RefScript

Refinement Types for Imperative Scripting Languages

Panagiotis Vekris, Ranjit Jhala

University of California, San Diego

Checking Scripting Languages with an SMT Solver

- Check dynamic language features:

  reflection, *unions, overloading*

- Check generic value properties:

  null, ***array bounds****, termination*

- Check specific invariants:

  jQuery accesses, information flow

Unchecked Downcasts in TypeScript

```
class Animal {
    constructor(public name: string) { }
    move(meters: number) { ... }
}

class Snake extends Animal {
    constructor(name: string) {super(name);}
    slither() { super.move(5); }
}

class Horse extends Animal {
    constructor(name: string) {super(name);}
    gallop() { super.move(45); }
}
```

Unchecked Downcasts in TypeScript

```
class Animal { move(meters: number) {...} }
class Snake extends Animal { slither() {...}}
class Horse extends Animal { gallop() {...}}
```

```
function move(x: Animal) {
  if (x instanceof Horse) {
    var horse = <Horse> x;
    horse.gallop();
  }
  else if (x instanceof Snake) {...}
}
var tom = new Horse("Tommy");
move(tom);
```

Check for **Horse** constructor

Downcast
Animal => Horse

Safe downcast

Unchecked Downcasts in TypeScript

```
class Animal { move(meters: number) {...} }
class Snake extends Animal { slither() {...}}
class Horse extends Animal { gallop() {...}}
```

```
function move(x: Animal) {
  if (x instanceof Snake) {
    var horse = <Horse> x;
    horse.gallop();
  }
  else if (x instanceof Horse) {...}
}
var tom = new Snake("Tommy");
move(tom);
```

Check for **Snake** constructor

Downcast
Animal => Horse

Unsafe downcast
Full Erasure: no RT checks

```
> TypeError: undefined is not a function
```

Safe Downcasting in RefScript

```
class Animal { move(meters: number) {...} }
class Snake extends Animal { slither() {...}}
class Horse extends Animal { gallop() {...}}
```

```
function move(x: Animal) {
  if (x instanceof Snake) {
    var horse = <Horse> x;
    horse.gallop();
  }
  else if (x instanceof Horse) {...}
}
var tom = new Snake("Tommy");
move(tom);
```

Γ

x → Animal

Path sensitive value tracking

Safe Downcasting in RefScript

```
class Animal { move(meters: number) {...} }
class Snake extends Animal { slither() {...}}
class Horse extends Animal { gallop() {...}}
```

```
function move(x: Animal) {
  if (x instanceof Snake) {
    var horse = <Horse> x;
    horse.gallop();
  }
  else if (x instanceof Horse) {...}
}
var tom = new Snake("Tommy");
move(tom);
```

Γ, x → { v:Animal | **instOf(v, Animal)** }

Enhance the refinement with a constructor predicate

Path sensitive value tracking

Safe Downcasting in RefScript

```
class Animal { move(meters: number) {...} }
class Snake extends Animal { slither() {...}}
class Horse extends Animal { gallop() {...}}
```

```
function move(x: Animal) {
  if (x instanceof Snake) {
    var horse = <Horse> x;
    horse.gallop();
  }
  else if (x instanceof Horse) {...}
}
var tom = new Snake("Tommy");
move(tom);
```

Γ

x → { v:Animal | **instOf(v, Animal)** }

z → { v:bool | True(**v**) <=> **instOf(x, Snake)** }

z = x instanceof Snake

```
instanceof :: ∀ T A . (arg: T, typeof A) =>
  { v: bool | True(v) <=> instOf(arg,A) }
```

Path sensitive value tracking

Safe Downcasting in RefScript

```
class Animal { move(meters: number) {...} }
class Snake extends Animal { slither() {...}}
class Horse extends Animal { gallop() {...}}
```

```
function move(x: Animal) {
  if (x instanceof Snake) {
    var horse = <Horse> x;
    horse.gallop();
  }
  else if (x instanceof Horse) {...}
}
var tom = new Snake("Tommy");
move(tom);
```

Path sensitive value tracking

Γ

x →	{ v:Animal | **instOf(v, Animal)** }
z →	{ v:bool | True(**v**) <=> **instOf(x, Snake)** }
Grd	**True(z)**

z = x instanceof Snake

$$\frac{\Gamma,\ \texttt{grds: True(c)} \vdash \texttt{s1}}{\Gamma \vdash \texttt{if (c) \{ s1 \}}}$$

Safe Downcasting in RefScript

```
class Animal { move(meters: number) {...} }
class Snake extends Animal { slither() {...}}
class Horse extends Animal { gallop() {...}}
```

```
function move(x: Animal) {
  if (x instanceof Snake) {
    var horse = <Horse> x;
    horse.gallop();
  }
  else if (x instanceof Horse) {...}
}
var tom = new Snake("Tommy");
move(tom);
```

Path sensitive value tracking

Γ

x →	{ v:Animal | **instOf(v, Animal)** }
z →	{ v:bool | True(**v**) <=> **instOf(x, Snake)** }
Grd	**True(z)**
Goal	???

z = x instanceof Snake

<Animal => Horse> x

Safe Downcasting in RefScript

```
class Animal { move(meters: number) {...} }
class Snake extends Animal { slither() {...}}
class Horse extends Animal { gallop() {...}}
```

```
function move(x: Animal) {
  if (x instanceof Snake) {
    var horse = <Horse> x;
    horse.gallop();
  }
  else if (x instanceof Horse) {...}
}
var tom = new Snake("Tommy");
move(tom);
```

Path sensitive value tracking

Γ

x → { v:Animal | **instOf(v, Animal)** }

z → { v:bool | True(**v**) <=> **instOf(x, Snake)** }

Grd **True(z)**

Goal {v:Animal|v=x} <: Horse

z = x instanceof Snake

<Animal => Horse> x

Safe Downcasting in RefScript

```
class Animal { move(meters: number) {...} }
class Snake extends Animal { slither() {...}}
class Horse extends Animal { gallop() {...}}
```

```
function move(x: Animal) {
  if (x instanceof Snake) {
    var horse = <Horse> x;
    horse.gallop();
  }
  else if (x instanceof Horse) {...}
}
var tom = new Snake("Tommy");
move(tom);
```

Path sensitive value tracking

Γ

x →	{ v:Animal | **instOf(v, Animal)** }
z →	{ v:bool | True(**v**) <=> **instOf(x, Snake)** }
Grd	**True(z)**
Goal	{v:Animal|v=x} <: {v:Animal|instOf(v,Horse)}

z = x instanceof Snake

<Animal => Horse> x

Safe Downcasting in RefScript

```
class Animal { move(meters: number) {...} }
class Snake extends Animal { slither() {...}}
class Horse extends Animal { gallop() {...}}
```

```
function move(x: Animal) {
  if (x instanceof Snake) {
    var horse = <Horse> x;
    horse.gallop();
  }
  else if (x instanceof Horse) {...}
}
var tom = new Snake("Tommy");
move(tom);
```

Path sensitive value tracking

Γ

x →	{ v:Animal | **instOf(v, Animal)** }
z →	{ v:bool | True(**v**) <=> **instOf(x, Snake)** }
Grd	**True(z)**
Goal	(v=x) => instOf(x,Horse)

z = x instanceof Snake

<Animal => Horse> x

Safe Downcasting in RefScript

```
class Animal { move(meters: number) {...} }
class Snake extends Animal { slither() {...}}
class Horse extends Animal { gallop() {...}}
```

```
function move(x: Animal) {
  if (x instanceof Snake) {
    var horse = <Horse> x;
    horse.gallop();
  }
  else if (x instanceof Horse) {...}
}
var tom = new Snake("Tommy");
move(tom);
```

Path sensitive value tracking

Γ

x → { v:Animal | **instOf(v, Animal)** }

z → { v:bool | True(**v**) <=> **instOf(x, Snake)** }

Grd **True(z)**

Goal (v=x) => instOf(x,Horse)

Is this valid?

Safe Downcasting in RefScript

```
class Animal { move(meters: number) {...} }
class Snake extends Animal { slither() {...}}
class Horse extends Animal { gallop() {...}}
```

```
function move(x: Animal) {
  if (x instanceof Snake) {
    var horse = <Horse> x;
    horse.gallop();
  }
  else if (x instanceof Horse) {...}
}
var tom = new Snake("Tommy");
move(tom);
```

Path sensitive value tracking

Γ

x →	{ v:Animal | **instOf(v, Animal)** }
z →	{ v:bool | True(**v**) <=> **instOf(x, Snake)** }

Grd **True(z)**

Goal (v=x) => instOf(x,Horse)

Is this valid? → **No**

Liquid Types

Safe Array Access in RefScript

```
function idxOf<T>(a: Array<T>,
    pred: (v: T) => boolean): number {

  for (var i = 0, i < a.length; i++) {

    if (pred(a[i])) return i;

  }
  throw new Error("Not found");
}
```

From src/compiler/core/ArraryUtils.ts

Safe Array Access in RefScript

```
function idxOf<T>(a: Array<T>,
    pred: (v: T) => boolean): number {

  for (var i = 0, i < a.length; i++) {

    if (pred(a[i])) return i;

  }
  throw new Error("Not found");
}
```

From src/compiler/core/ArraryUtils.ts

SSA Expansion

```
function idxOf<T>(a: Array<T>,
    pred: (v: T) => boolean): number {

  var i_0 = 0;

  var i_2 = i_0;

  while [i_2](i_2 < a.length) {

    if (pred(a[i_2])) return i_2;

    i_1 = i_2 + 1;

    i_2 = i_1;
  }
  throw new Error("Not found");
}
```

Safe Array Access in RefScript

Φ – Vars

Capture the loop invariants

`i_2 :: { v: number | K_i }`

SSA Expansion

```
function idxOf<T>(a: Array<T>,
    pred: (v: T) => boolean): number {

  var i_0 = 0;

  var i_2 = i_0;

  while [i_2](i_2 < a.length) {

    if (pred(a[i_2])) return i_2;

    i_1 = i_2 + 1;

    i_2 = i_1;
  }
  throw new Error("Not found");
}
```

Φ - Vars

Safe Array Access in RefScript

Φ – Vars

Capture the loop invariants

```
i_2 :: { v: number | K_i }
```

Can be inferred based on **constraints**:

- **Base**

  `Γ(i_0) <: Γ(i_2)`

- **Loop update**

  `Γ'(i_1) <: Γ'(i_2)`

  Where

  `Γ'= Γ,` `grd: i_2 < a.length`

SSA Expansion

```
function idxOf<T>(a: Array<T>,
    pred: (v: T) => boolean): number {

  var i_0 = 0;

  var i_2 = i_0;

  while [i_2](i_2 < a.length) {

    if (pred(a[i_2])) return i_2;

    i_1 = i_2 + 1;

    i_2 = i_1;
  }
  throw new Error("Not found");
}
```

Safe Array Access in RefScript

$\Gamma(i\_0) <: \Gamma(i\_2)$

Assignement

$\vdash \{v:number|v=0\} <: \{v:number|K_i\}$

`i_2 :: { v: number | K_i }`

SSA Expansion

```
function idxOf<T>(a: Array<T>,
    pred: (v: T) => boolean): number {

  var i_0 = 0;

  var i_2 = i_0;

  while [i_2](i_2 < a.length) {

    if (pred(a[i_2])) return i_2;

    i_1 = i_2 + 1;

    i_2 = i_1;
  }
  throw new Error("Not found");
}
```

Safe Array Access in RefScript

$\Gamma(i\_0) <: \Gamma(i\_2)$

Assignement

$\vdash \{ v=0 \} <: K_i$

`i_2 :: { v: number | K_i }`

SSA Expansion

```
function idxOf<T>(a: Array<T>,
    pred: (v: T) => boolean): number {

  var i_0 = 0;

  var i_2 = i_0;

  while [i_2](i_2 < a.length) {

    if (pred(a[i_2])) return i_2;

    i_1 = i_2 + 1;

    i_2 = i_1;
  }
  throw new Error("Not found");
}
```

Safe Array Access in RefScript

$\Gamma(i\_0) <: \Gamma(i\_2)$

$\vdash \{ v=0 \} <: K_i$

`i_2 :: { v: number | K_i }`

SSA Expansion

```
function idxOf<T>(a: Array<T>,
    pred: (v: T) => boolean): number {

  var i_0 = 0;

  var i_2 = i_0;

  while [i_2](i_2 < a.length) {

    if (pred(a[i_2])) return i_2;

    i_1 = i_2 + 1;

    i_2 = i_1;
  }
  throw new Error("Not found");
}
```

Safe Array Access in RefScript

$\Gamma(i\_0) <: \Gamma(i\_2)$

$\vdash \{ v=0 \} <: K_i$

Guard

```
getProp :: (a: Array<T>, x:string) =>
  { v:_ | x = "length" => v=len a }
```

```
ltOp:: (x: number, y: number) =>
 { v: bool | True(v) <=> x < y }
```

`i_2 :: { v: number | K_i }`

SSA Expansion

```
function idxOf<T>(a: Array<T>,
    pred: (v: T) => boolean): number {

  var i_0 = 0;

  var i_2 = i_0;

  while [i_2](i_2 < a.length) {

    if (pred(a[i_2])) return i_2;

    i_1 = i_2 + 1;

    i_2 = i_1;
  }
  throw new Error("Not found");
}
```

Safe Array Access in RefScript

$\Gamma(i\_0) <: \Gamma(i\_2)$

$\vdash \{ v=0 \} <: K_i$

Guard

Grd: i_2 < (len a)

i_2 :: { v: number | K_i }

SSA Expansion

```
function idxOf<T>(a: Array<T>,
    pred: (v: T) => boolean): number {

  var i_0 = 0;

  var i_2 = i_0;

  while [i_2](i_2 < a.length) {

    if (pred(a[i_2])) return i_2;

    i_1 = i_2 + 1;

    i_2 = i_1;
  }
  throw new Error("Not found");
}
```

Safe Array Access in RefScript

$\Gamma(i\_0) <: \Gamma(i\_2)$

$\vdash \{ v=0 \} <: K_i$

Guard

Grd: i_2 < (len a)

$\Gamma'(i\_1) <: \Gamma'(i\_2)$

$Grd \vdash \{v = i\_2 + 1\} <: K_i$

`i_2 :: { v: number | K_i }`

SSA Expansion

```
function idxOf<T>(a: Array<T>,
    pred: (v: T) => boolean): number {

  var i_0 = 0;

  var i_2 = i_0;

  while [i_2](i_2 < a.length) {

    if (pred(a[i_2])) return i_2;

    i_1 = i_2 + 1;

    i_2 = i_1;
  }
  throw new Error("Not found");
}
```

Safe Array Access in RefScript

$\vdash$ { v=0 } <: K_i

Grd: i_2 < (len a)

Grd $\vdash$ {v = i_2 + 1} <: K_i

i_2 :: { v: number | K_i }

SSA Expansion

```
function idxOf<T>(a: Array<T>,
    pred: (v: T) => boolean): number {

  var i_0 = 0;

  var i_2 = i_0;

  while [i_2](i_2 < a.length) {

    if (pred(a[i_2])) return i_2;

    i_1 = i_2 + 1;

    i_2 = i_1;
  }
  throw new Error("Not found");
}
```

Safe Array Access in RefScript

$\vdash$ { v=0 } <: K_i

Grd: i_2 < (len a)

Grd $\vdash$ {v = i_2 + 1} <: K_i

Array bounds check

Goal

```
arrIdx::
 (a: Array<T>, {0≤v ∧ v<(len a)) => T
```

```
i_2 :: { v: number | K_i }
```

SSA Expansion

```
function idxOf<T>(a: Array<T>,
    pred: (v: T) => boolean): number {

  var i_0 = 0;

  var i_2 = i_0;

  while [i_2](i_2 < a.length) {

    if (pred(a[i_2])) return i_2;

    i_1 = i_2 + 1;

    i_2 = i_1;
  }
  throw new Error("Not found");
}
```

Safe Array Access in RefScript

$\vdash$ { v=0 } <: K_i

Grd: i_2 < (len a)

Grd $\vdash$ {v = i_2 + 1} <: K_i

Grd $\vdash$ K_i <: { 0≤v ∧ v<(len a)}

i_2 :: { v: number | K_i }

SSA Expansion

```
function idxOf<T>(a: Array<T>,
    pred: (v: T) => boolean): number {

  var i_0 = 0;

  var i_2 = i_0;

  while [i_2](i_2 < a.length) {

    if (pred(a[i_2])) return i_2;

    i_1 = i_2 + 1;

    i_2 = i_1;
  }
  throw new Error("Not found");
}
```

Safe Array Access in RefScript

$\vdash$ { v=0 } <: K_i

Grd: i_2 < (len a)

Grd $\vdash$ {v = i_2 + 1} <: K_i

Grd $\vdash$ K_i <: { 0≤v ∧ v<(len a)}

Is there a solution for K_i?

`i_2 :: { v: number | K_i }`

SSA Expansion

```
function idxOf<T>(a: Array<T>,
    pred: (v: T) => boolean): number {

  var i_0 = 0;

  var i_2 = i_0;

  while [i_2](i_2 < a.length) {

    if (pred(a[i_2])) return i_2;

    i_1 = i_2 + 1;

    i_2 = i_1;
  }
  throw new Error("Not found");
}
```

Safe Array Access in RefScript

$\vdash$ { v=0 } <: K_i

Grd: i_2 < (len a)

Grd $\vdash$ {v = i_2 + 1} <: K_i

Grd $\vdash$ K_i <: { 0≤v ∧ v<(len a)}

Is there a solution for K_i?

Liquid Types

Solution: K_i = 0≤v ∧ v<(len a)

i_2 :: { v: number | K_i }

SSA Expansion

```
function idxOf<T>(a: Array<T>,
    pred: (v: T) => boolean): number {

  var i_0 = 0;

  var i_2 = i_0;

  while [i_2](i_2 < a.length) {

    if (pred(a[i_2])) return i_2;

    i_1 = i_2 + 1;

    i_2 = i_1;
  }
  throw new Error("Not found");
}
```

Safe Array Access in RefScript

$\vdash$ { v=0 } <: K_i

Grd: i_2 < (len a)

Grd $\vdash$ {v = i_2 + 1} <: K_i

Grd $\vdash$ K_i <: { 0≤v ∧ v<(len a)}

Is there a solution for K_i?

Liquid Types

Solution: K_i = 0≤v ∧ v<(len a)

i_2 :: { v: number | K_i }

SSA Expansion

```
function idxOf<T>(a: Array<T>,
    pred: (v: T) => boolean): number {

  var i_0 = 0;

  var i_2 = i_0;

  while [i_2](i_2 < a.length) {

    if (pred(a[i_2])) return i_2;

    i_1 = i_2 + 1;

    i_2 = i_1;
  }
  throw new Error("Not found");
}
```

Safe Array Access in RefScript

Caution:
Mutable arrays

```
function idxOf<T>(a: Array<T>,
    pred: (v: T) => boolean): number {

  for (var i = 0, i < a.length; i++) {

    a.pop();

    if (pred(a[i])) return i;

  }
  throw new Error("Not found");
}
```

Safe Array Access in RefScript

Caution:
Mutable arrays

```
function idxOf<T>(a: Array<T>,
    pred: (v: T) => boolean): number {
  for (var i = 0, i < a.length; i++) {
    a.pop();
    if (pred(a[i])) return i;
  }
  throw new Error("Not found");
}
```

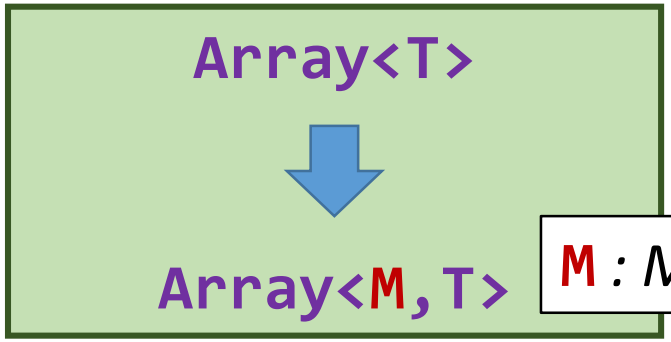
Calling **pop** changes the array's length

`a[i]` may be **undefined**

Safe Array Access in RefScript

```
function idxOf<M,T>(a: Array<M,T>,
    pred: (v: T) => boolean): number {

  for (var i = 0, i < a.length; i++) {

    a.pop();

    if (pred(a[i])) return i;

  }
  throw new Error("Not found");
}
```

Idea:
Mutability Modifiers as Generic Annotations [Zibin'07]




```
interface Array<M,T> {
  ...
  pop(this: Array<Mut, T>): T;
  get length(this: Array<Mut,T>): number;
  get length(this: Array<Imm,T>):
       {v: number | v = len this }
}
```

Safe Array Access in RefScript

Idea:
Mutability Modifiers as Generic Annotations [Zibin'07]

```
function idxOf<T>(a: Array<Mut,T>,
    pred: (v: T) => boolean): number {

  for (var i = 0, i < a.length; i++) {

    a.pop();

    if (pred(a[i])) return i;

  }
  throw new Error("Not found");
}
```

If **M = Mut**

- call to **a.pop()** **succeeds**
- **a.length :: number**
- **Array bound check fails**

```
interface Array<M,T> {
  ...
  pop(this: Array<Mut, T>): T;
  get length(this: Array<Mut,T>): number;
  get length(this: Array<Imm,T>):
       {v: number | v = len this }
}
```

Safe Array Access in RefScript

Idea:
Mutability Modifiers as Generic Annotations [Zibin'07]

```
function idxOf<T>(a: Array<Imm,T>,
    pred: (v: T) => boolean): number {

  for (var i = 0, i < a.length; i++) {

    a.pop();

    if (pred(a[i])) return i;

  }
  throw new Error("Not found");
}
```

If **M = Imm**

- call to **a.pop() fails**

```
interface Array<M,T> {
  ...
  pop(this: Array<Mut, T>): T;
  get length(this: Array<Mut,T>): number;
  get length(this: Array<Imm,T>):
       {v: number | v = len this }
}
```

Safe Array Access in RefScript

Idea:
Mutability Modifiers as Generic Annotations [Zibin'07]

```
function idxOf<M,T>(a: Array<M,T>,
    pred: (v: T) => boolean): number {

  for (var i = 0, i < a.length; i++) {

    a.pop();

    if (pred(a[i])) return i;

  }
  throw new Error("Not found");
}
```

- ✓ **Sound** result in either case
- ✓ **Easy** integration with generics, overloading

```
interface Array<M,T> {
  ...
  pop(this: Array<Mut, T>): T;
  get length(this: Array<Mut,T>): number;
  get length(this: Array<Imm,T>):
       {v: number | v = len this }
}
```

End