;
s->cnt = 0;
return s;
}
struct node *createNode(int v)
//@ requires true;
/*@ ensures malloc_block_node(result) &*&
result->value |-> v &*&
result->next |-> 0;
@*/
{
struct node *n = malloc(sizeof(struct node));
if (n == 0) { abort(); }
n->value = v;
n->next = 0;
return n;
}
void push(struct stack *s, int v)
/*@ requires s->head |-> ?h &*&
s->cnt |-> ?c &*& c < INT_MAX;
@*/
/*@ ensures s->head |-> ?n &*& s->cnt |-> c+1 &*&
n->value |-> v &*& n->next |-> h &*&
malloc_block_node(n);
@*/
{
push(stack *s, int v)
React Flow
s:stack
cnt = ?c ~> ?c+1
<<constraints>>
<<requires>>
?c < INT_MAX
head
head
?n:node
value=v
next
?h:node

First Look

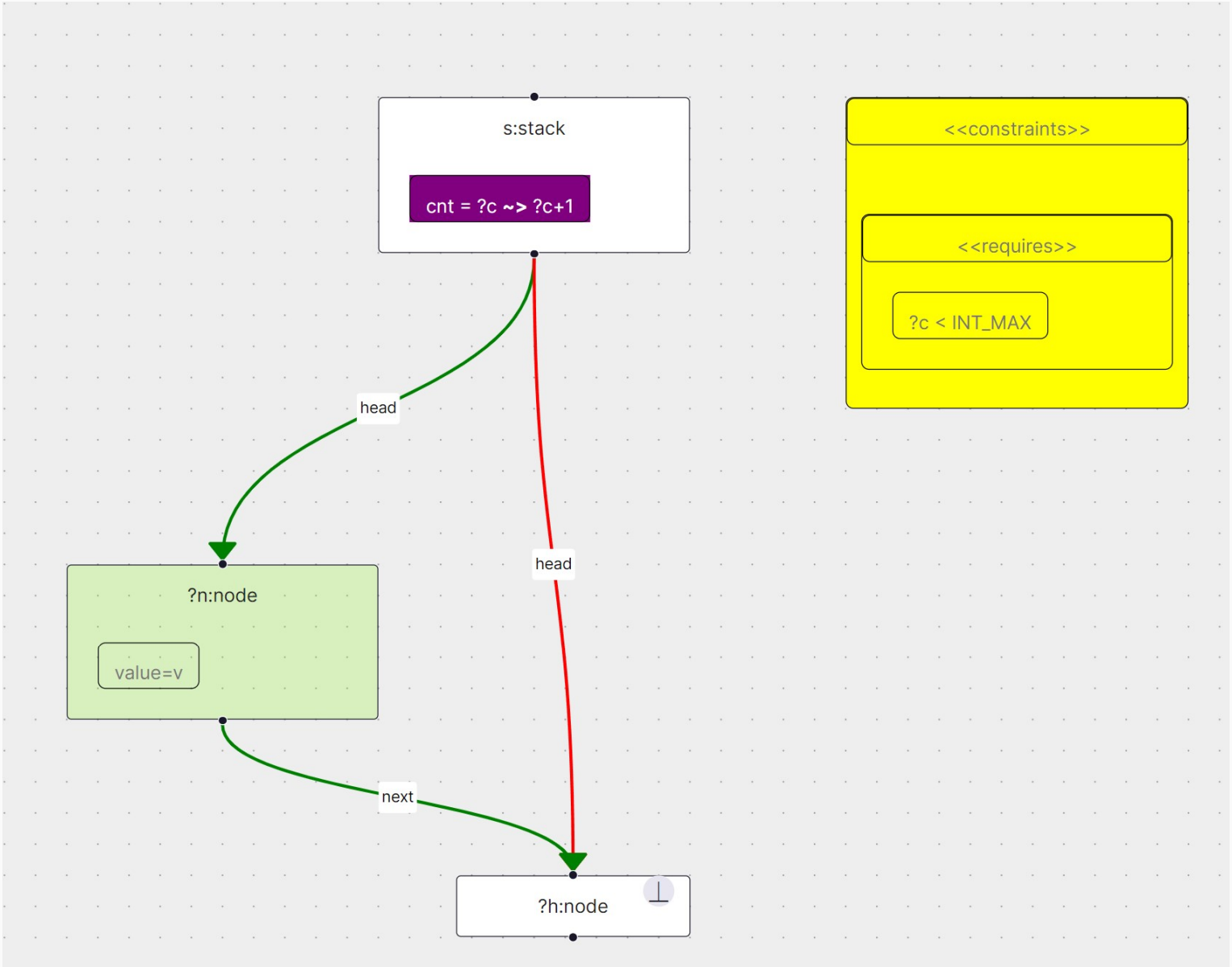

ProofVisualization
Documentation
Showcase
Playground
GitLab
Run
// Enter your C code with Verifast annotations here...
#include "stdlib.h"
#include "assert.h"
struct node {
int value;
struct node *next;
};
struct stack {
struct node *head;
int cnt;
};
struct stack *createStack()
//@ requires true;
/*@ ensures malloc_block_stack(result) &*&
result->head |-> 0 &*&
result->cnt |-> 0;
@*/
{
struct stack *s = malloc(sizeof(struct stack));
if (s == 0) { abort(); }
s->head = 0;
s->cnt = 0;
return s;
}
struct node *createNode(int v)
//@ requires true;
/*@ ensures malloc_block_node(result) &*&
result->value |-> v &*&
result->next |-> 0;
@*/
{
struct node *n = malloc(sizeof(struct node));
if (n == 0) { abort(); }
n->value = v;
n->next = 0;
return n;
}
void push(struct stack *s, int v)
/*@ requires s->head |-> ?h &*&
s->cnt |-> ?c &*& c < INT_MAX;
@*/
/*@ ensures s->head |-> ?n &*& s->cnt |-> c+1 &*&
n->value |-> v &*& n->next |-> h &*&
malloc_block_node(n);
@*/
{
push(stack *s, int v)
React Flow
s:stack
cnt = ?c ~> ?c+1
<<constraints>>
<<requires>>
?c < INT_MAX
head
head
?n:node
value=v
next
?h:node

push(struct stack *s, int v) Visualization

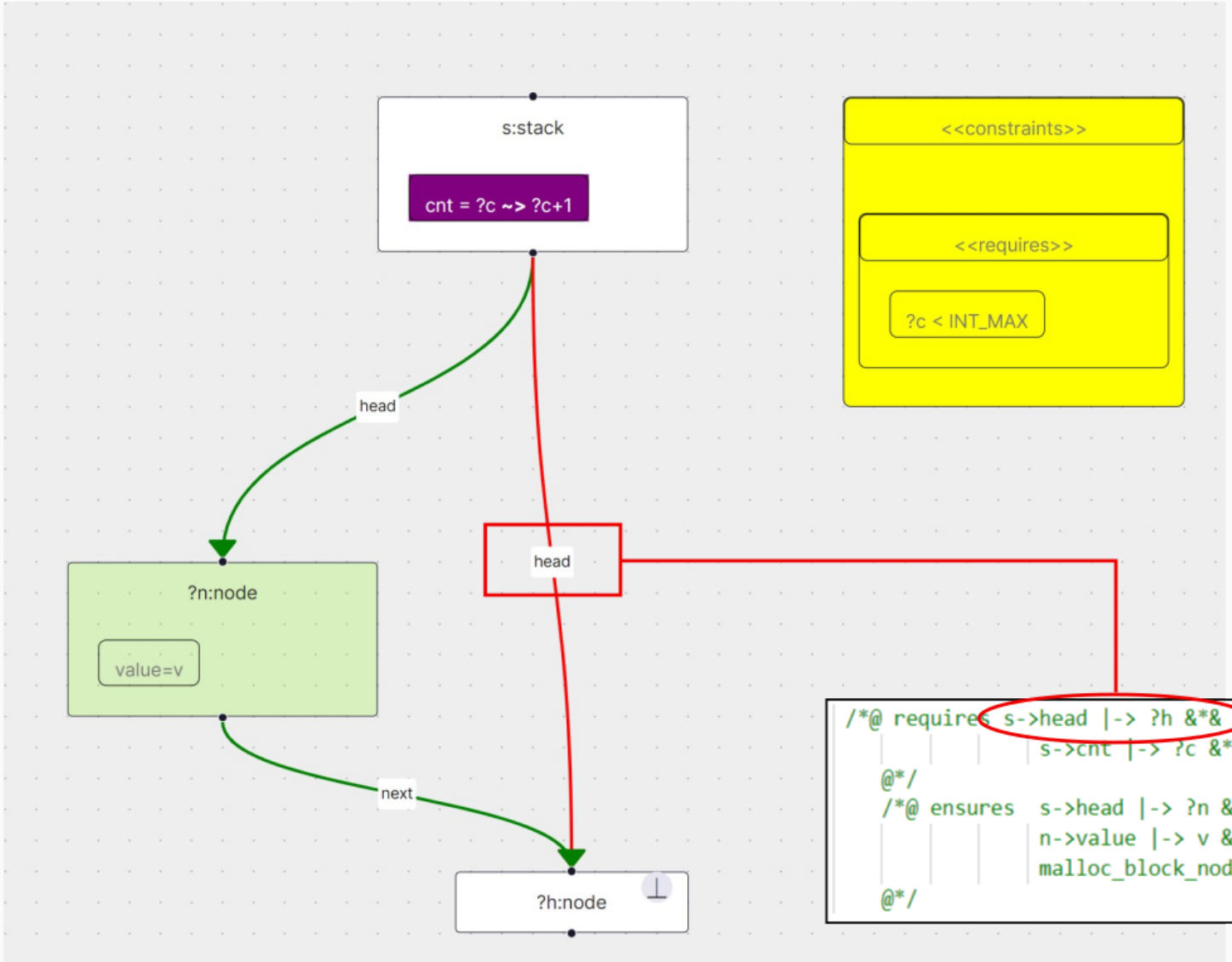
push(struct stack *s, int v) Visualization

```
void push(struct stack *s, int v)
    /*@ requires s->head |-> ?h &*&
                 s->cnt |-> ?c &*& c < INT_MAX;
    @*/
    /*@ ensures  s->head |-> ?n &*& s->cnt |-> c+1 &*&
                 n->value |-> v &*& n->next |-> h &*&
                 malloc_block_node(n);
    @*/
{
    struct node *n = createNode(v);
    n->next = s->head;
    s->head = n;
    s->cnt = s->cnt+1;
}
```

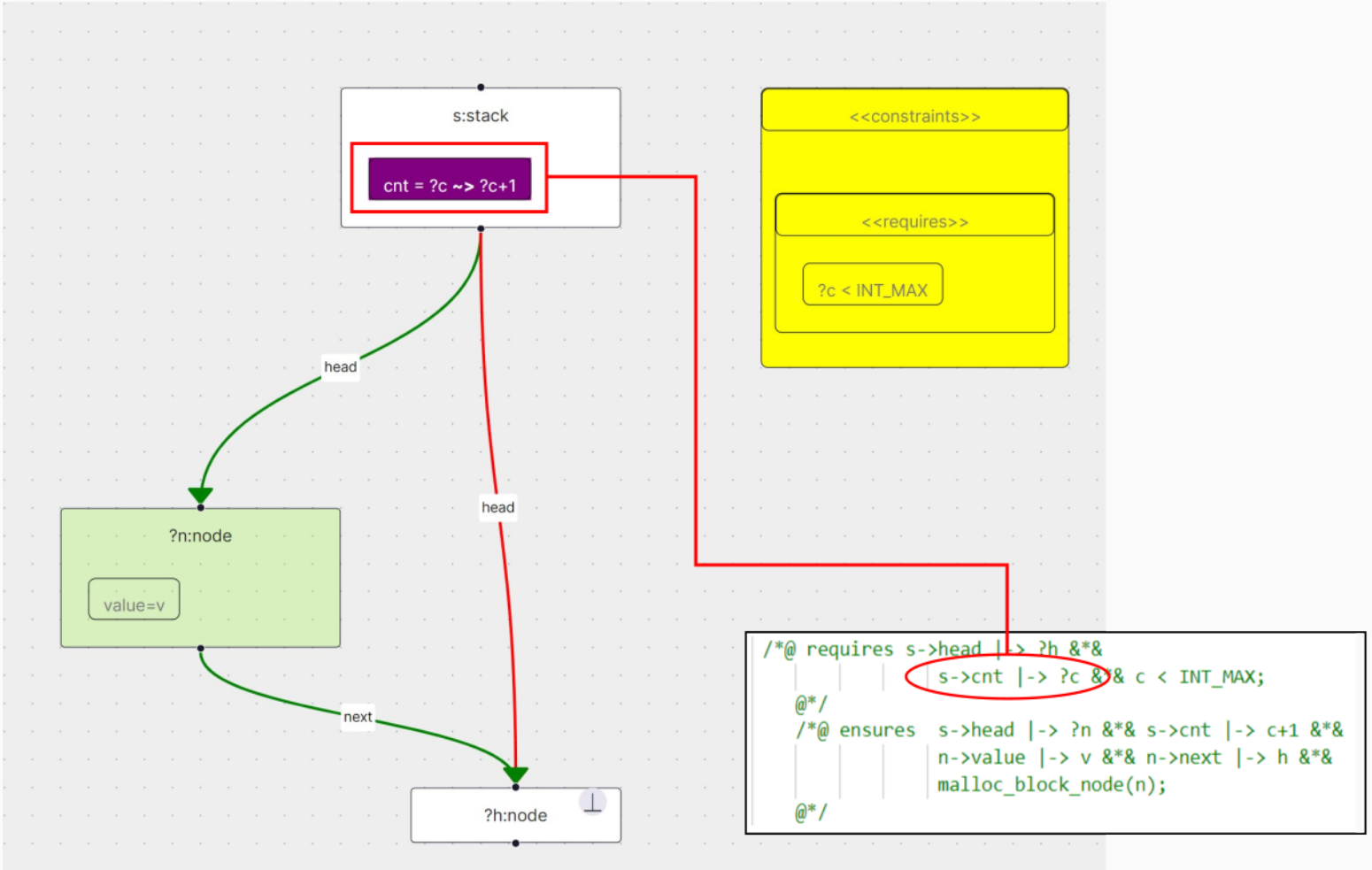
push(struct stack *s, int v) visualization



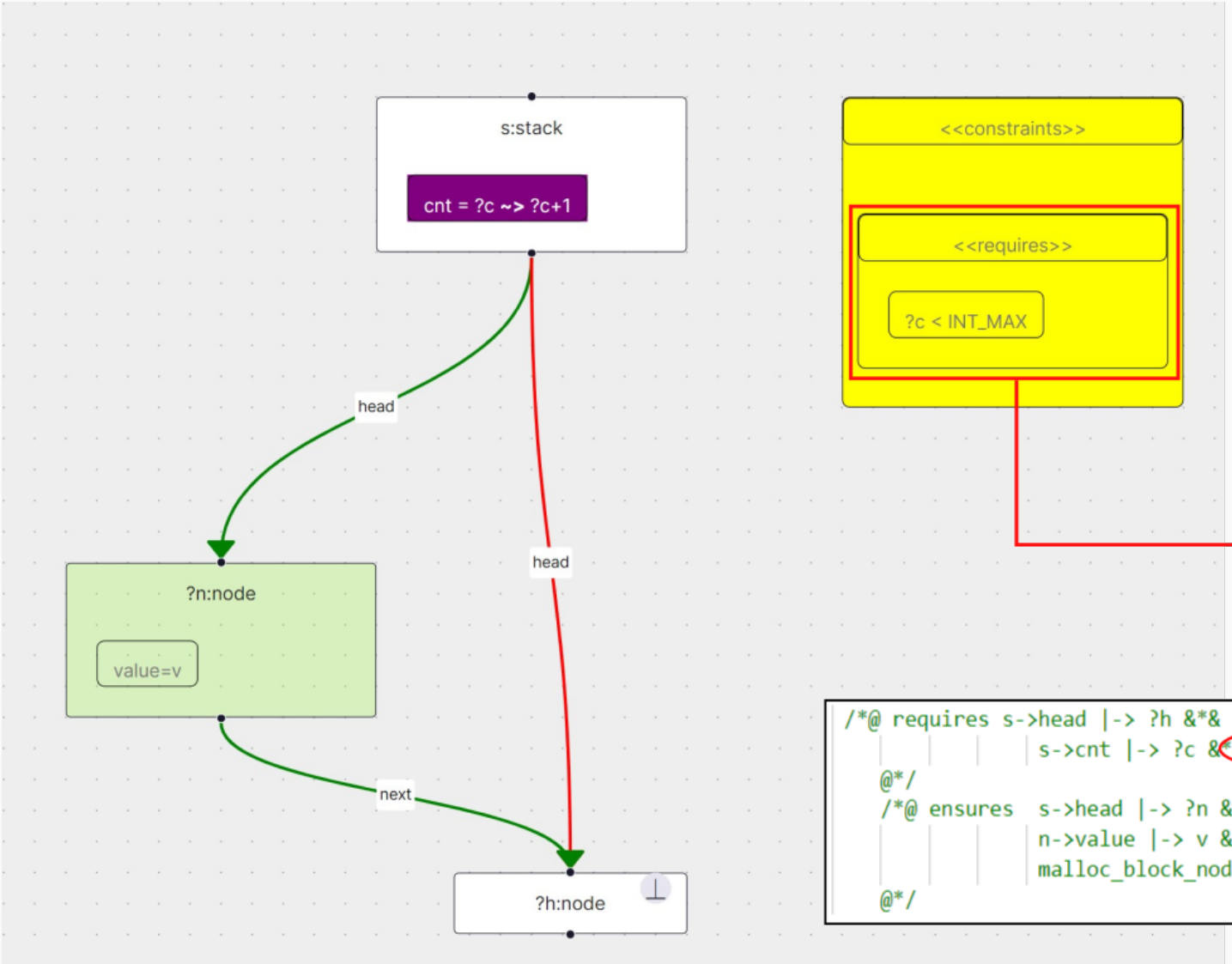
push(struct stack *s, int v) visualization



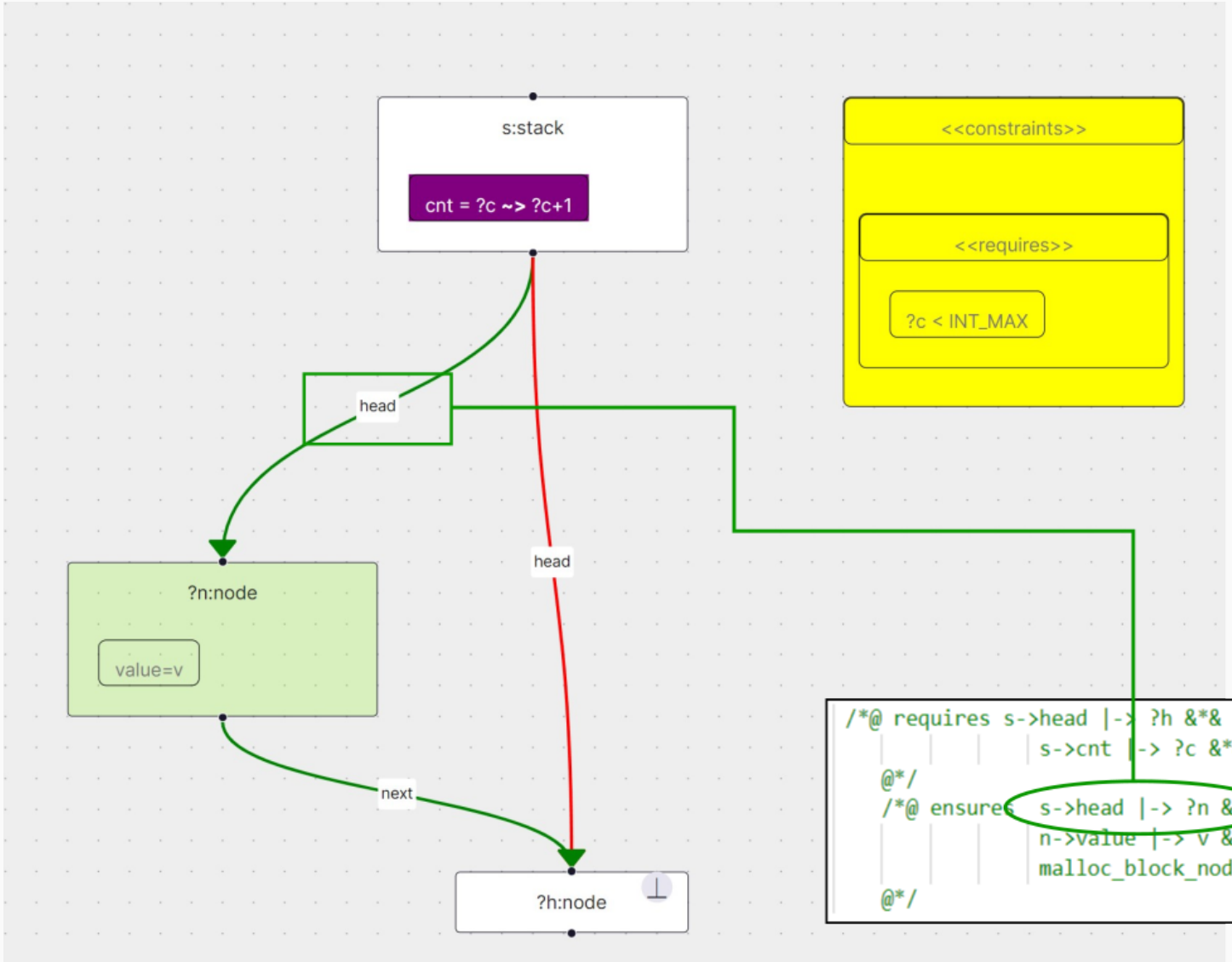
push(struct stack *s, int v) visualization



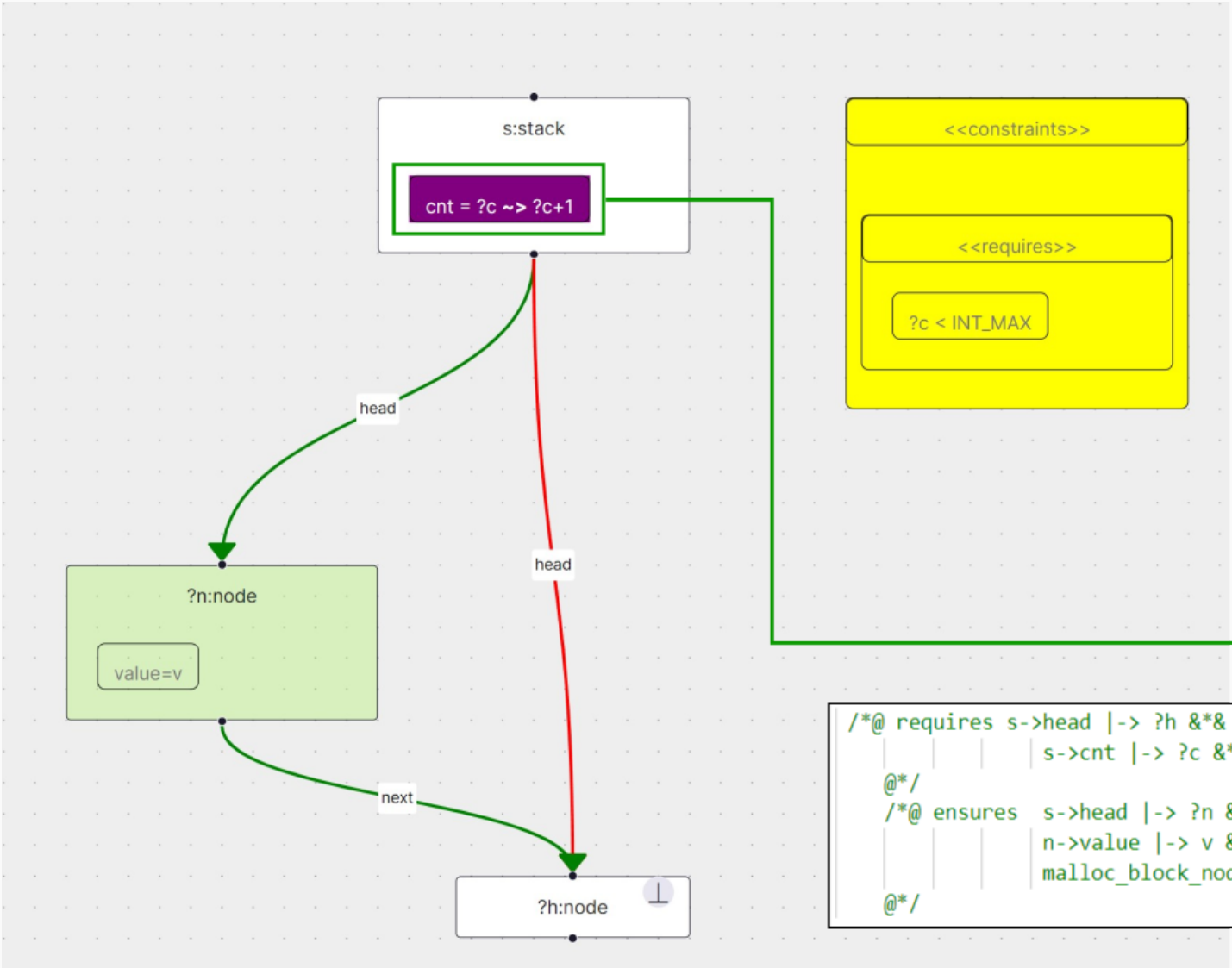
push(struct stack *s, int v) visualization



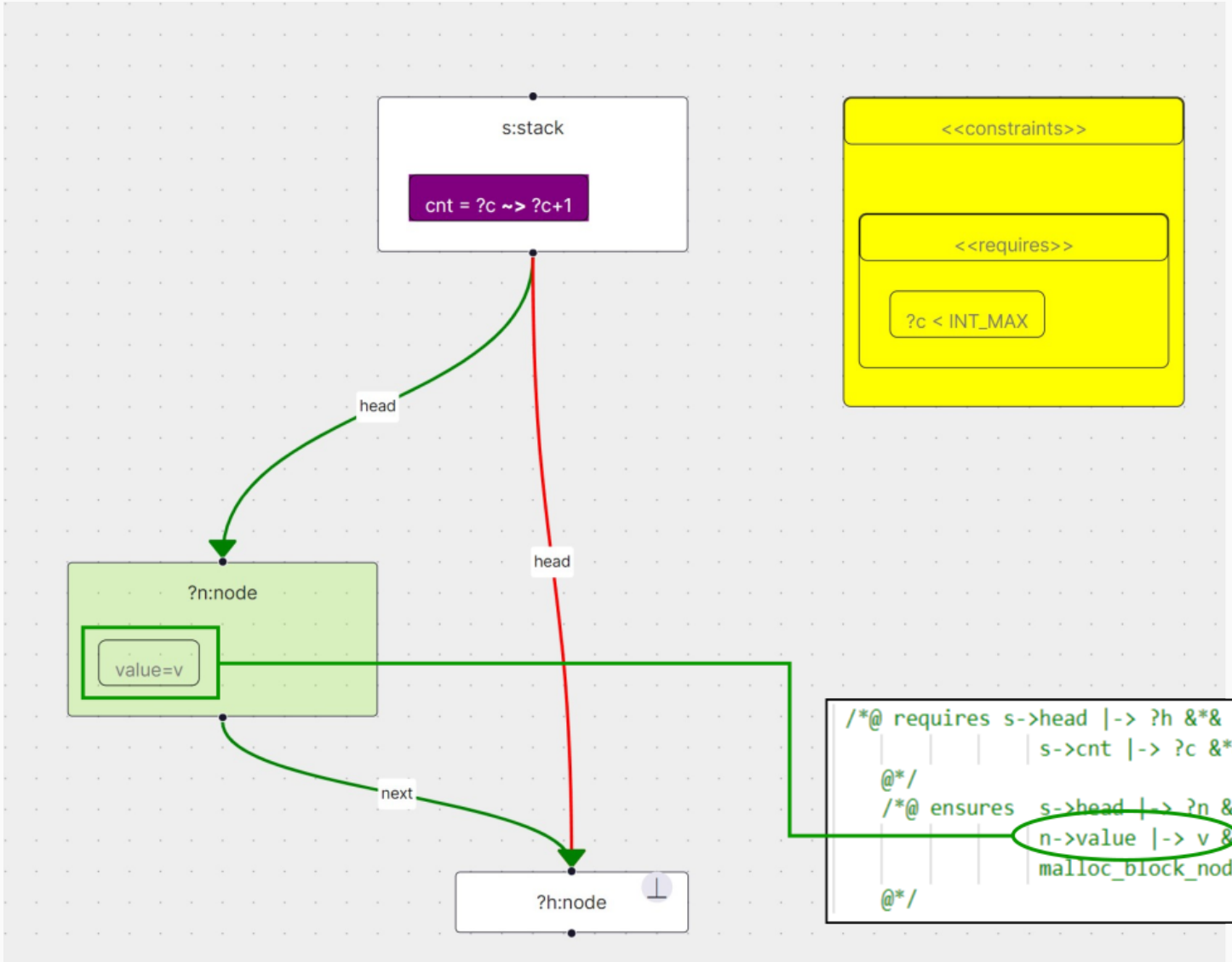
push(struct stack *s, int v) visualization



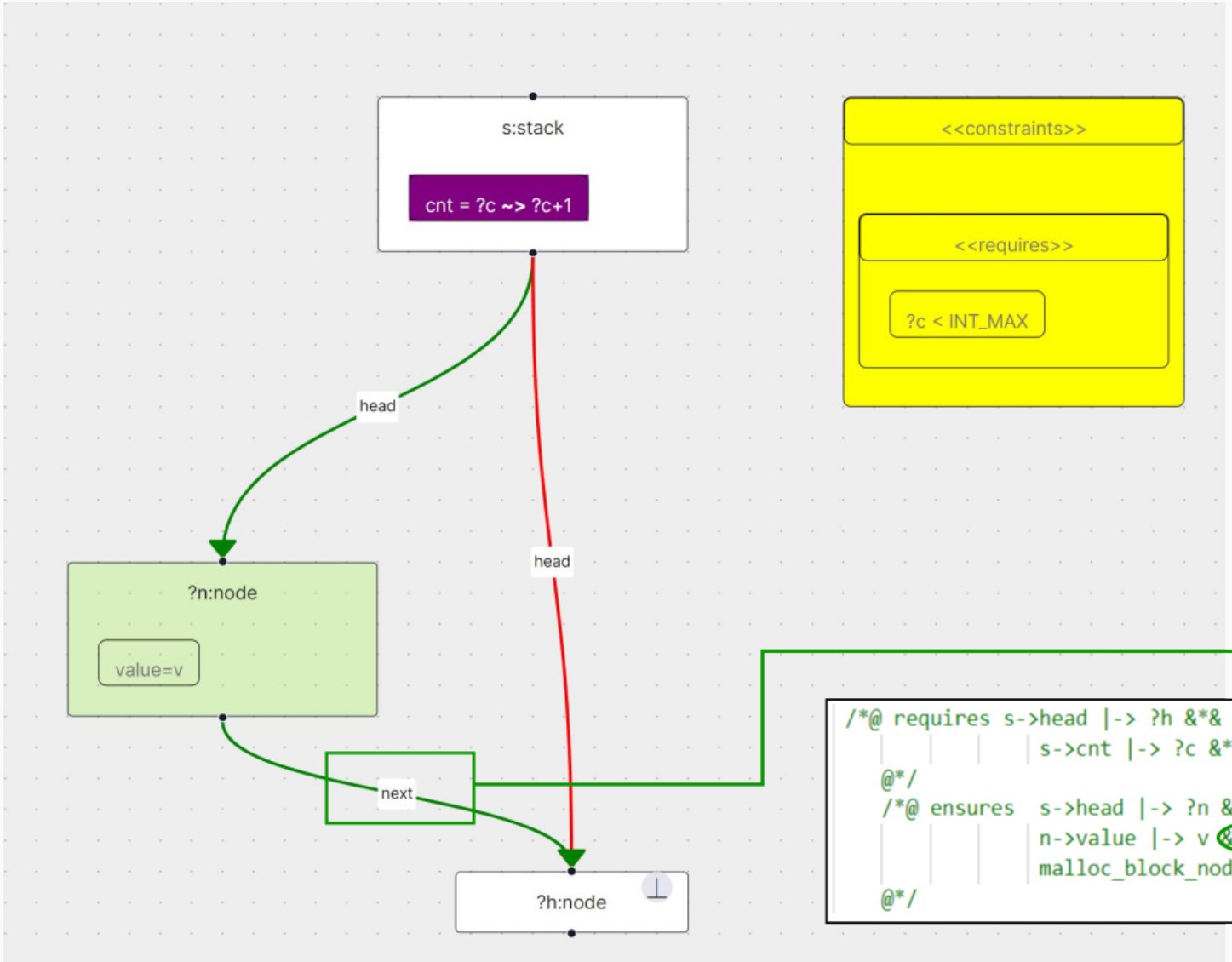
push(struct stack *s, int v) visualization



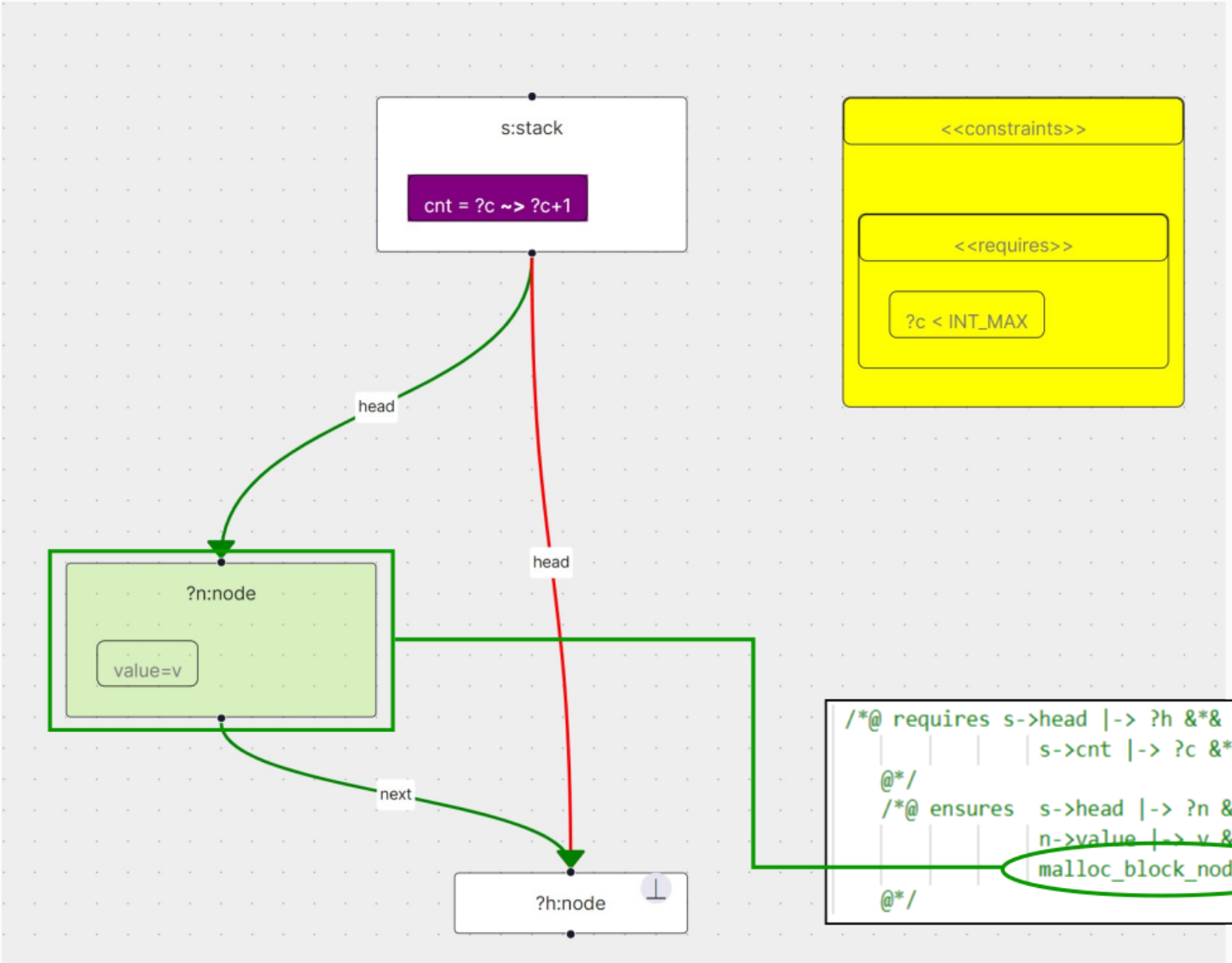
push(struct stack *s, int v) visualization



push(struct stack *s, int v) visualization



push(struct stack *s, int v) visualization



Cognitive Features

Cognitive Features

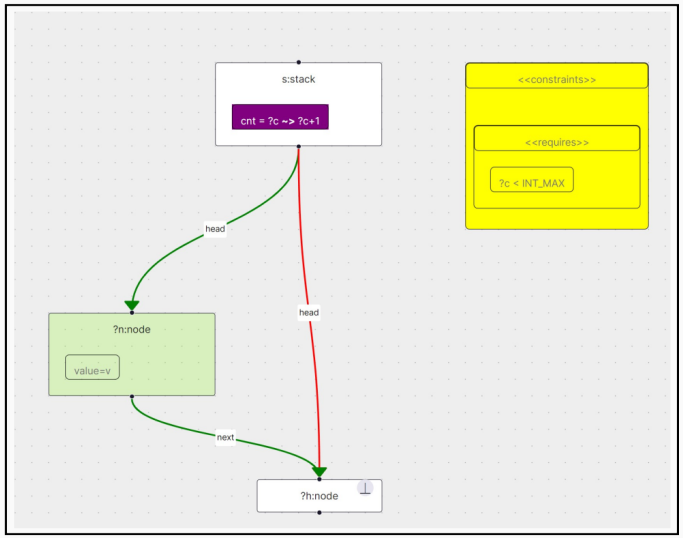
- the user now has a visual representation in addition to the textual contracts
- a way of evalutating the usefulness of the visualization is necessary
- neuroscientists have explored different approaches
- one approach: Card et al. (1999) "Using vision to think"

➡ we evaluate our tool based on six cognitive features from "Using vision to think"

Increasing Memory

"increasing the memory and processing resources available to the users"

➡ outsource information by creating a graph that represents the information from the contracts

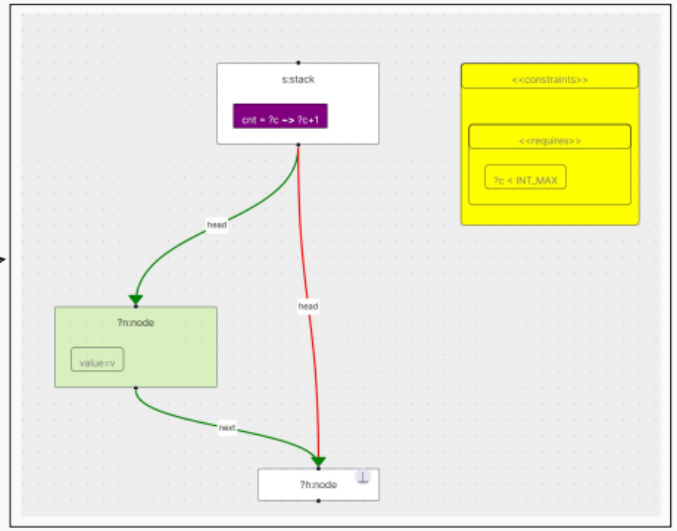


Search for Information

"reducing the search for information"

➡ differences in pre- and postcondition states are easier to detect as they are in one graph and distinguishable by colors

```
/*@ requires s->head |-> ?h &*&
             s->cnt |-> ?c &*& c < INT_MAX;
@*/
/*@ ensures  s->head |-> ?n &*& s->cnt |-> c+1 &*&
             n->value |-> v &*& n->next |-> h &*&
             malloc_block_node(n);
@*/
```

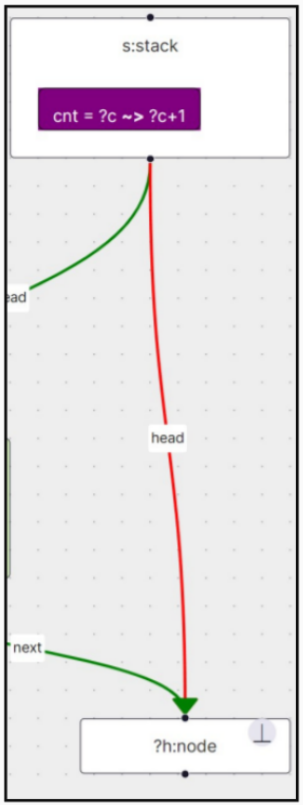


Detection of Patterns

"enhancing the detection of patterns"

- pointers are easier to recognize as they are simple edges in our graph

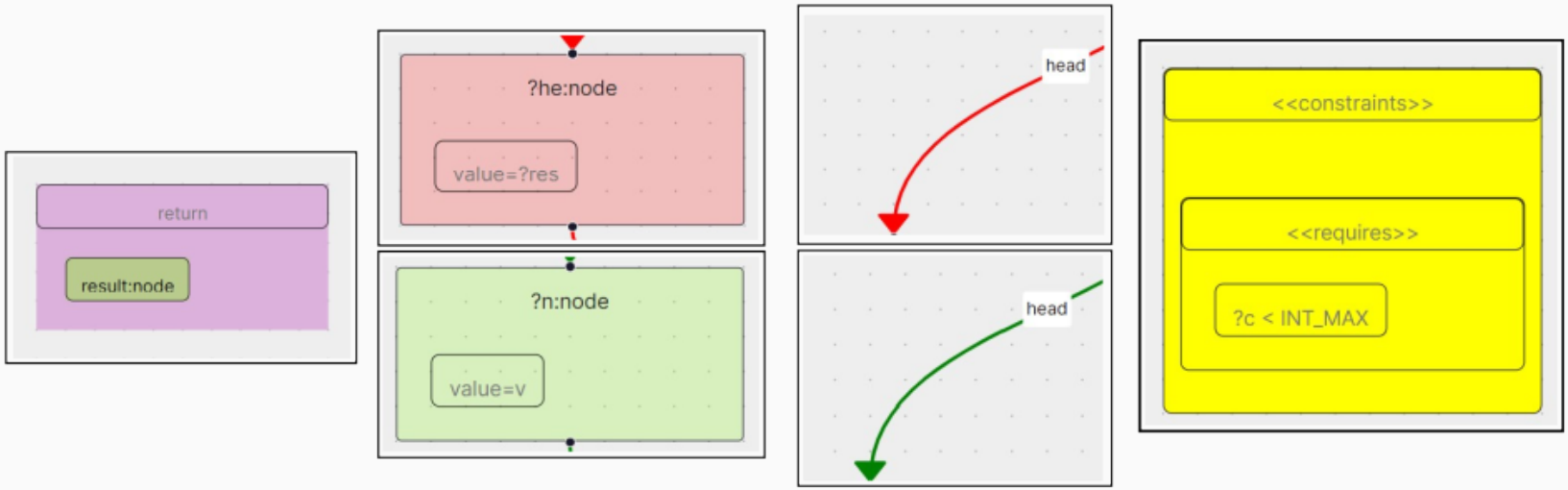
```
/*@ requires s->head |-> ?h &*&
             s->cnt |-> ?c &*& c < INT_MAX;
@*/
```



Perceptual Inference Operations

"enabling perceptual inference operations"

➡ using the same shapes and colors for objects of the same type (instances, return values, constraints, etc.)



Perceptual Attention Mechanisms

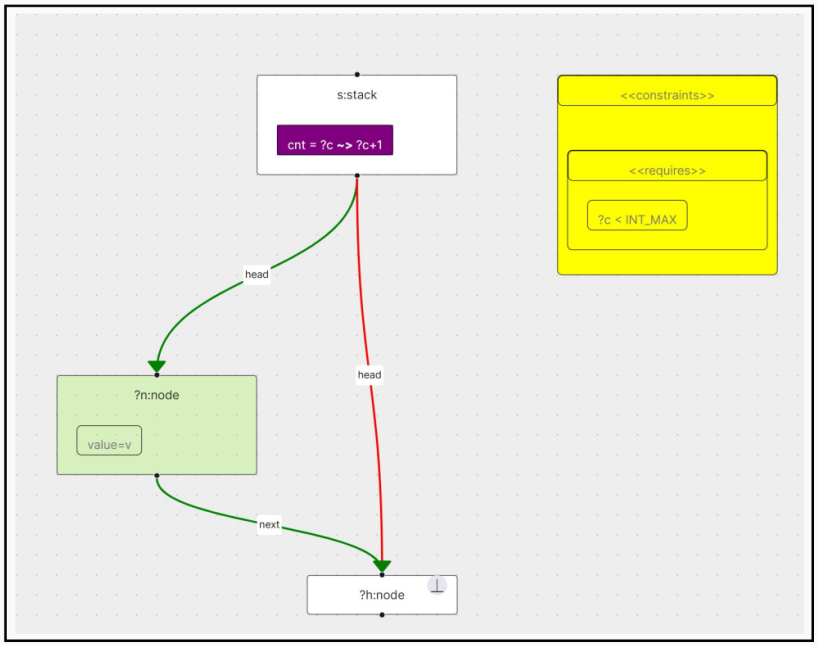
"using perceptual attention mechanisms"

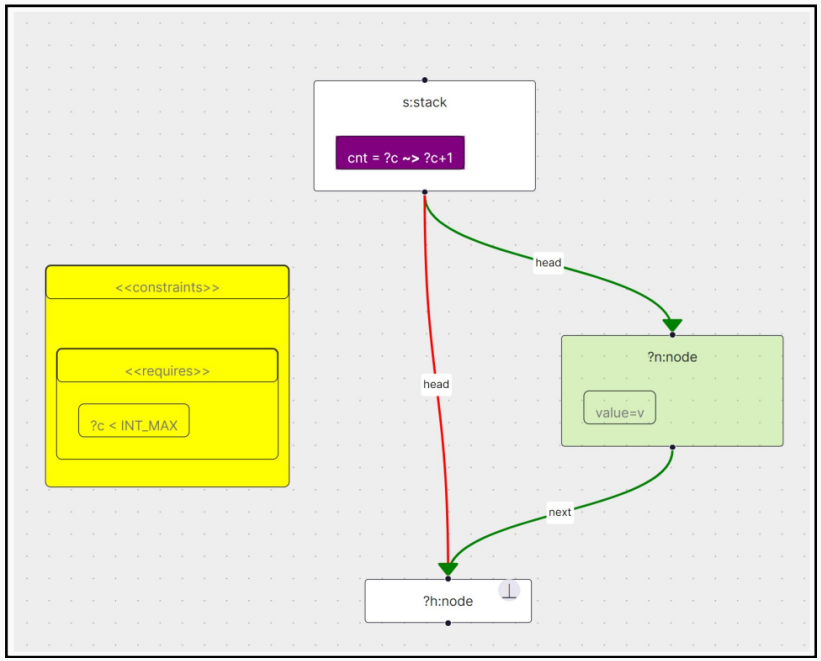
➡ not yet implemented: highlight differences after changes to the contract have been made

Manipulable Medium

”using a manipulable medium”

➡ shapes can be arranged and sorted to match the users demands





Summary

Summary

- we have realized a visualization of VeriFast contracts
- first version will be available online soon
- next step is to let students use the visualization tool and give feedback
- finally, our goal is to see our tool in use not only at our university...

➡ stay tuned for Bastians talk to get technical insights :)

Thank you.