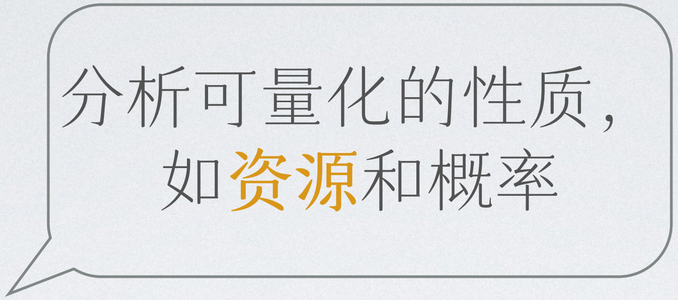
量化程序分析与验证

王迪
北京大学

量化程序分析与验证

(Quantitative Program Analysis and Verification)



量化程序分析与验证

(Quantitative Program Analysis and Verification)

为什么需要量化分析与验证?

资源

时间

能源

...

Energy consumption of AI poses environmental problems

Data centers and large AI models use massive amounts of energy and are harmful to the environment. Businesses can take action to lower their environmental impact.

TechTarget

Sources: https://www.techtarget.com/searchenterpriseai/feature/Energy-consumption-of-AI-poses-environmental-problems; https://www.wired.com/story/ai-great-things-burn-planet/.

为什么需要量化分析与验证？

资源
时间
能源
…

追求**高性能**软件，降低能源足迹

为什么需要量化分析与验证?

资源

时间
能源
…

追求**高性能**软件，降低能源足迹

概率

随机性
不确定性
…

IEEE Spectrum

Intel Starts R&D Effort in Probabilistic Computing for AI ›Seeks ways to help self-driving cars and autonomous robots deal with the uncertainty of the real world

Source: https://spectrum.ieee.org/intel-starts-rd-effort-in-probabilistic-computing-for-ai.

为什么需要量化分析与验证？

资源

时间
能源
…

追求**高性能**软件，降低能源足迹

概率

随机性
不确定性
…

追求**高可靠性**软件，更好地与真实世界交互

为什么需要量化分析与验证？

资源

时间
能源
…

追求**高性能**软件，降低能源足迹

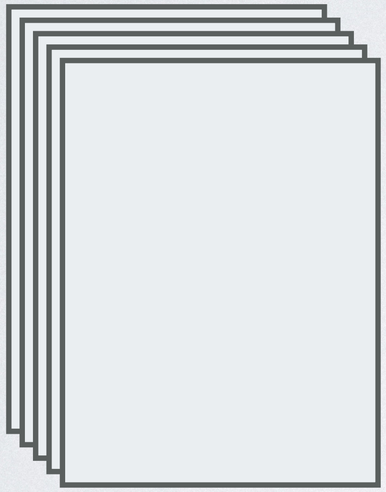
概率

随机性
不确定性
…

追求**高可靠性**软件，更好地与真实世界交互

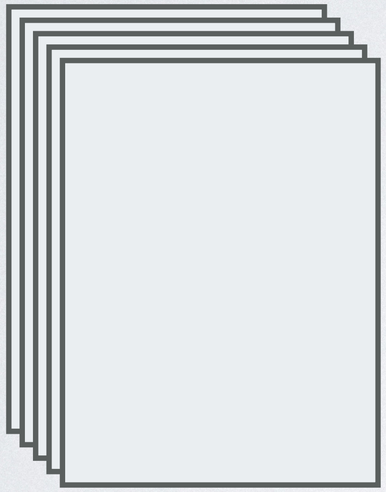
机遇与挑战：传统程序分析注重定性；定量分析往往更难

程序的资源消耗



程序

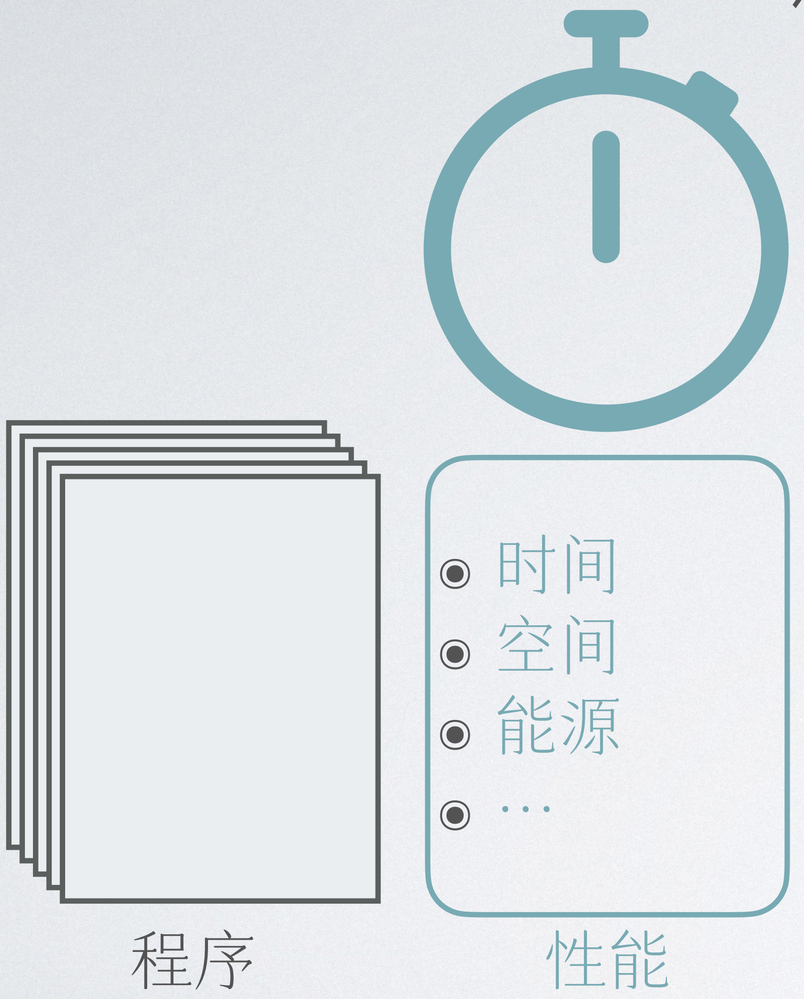
程序的资源消耗



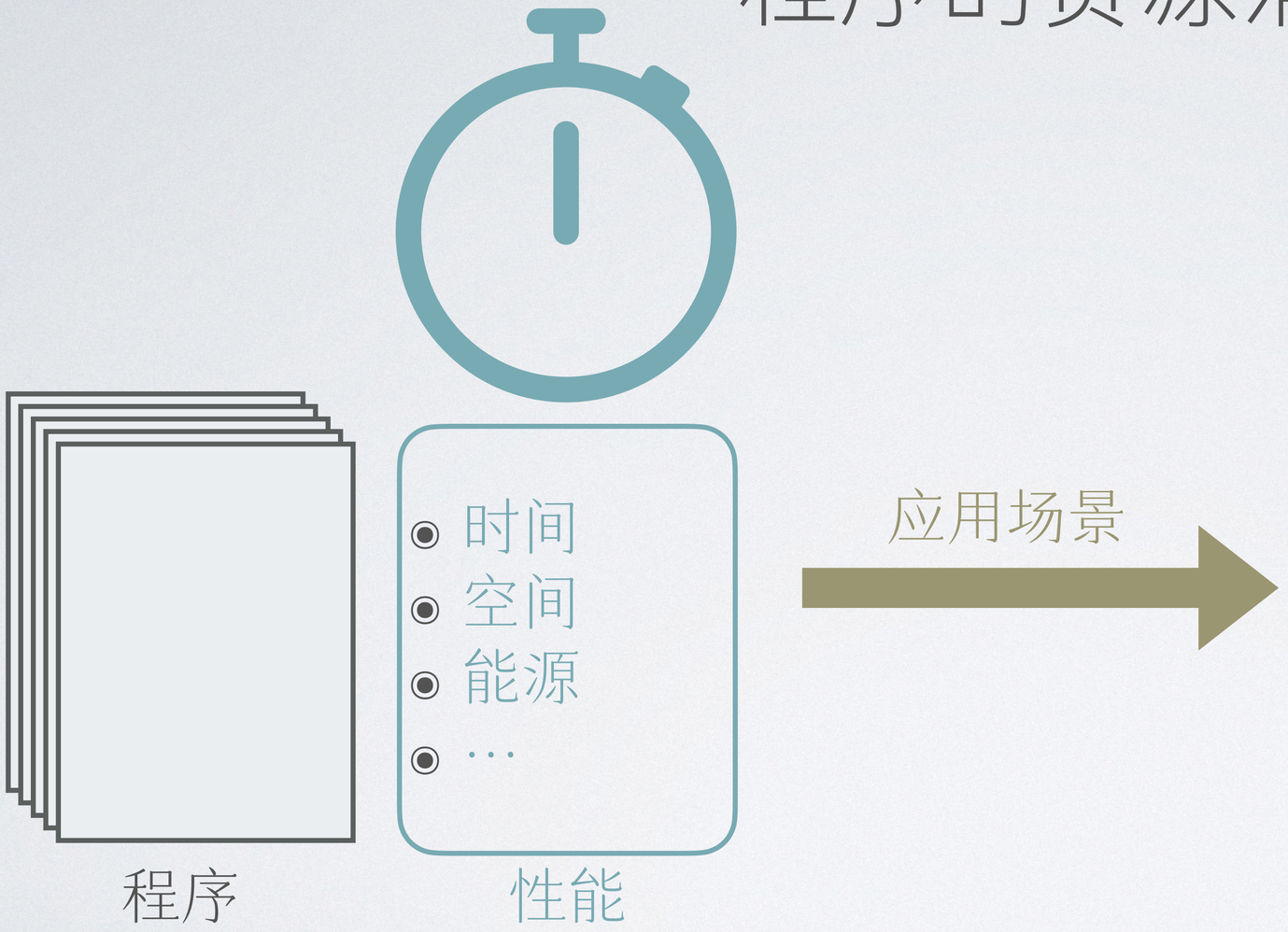
程序

性能

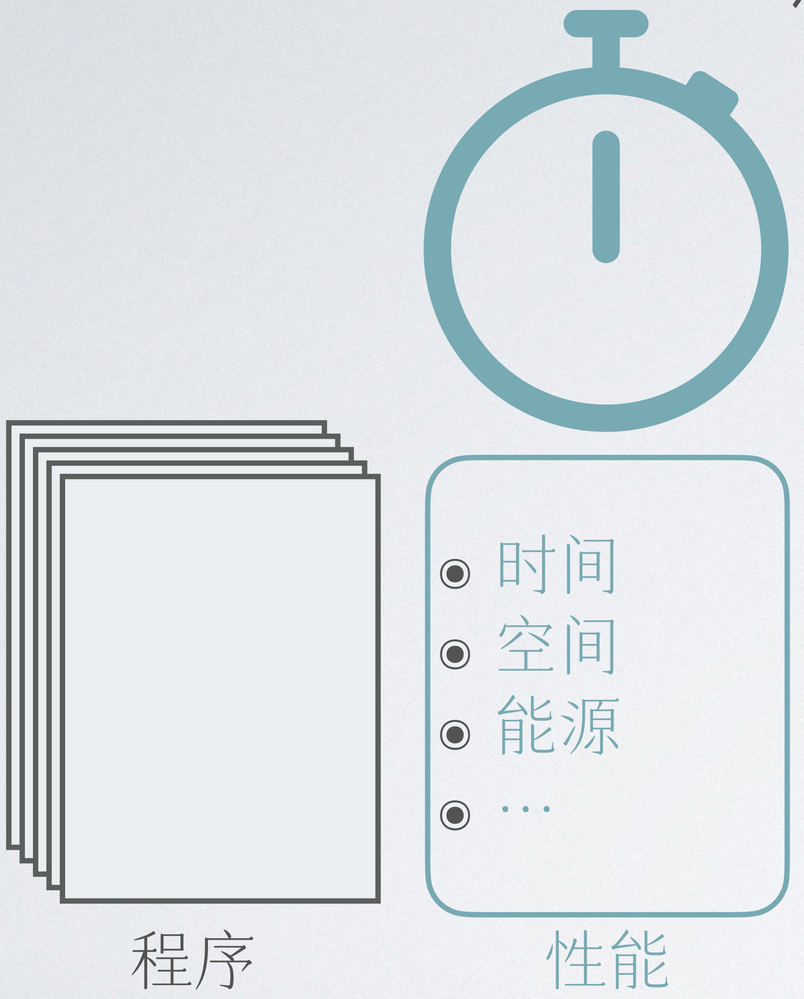
程序的资源消耗



程序的资源消耗



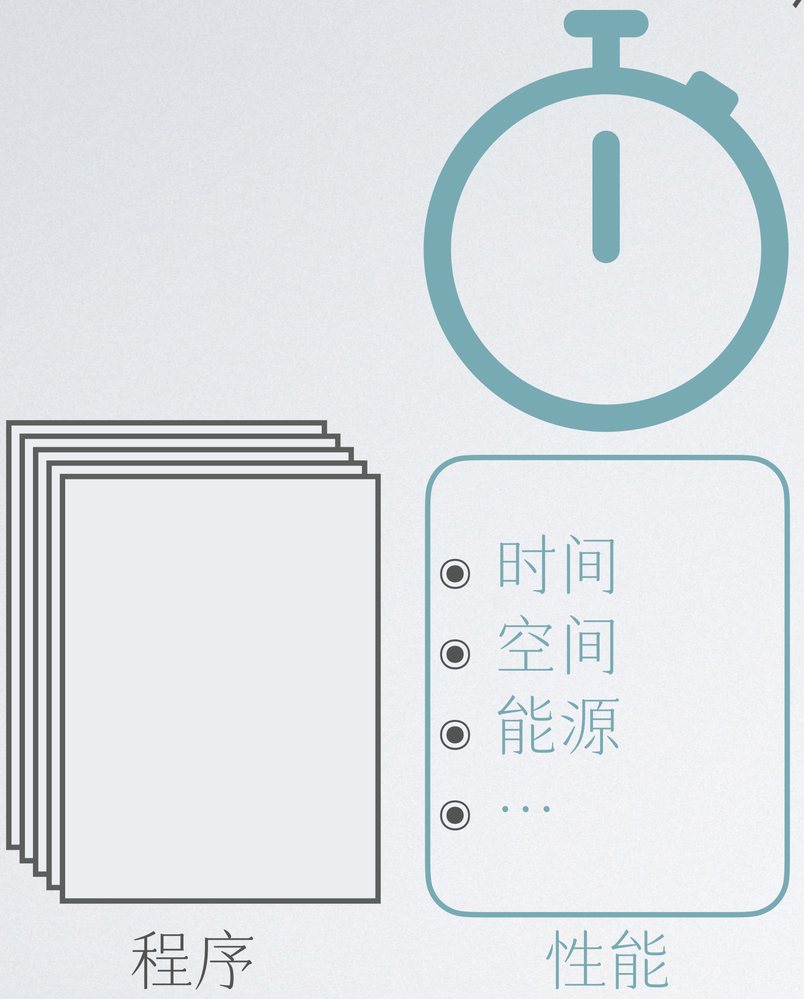
程序的资源消耗



应用场景

- 分析软件的性能瓶颈

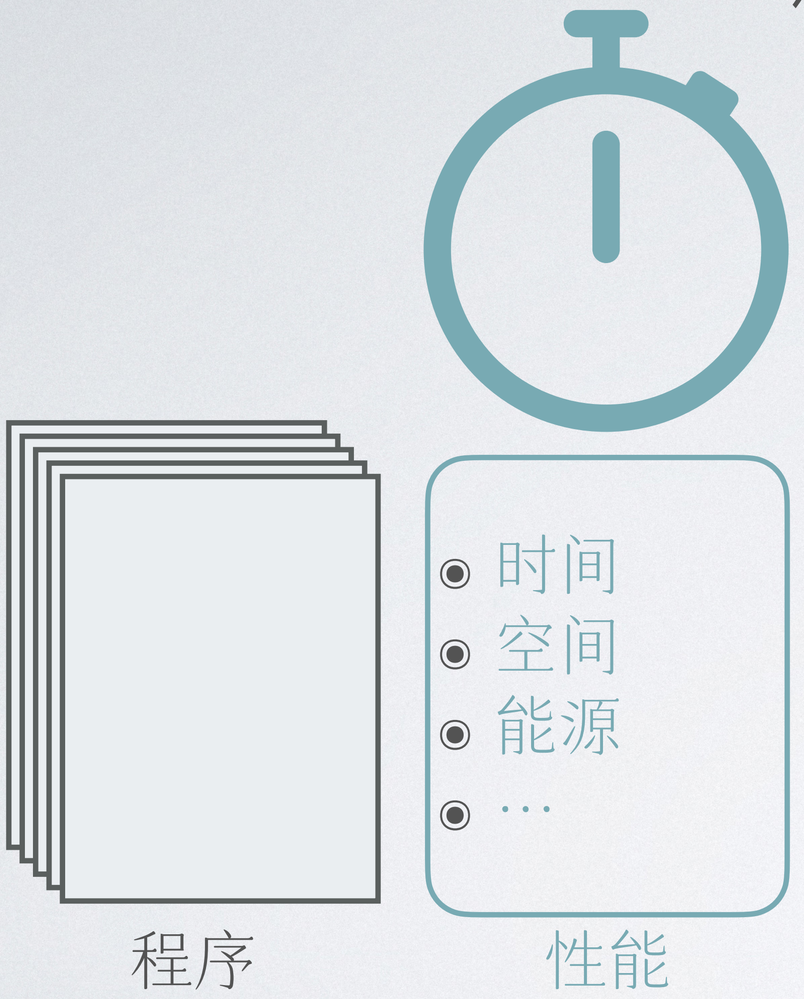
程序的资源消耗



应用场景

- 分析软件的性能瓶颈
- 预测智能合约的 gas 消耗

程序的资源消耗



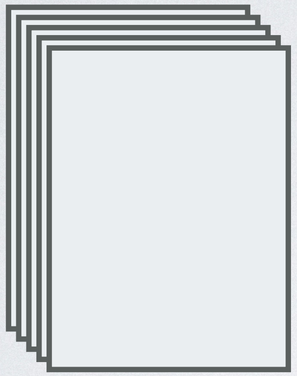
应用场景

- 分析软件的性能瓶颈
- 预测智能合约的 gas 消耗
- 发现潜在的算法复杂性攻击（如 DoS）和侧信道攻击隐患

提纲

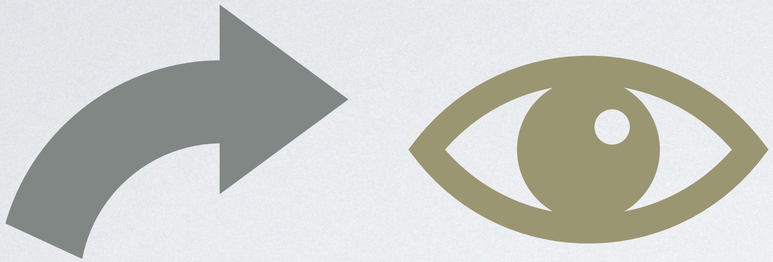
- [x] 目标阐述
- [ ] 静态分析：基于势能方法的均摊分析
- [ ] 动态分析：最坏情况测试生成
- [ ] 程序验证：量化霍尔逻辑

静态资源分析

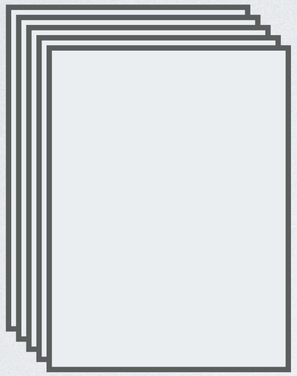


软件开发

静态资源分析

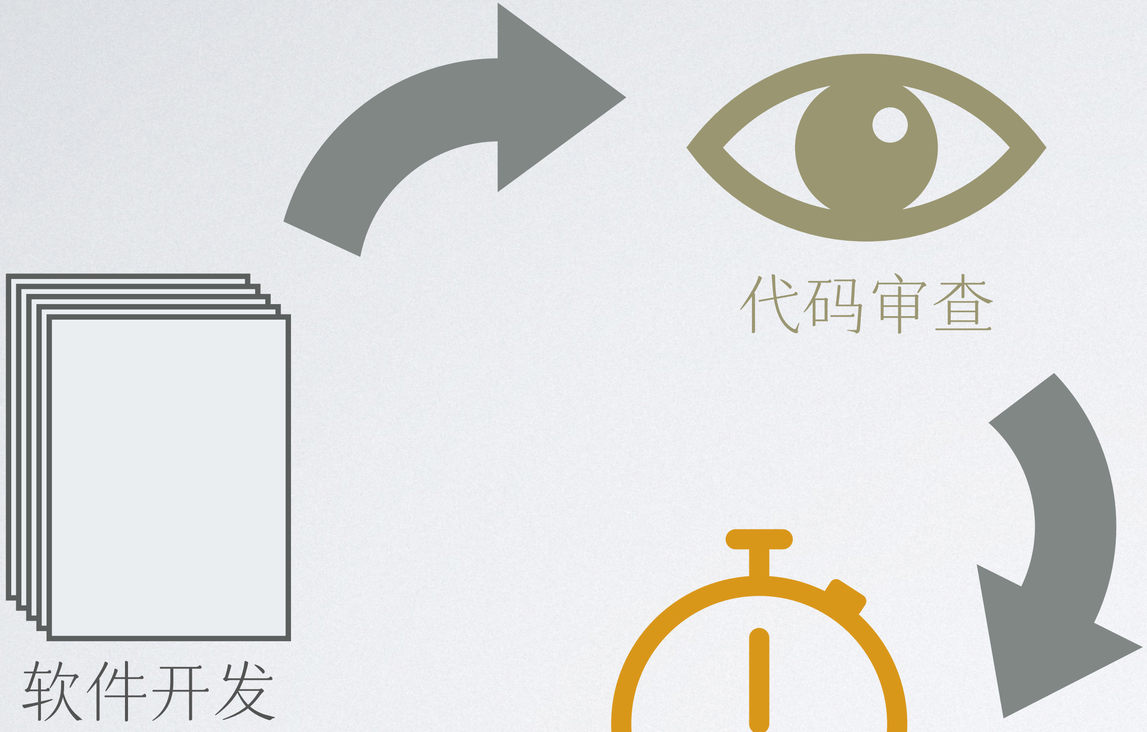


代码审查

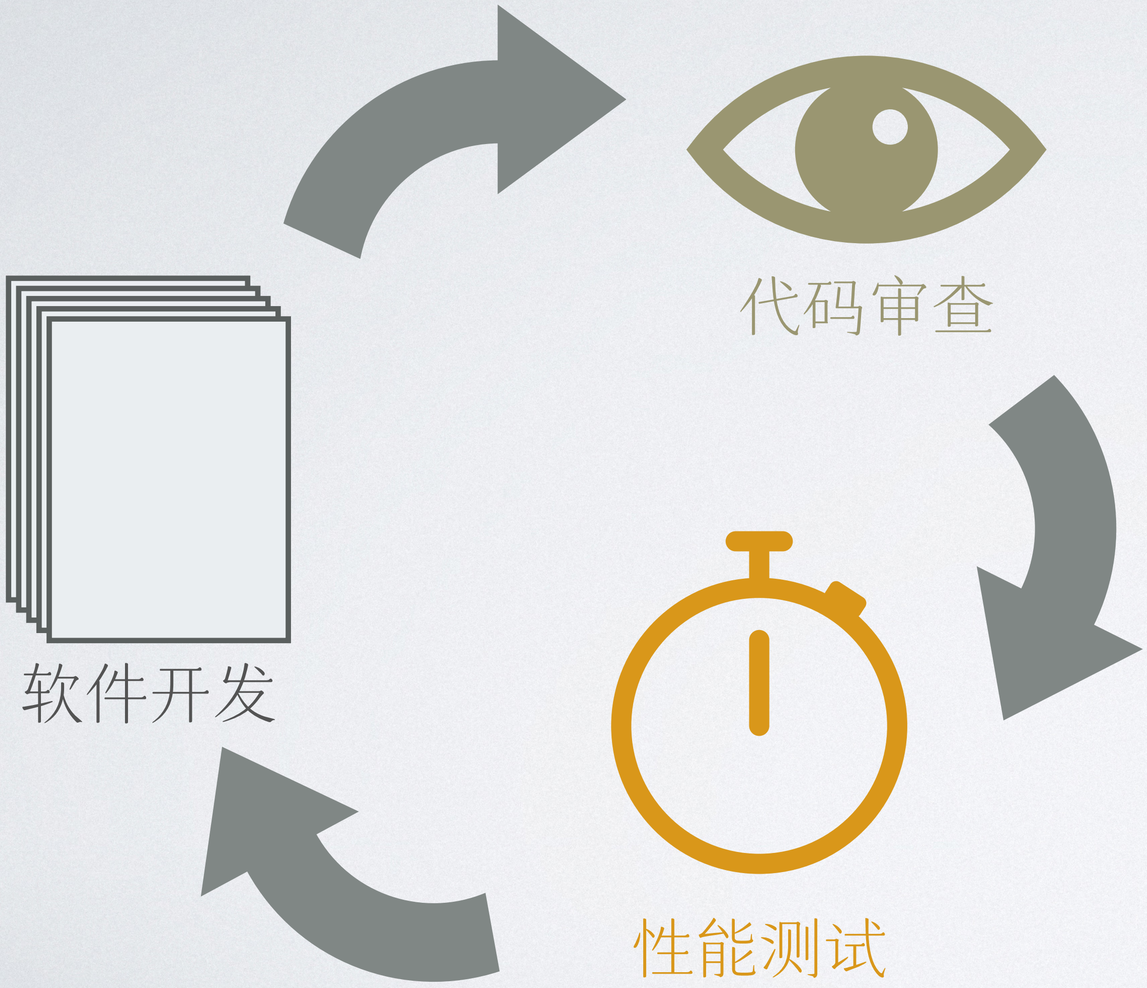


软件开发

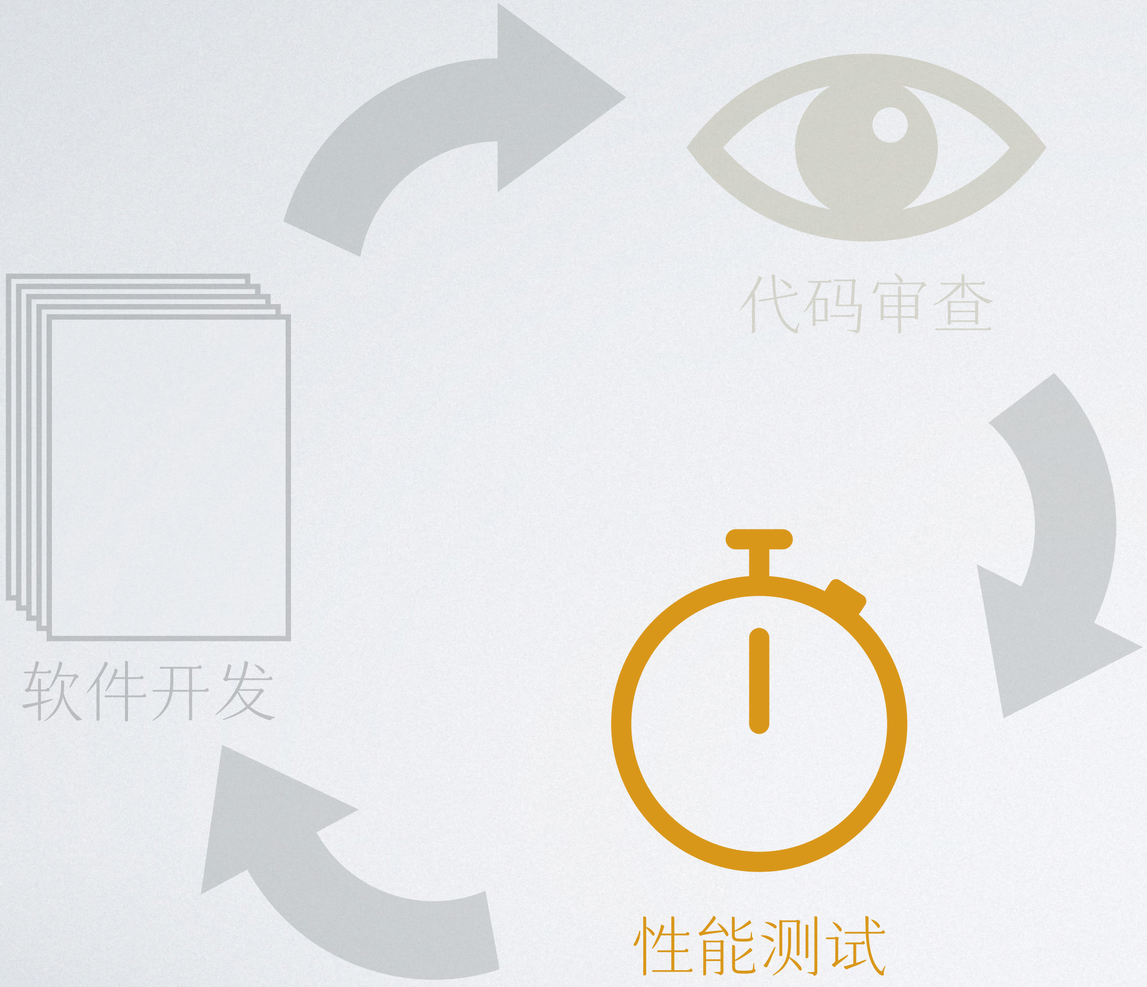
静态资源分析



静态资源分析



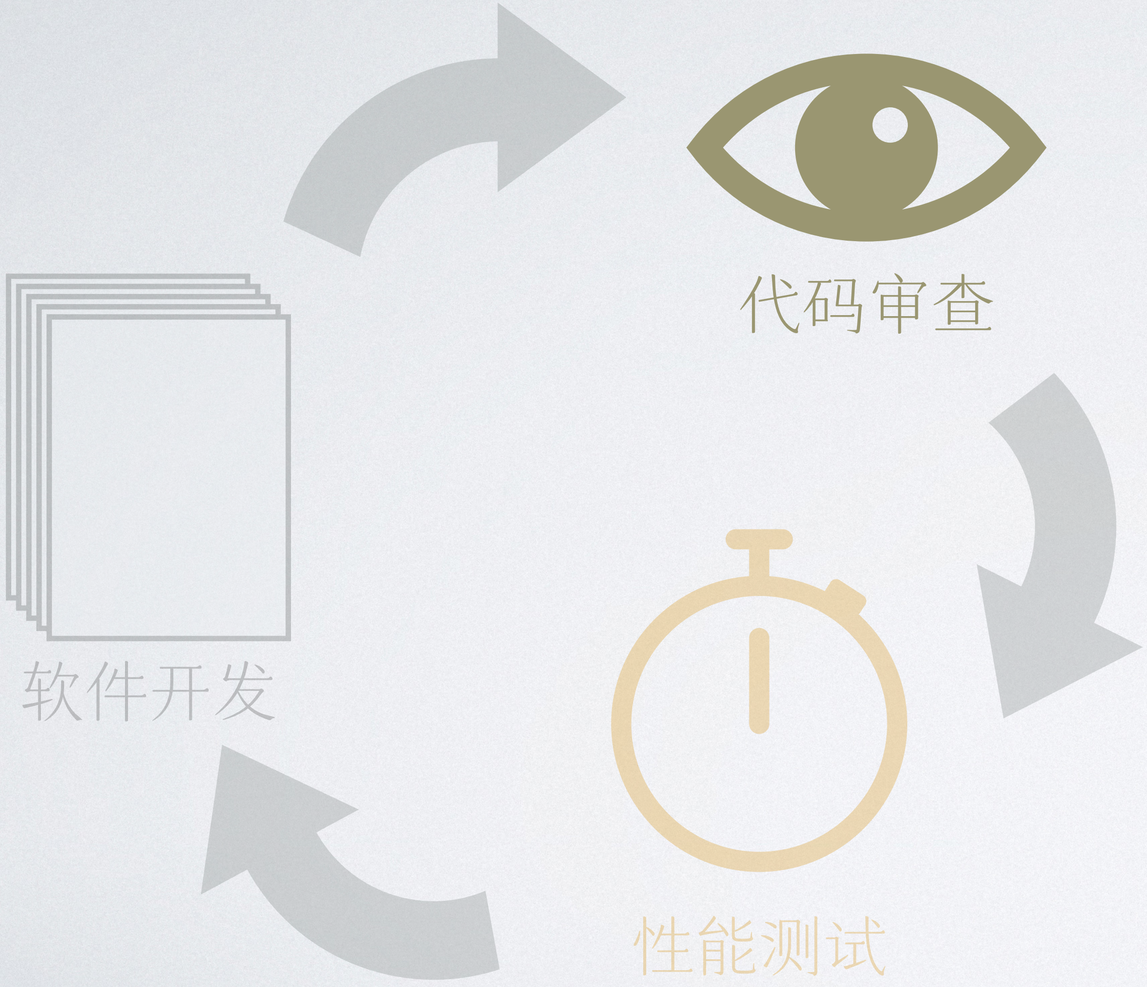
静态资源分析



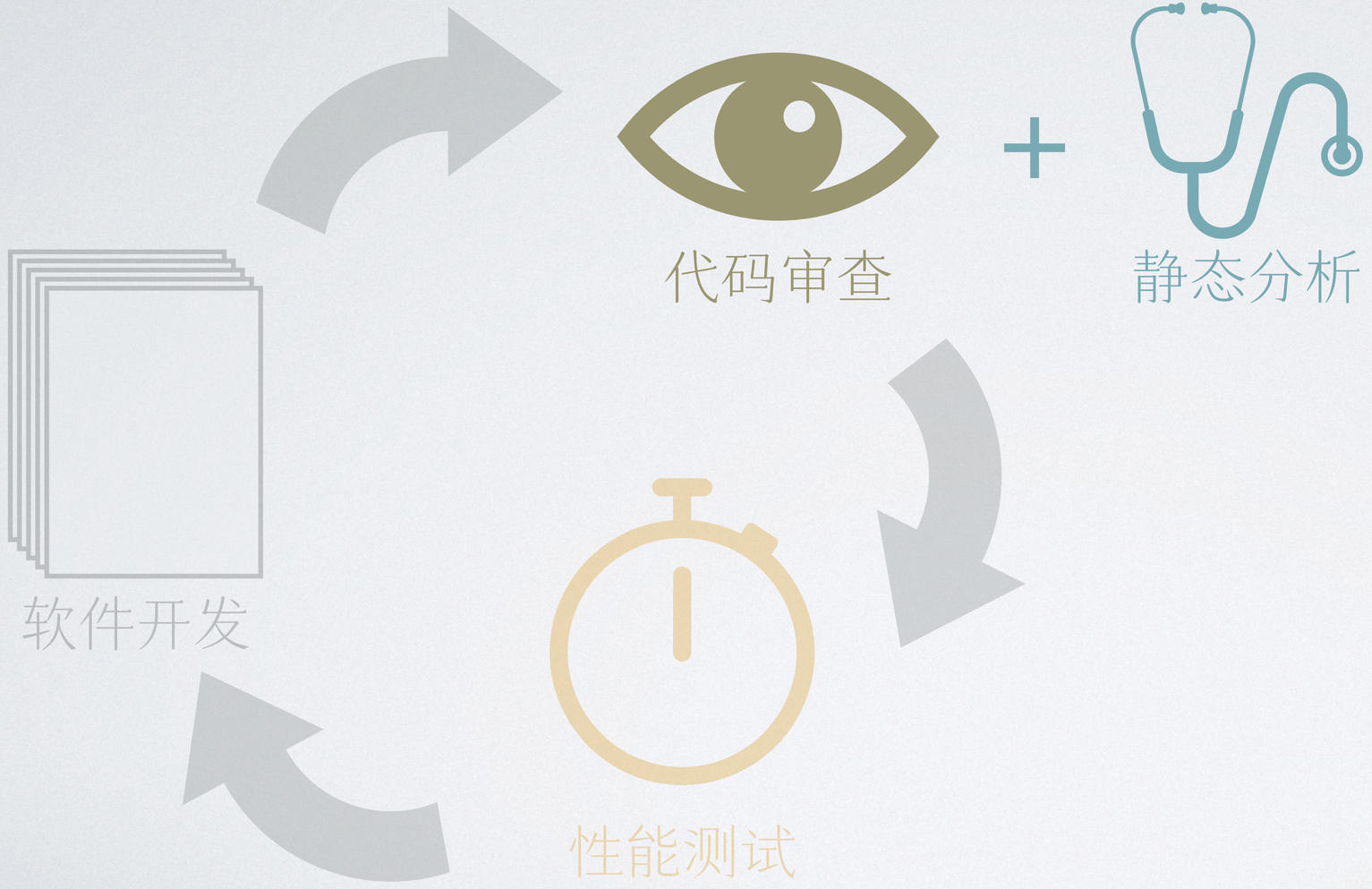
可能存在的缺点：

- 测试覆盖率不全
- 测试会进行较长的时间

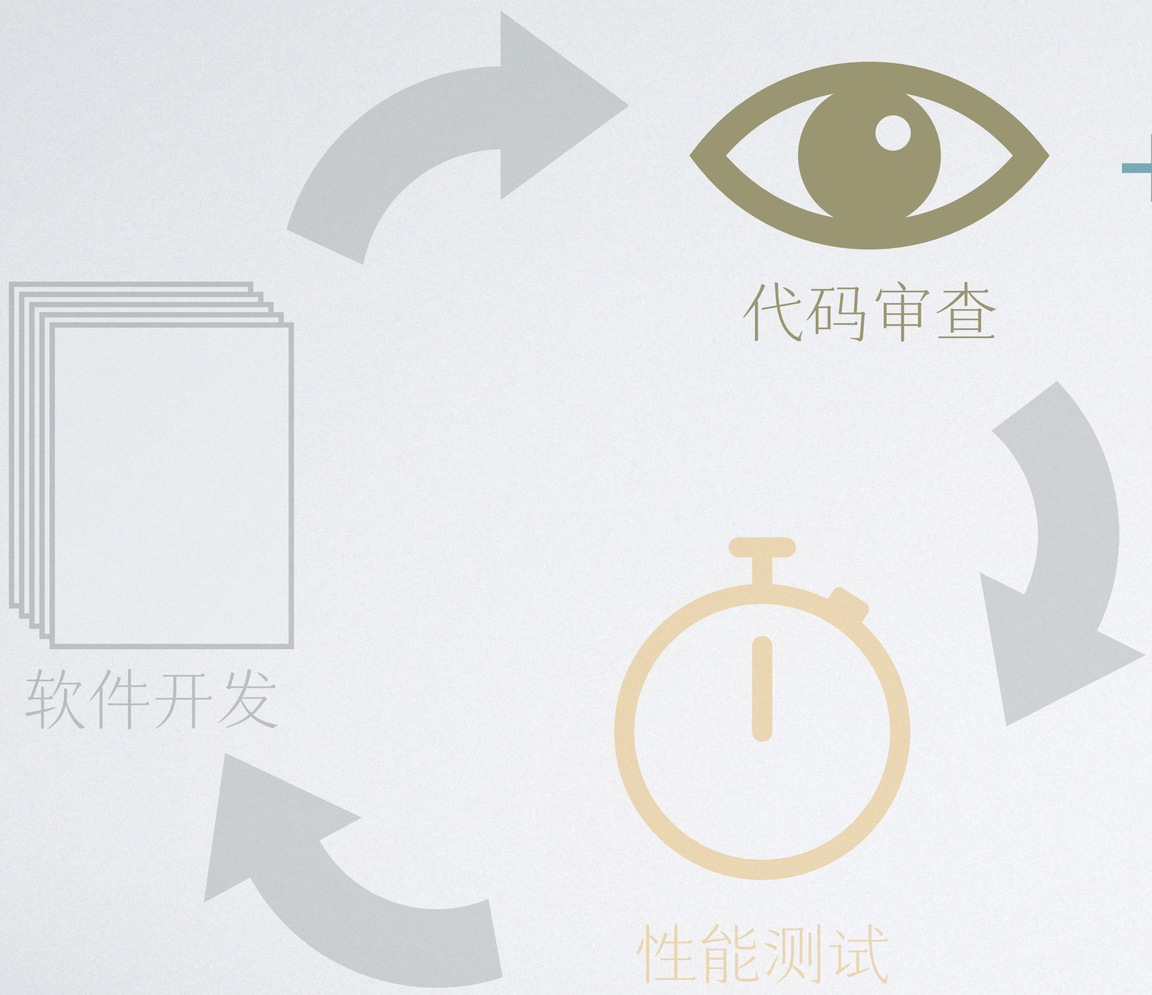
静态资源分析



静态资源分析



静态资源分析



静态分析的优点：

- 分析结果考虑了所有可能的输入情况
- 可以通过模块化提高分析效率

现实案例：Infer 中的静态资源分析

该案例来源于 Infer 的官方文档：https://fbinfer.com/docs/next/checker-cost/

现实案例：Infer 中的静态资源分析

该案例来源于 Infer 的官方文档：https://fbinfer.com/docs/next/checker-cost/

现实案例：Infer 中的静态资源分析

```
void loop(ArrayList<Integer> list) {
  for (int i = 0; i <= list.size(); i++) {
  }
}
```

该案例来源于 Infer 的官方文档：https://fbinfer.com/docs/next/checker-cost/

现实案例：Infer 中的静态资源分析

```
void loop(ArrayList<Integer> list) {
  for (int i = 0; i <= list.size(); i++) {
  }
}
```

$$8|list| + 16 = O(|list|)$$

该案例来源于 Infer 的官方文档：https://fbinfer.com/docs/next/checker-cost/

现实案例：Infer 中的静态资源分析

```
void loop(ArrayList<Integer> list) {
  for (int i = 0; i <= list.size(); i++) {
  }
}
```

$$8|list| + 16 = O(|list|)$$

```
void loop(ArrayList<Integer> list) {
  for (int i = 0; i <= list.size(); i++) {
    print(list); // new function call
  }
}
```

该案例来源于 Infer 的官方文档：https://fbinfer.com/docs/next/checker-cost/

现实案例：Infer 中的静态资源分析

```
void loop(ArrayList<Integer> list) {
  for (int i = 0; i <= list.size(); i++) {
  }
}
```

$8|list| + 16 = O(|list|)$

```
void loop(ArrayList<Integer> list) {
  for (int i = 0; i <= list.size(); i++) {
    print(list); // new function call
  }
}
```

$O(|list|^2)$

该案例来源于 Infer 的官方文档：https://fbinfer.com/docs/next/checker-cost/

现实案例：Infer 中的静态资源分析

```
void loop(ArrayList<Integer> list) {
  for (int i = 0; i <= list.size(); i++) {
  }
}
```

```
void loop(ArrayList<Integer> list) {
  for (int i = 0; i <= list.size(); i++) {
    print(list); // new function call
  }
}
```

$$8|list| + 16 = O(|list|)$$

时间复杂度变高了！

$$O(|list|^2)$$

该案例来源于 Infer 的官方文档：https://fbinfer.com/docs/next/checker-cost/

基于类型的资源分析

OCaml

```
let rec append l1 l2 =
  match l1 with
  | [] -> l2
  | x::xs -> x::(append xs l2)
```

RaML

[HDW17] J. Hoffmann, A. Das, and S.-C. Weng. 2017. Towards Automatic Resource Bound Analysis for OCaml. In *POPL'17*.

基于类型的资源分析

OCaml

```
let rec append l1 l2 =
  match l1 with
  | [] -> l2
  | x::xs -> x::(append xs l2)
```

RaML

append : $\langle L^9(\alpha) \times L^0(\alpha), 3 \rangle \to \langle L^0(\alpha), 0 \rangle$

[HDW17] J. Hoffmann, A. Das, and S.-C. Weng. 2017. Towards Automatic Resource Bound Analysis for OCaml. In *POPL'17*.

基于类型的资源分析

OCaml

```
let rec append l1 l2 =
  match l1 with
  | [] -> l2
  | x::xs -> x::(append xs l2)
```

[HDW17] J. Hoffmann, A. Das, and S.-C. Weng. 2017. Towards Automatic Resource Bound Analysis for OCaml. In *POPL'17*.

基于类型的资源分析

OCaml

```
let rec append l1 l2 =
  match l1 with
  | [] -> l2
  | x::xs -> x::(append xs l2)
```

RaML

append : 带有资源消耗信息的类型

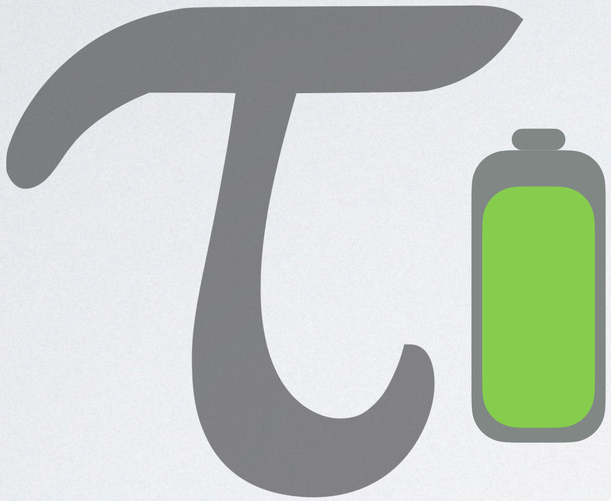
简化后可得到资源消耗的上界：

$$9|\ell_1| + 3 = O(|\ell_1|)$$

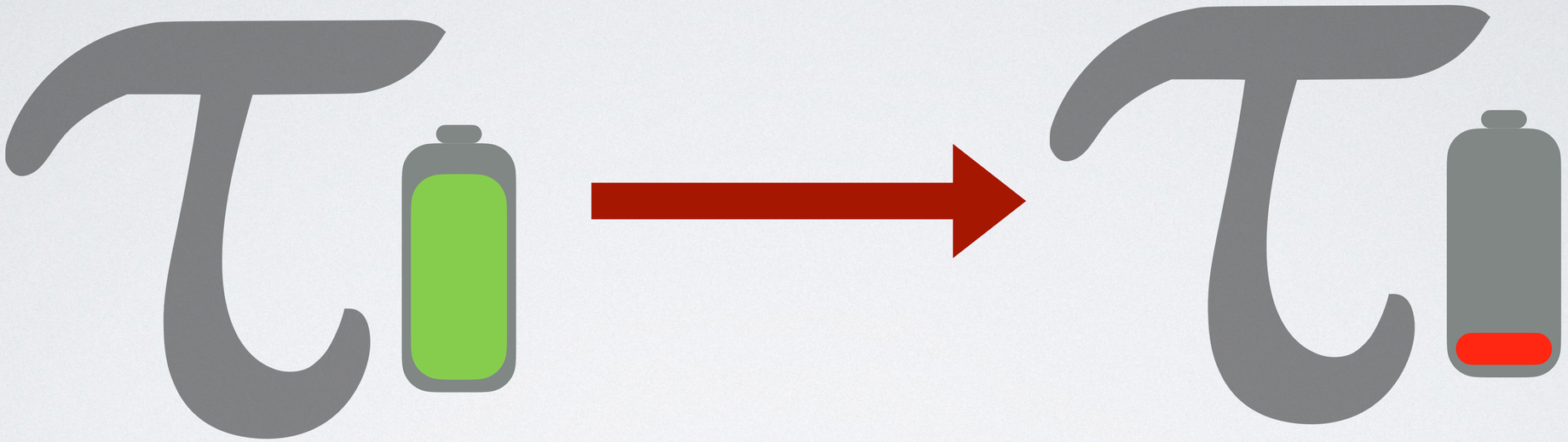
[HDW17] J. Hoffmann, A. Das, and S.-C. Weng. 2017. Towards Automatic Resource Bound Analysis for OCaml. In *POPL'17*.

基于类型的资源分析

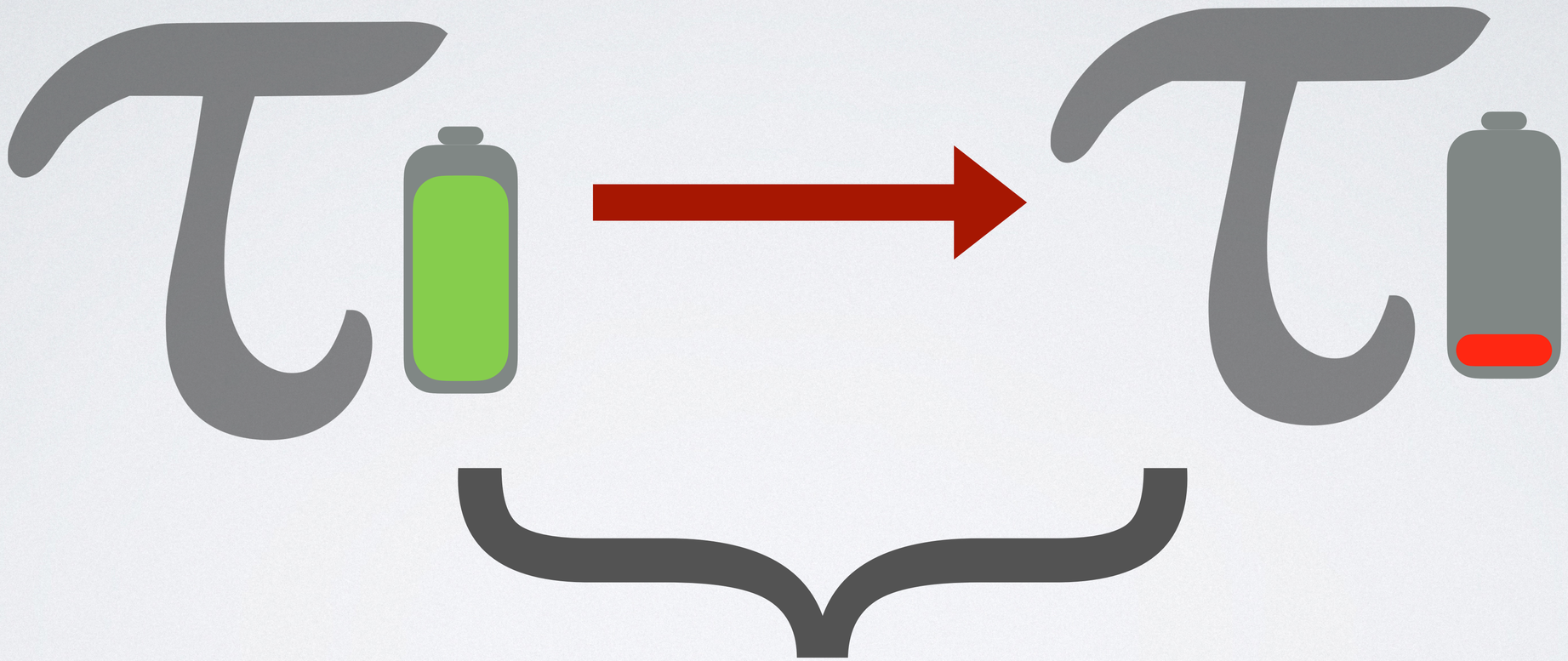
基于类型的资源分析



基于类型的资源分析



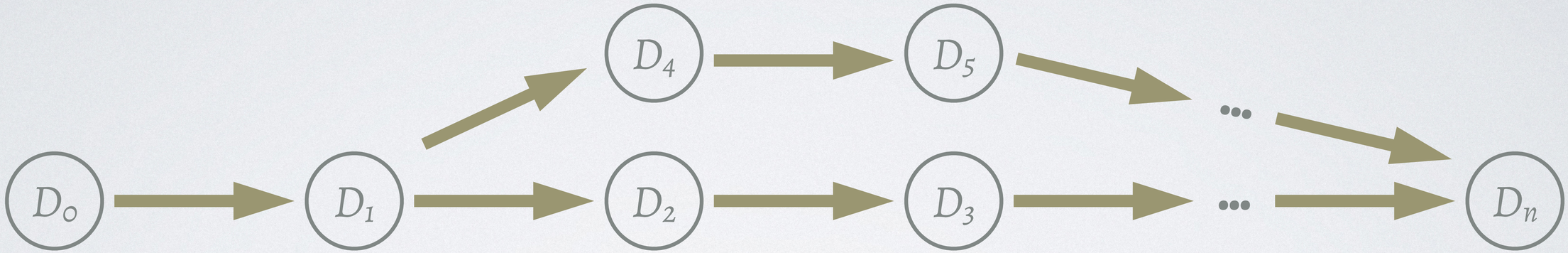
基于类型的资源分析



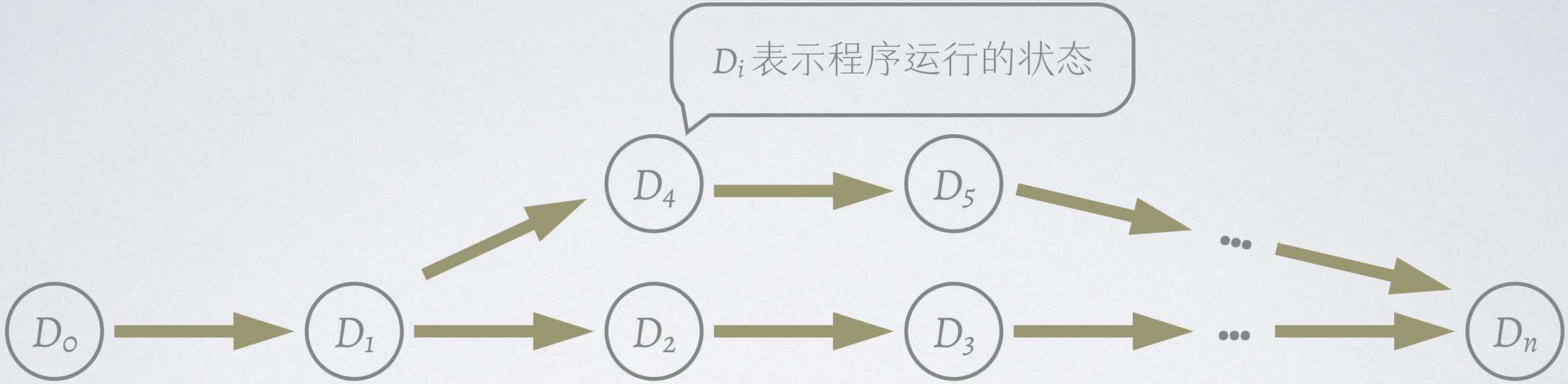
资源消耗

均摊分析的势能方法

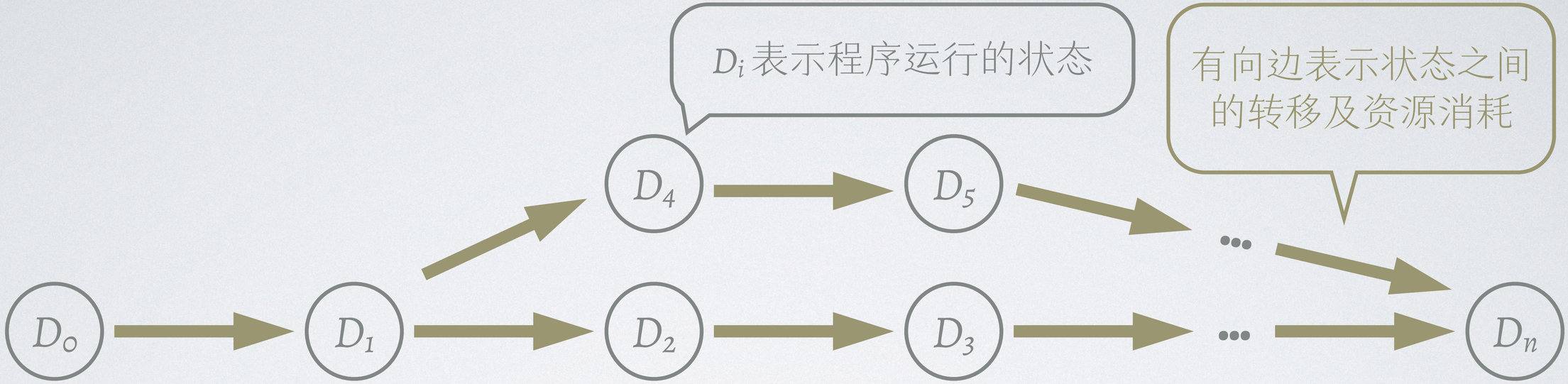
均摊分析的势能方法



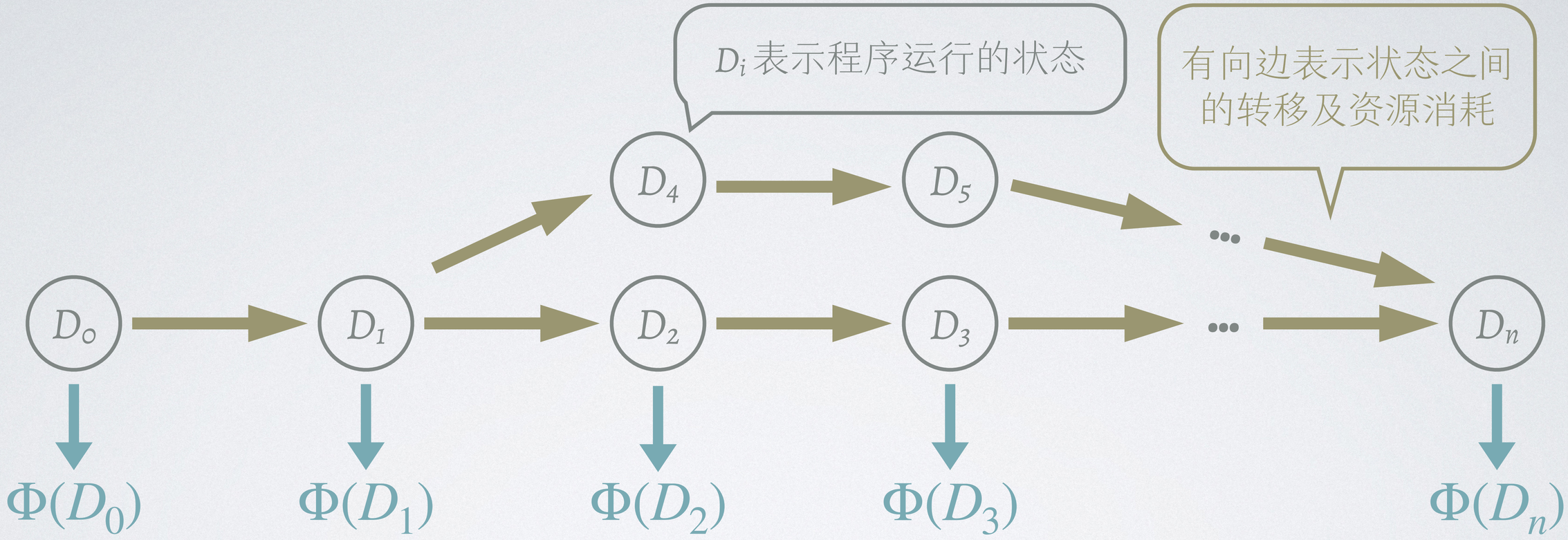
均摊分析的势能方法



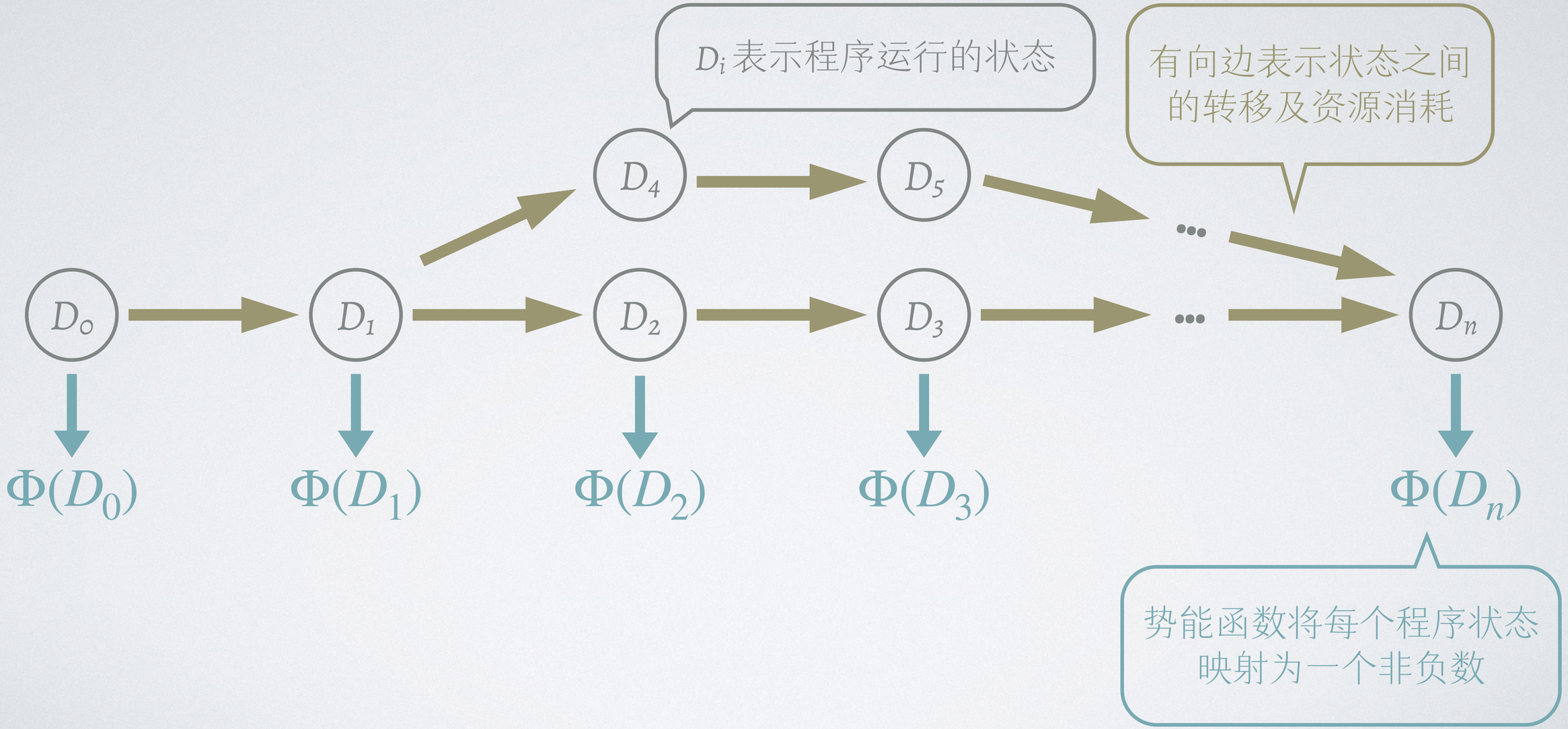
均摊分析的势能方法



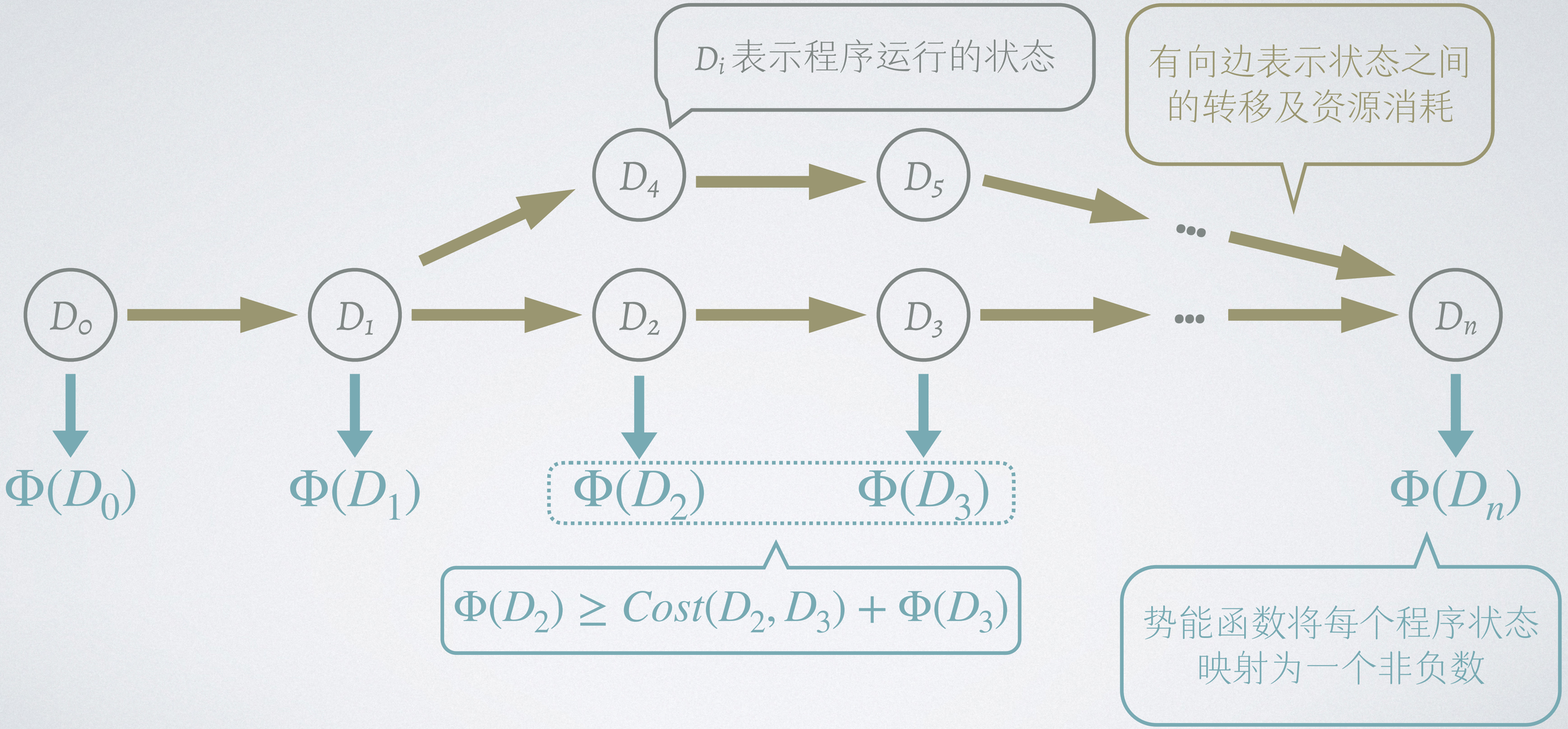
均摊分析的势能方法


D_i 表示程序运行的状态
有向边表示状态之间的转移及资源消耗
D_0
D_1
D_2
D_3
D_4
D_5
...
...
D_n
$\Phi(D_0)$
$\Phi(D_1)$
$\Phi(D_2)$
$\Phi(D_3)$
$\Phi(D_n)$

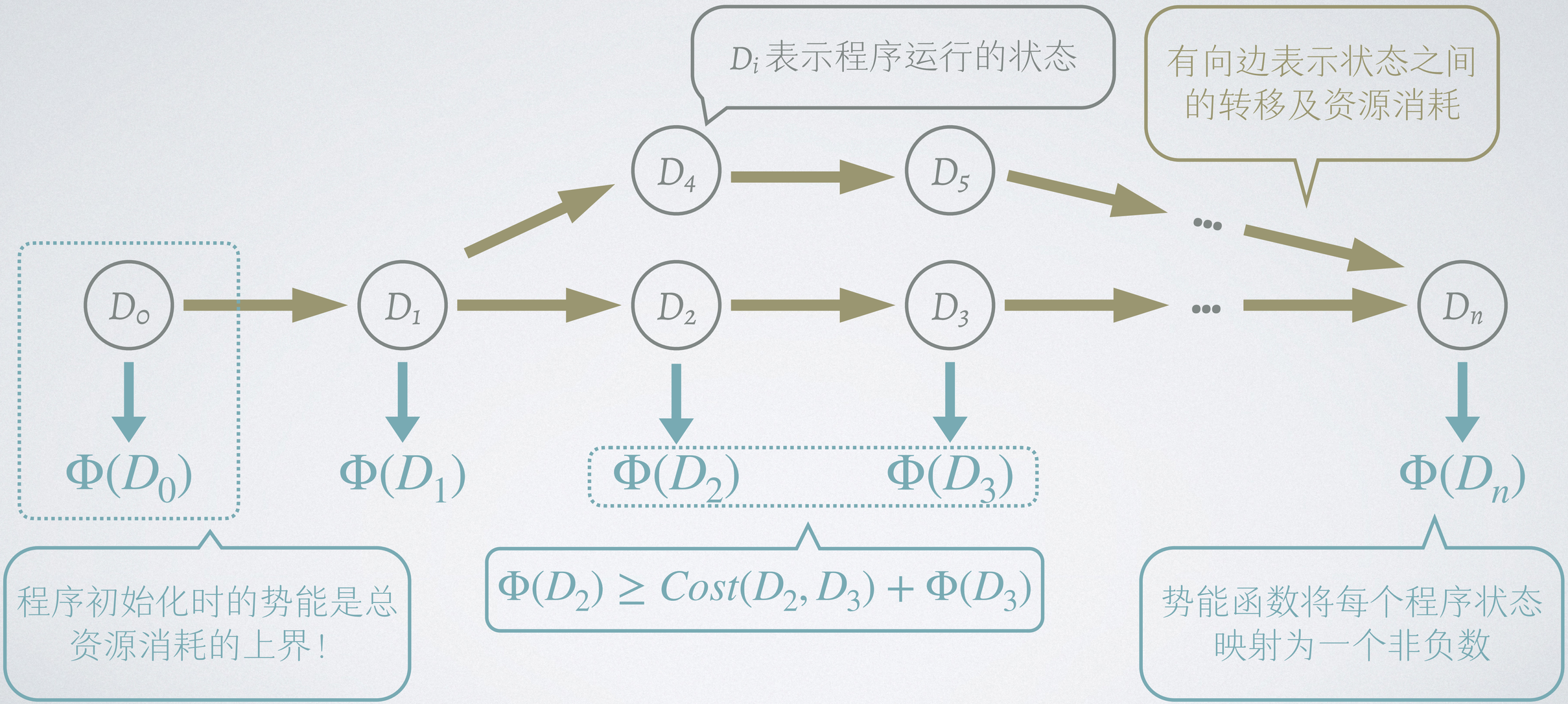
均摊分析的势能方法


D_i 表示程序运行的状态
有向边表示状态之间的转移及资源消耗
D_0
D_1
D_2
D_3
D_4
D_5
...
...
D_n
$\Phi(D_0)$
$\Phi(D_1)$
$\Phi(D_2)$
$\Phi(D_3)$
$\Phi(D_n)$
势能函数将每个程序状态映射为一个非负数

均摊分析的势能方法


D_i 表示程序运行的状态
有向边表示状态之间的转移及资源消耗
D_0
D_1
D_2
D_3
D_4
D_5
...
...
D_n
$\Phi(D_0)$
$\Phi(D_1)$
$\Phi(D_2)$
$\Phi(D_3)$
$\Phi(D_n)$
$\Phi(D_2) \geq Cost(D_2, D_3) + \Phi(D_3)$
势能函数将每个程序状态映射为一个非负数

均摊分析的势能方法


D_i 表示程序运行的状态
有向边表示状态之间的转移及资源消耗
D_0
D_1
D_2
D_3
D_4
D_5
...
...
D_n
$\Phi(D_0)$
$\Phi(D_1)$
$\Phi(D_2)$
$\Phi(D_3)$
$\Phi(D_n)$
程序初始化时的势能是总资源消耗的上界！
$\Phi(D_2) \geq Cost(D_2, D_3) + \Phi(D_3)$
势能函数将每个程序状态映射为一个非负数

带有势能标注的类型

```
let rec append l1 l2 =
  match l1 with
  | [] ->
    l2
  | x::xs ->
    let () = tick(1) in
    let rest = append xs l2 in
    x::rest
```

带有势能标注的类型

```
let rec append l1 l2 =
  match l1 with
  | [] ->
    l2
  | x::xs ->
    let () = tick(1) in
    let rest = append xs l2 in
    x::rest
```

通过 `tick` 显式标注程序的资源消耗模型

带有势能标注的类型

$Cost = |\ell_1|$

append : $\langle L^1(\alpha) \times L^0(\alpha), 0\rangle \to \langle L^0(\alpha), 0\rangle$

```
let rec append l1 l2 =
  match l1 with
  | [] ->
    l2
  | x::xs ->
    let () = tick(1) in
    let rest = append xs l2 in
    x::rest
```

通过 `tick` 显式标注程序的资源消耗模型

带有势能标注的类型

$Cost = |\ell_1|$

append : $\langle L^1(\alpha) \times L^0(\alpha), 0\rangle \to \langle L^0(\alpha), 0\rangle$

```
let rec append l1 l2 =
  match l1 with
  | [] ->
    l2
  | x::xs ->
    let () = tick(1) in
    let rest = append xs l2 in
    x::rest
```

通过 `tick` 显式标注程序的资源消耗模型

`L^p(a)`

列表中的每个元素都携带了 `p` 单位的势能

带有势能标注的类型

$Cost = |\ell_1|$

append : $\langle L^1(\alpha) \times L^0(\alpha), 0\rangle \to \langle L^0(\alpha), 0\rangle$

```
let rec append l1 l2 =
  match l1 with
  | [] ->
    l2
  | x::xs ->
    let () = tick(1) in
    let rest = append xs l2 in
    x::rest
```

通过 `tick` 显式标注程序的资源消耗模型

`L^p(a)`

列表中的每个元素都携带了 `p` 单位的势能

带有势能标注的类型

$Cost = |\ell_1|$

append : $\langle L^1(\alpha) \times L^0(\alpha), 0\rangle \to \langle L^0(\alpha), 0\rangle$

```
let rec append l1 l2 =
  match l1 with
  | [] ->
    l2
  | x::xs ->
    let () = tick(1) in
    let rest = append xs l2 in
    x::rest
```

通过 `tick` 显式标注程序的资源消耗模型

`L^p(a)`

列表中的每个元素都携带了 `p` 单位的势能

带有势能标注的类型

$Cost = |\ell_1|$

append : $\langle L^1(\alpha) \times L^0(\alpha), 0\rangle \to \langle L^0(\alpha), 0\rangle$

```
let rec append l1 l2 =
  match l1 with
  | [] ->
    l2
  | x::xs ->
    let () = tick(1) in
    let rest = append xs l2 in
    x::rest
```

通过 `tick` 显式标注程序的资源消耗模型

`Lᵖ(a)`

列表中的每个元素都携带了 `p` 单位的势能

`[l1: L¹(a), l2: L⁰(a)]; 0 units`

带有势能标注的类型

$Cost = |\ell_1|$

append : $\langle L^1(\alpha) \times L^0(\alpha), 0\rangle \to \langle L^0(\alpha), 0\rangle$

```
let rec append l1 l2 =
  match l1 with
  | [] ->
    l2
  | x::xs ->
    let () = tick(1) in
    let rest = append xs l2 in
    x::rest
```

通过 `tick` 显式标注程序的资源消耗模型

`Lᵖ(a)`

列表中的每个元素都携带了 `p` 单位的势能

```
[l1: L¹(a), l2: L⁰(a)]; 0 units
// l1 被消耗
```

带有势能标注的类型

$Cost = |\ell_1|$

append : $\langle L^1(\alpha) \times L^0(\alpha), 0\rangle \to \langle L^0(\alpha), 0\rangle$

```
let rec append l1 l2 =
  match l1 with
  | [] ->
    l2
  | x::xs ->
    let () = tick(1) in
    let rest = append xs l2 in
    x::rest
```

通过 `tick` 显式标注程序的资源消耗模型

`Lᵖ(a)`

列表中的每个元素都携带了 `p` 单位的势能

```
[l1: L¹(a), l2: L⁰(a)]; 0 units
// l1 被消耗
[l2: L⁰(a)]; 0 units
```

带有势能标注的类型

$Cost = |\ell_1|$

append : $\langle L^1(\alpha) \times L^0(\alpha), 0\rangle \to \langle L^0(\alpha), 0\rangle$

```
let rec append l1 l2 =
  match l1 with
  | [] ->
    l2
  | x::xs ->
    let () = tick(1) in
    let rest = append xs l2 in
    x::rest
```

通过 `tick` 显式标注程序的资源消耗模型

`Lᵖ(a)`

列表中的每个元素都携带了 `p` 单位的势能

```
[l1: L¹(a), l2: L⁰(a)]; 0 units
// l1 被消耗
[l2: L⁰(a)]; 0 units
// l2 被消耗且返回类型符合签名
```

带有势能标注的类型

$Cost = |\ell_1|$

append : $\langle L^1(\alpha) \times L^0(\alpha), 0\rangle \to \langle L^0(\alpha), 0\rangle$

```
let rec append l1 l2 =
  match l1 with
  | [] ->
    l2
  | x::xs ->
    let () = tick(1) in
    let rest = append xs l2 in
    x::rest
```

通过 `tick` 显式标注程序的资源消耗模型

`Lᵖ(a)`

列表中的每个元素都携带了 `p` 单位的势能

```
[l1: L¹(a), l2: L⁰(a)]; 0 units
// l1 被消耗
[l2: L⁰(a)]; 0 units
// l2 被消耗且返回类型符合签名
[l2: L⁰(a), x: a, xs: L¹(a)]; 1 unit
```

带有势能标注的类型

$Cost = |\ell_1|$

append : $\langle L^1(\alpha) \times L^0(\alpha), 0\rangle \to \langle L^0(\alpha), 0\rangle$

```
let rec append l1 l2 =
  match l1 with
  | [] ->
    l2
  | x::xs ->
    let () = tick(1) in
    let rest = append xs l2 in
    x::rest
```

通过 `tick` 显式标注程序的资源消耗模型

`Lᵖ(a)`

列表中的每个元素都携带了 `p` 单位的势能

```
[l1: L¹(a), l2: L⁰(a)]; 0 units
// l1 被消耗
[l2: L⁰(a)]; 0 units
// l2 被消耗且返回类型符合签名
[l2: L⁰(a), x: a, xs: L¹(a)]; 1 unit
[l2: L⁰(a), x: a, xs: L¹(a)]; 0 units
```

带有势能标注的类型

$Cost = |\ell_1|$

append : $\langle L^1(\alpha) \times L^0(\alpha), 0\rangle \to \langle L^0(\alpha), 0\rangle$

```
let rec append l1 l2 =
  match l1 with
  | [] ->
    l2
  | x::xs ->
    let () = tick(1) in
    let rest = append xs l2 in
    x::rest
```

通过 `tick` 显式标注程序的资源消耗模型

`Lᵖ(a)`

列表中的每个元素都携带了 `p` 单位的势能

```
[l1: L¹(a), l2: L⁰(a)]; 0 units
// l1 被消耗
[l2: L⁰(a)]; 0 units
// l2 被消耗且返回类型符合签名
[l2: L⁰(a), x: a, xs: L¹(a)]; 1 unit
[l2: L⁰(a), x: a, xs: L¹(a)]; 0 units
[x: a, rest: L⁰(a)]; 0 units
```

带有势能标注的类型

$Cost = |\ell_1|$

append : $\langle L^1(\alpha) \times L^0(\alpha), 0\rangle \to \langle L^0(\alpha), 0\rangle$

```
let rec append l1 l2 =
  match l1 with
  | [] ->
    l2
  | x::xs ->
    let () = tick(1) in
    let rest = append xs l2 in
    x::rest
```

通过 `tick` 显式标注程序的资源消耗模型

`Lᵖ(a)`

列表中的每个元素都携带了 `p` 单位的势能

```
[l1: L¹(a), l2: L⁰(a)]; 0 units
// l1 被消耗
[l2: L⁰(a)]; 0 units
// l2 被消耗且返回类型符合签名
[l2: L⁰(a), x: a, xs: L¹(a)]; 1 unit
[l2: L⁰(a), x: a, xs: L¹(a)]; 0 units
[x: a, rest: L⁰(a)]; 0 units
// x 和 rest 被消耗且返回类型符合签名
```

带有势能标注的类型

$Cost = |\ell_1|$

append : $\langle L^1(\alpha) \times L^0(\alpha), 0\rangle \to \langle L^0(\alpha), 0\rangle$

通过 `tick` 显式标注程序的资源消耗模型

```
let rec append l1 l2 =
  match l1 with
  | [] ->
    l2
  | x::xs ->
    let () = tick(1) in
    let rest = append xs l2 in
    x::rest
```

`Lᵖ(a)`

列表中的每个元素都携带了 `p` 单位的势能

```
[l1: L¹(a), l2: L⁰(a)]; 0 units
// l1 被消耗
[l2: L⁰(a)]; 0 units
// l2 被消耗且返回类型符合签名
[l2: L⁰(a), x: a, xs: L¹(a)]; 1 unit
[l2: L⁰(a), x: a, xs: L¹(a)]; 0 units
[x: a, rest: L⁰(a)]; 0 units
// x 和 rest 被消耗且返回类型符合签名
```

原理：每个程序点的**势能函数**由程序操作的数据结构的**静态类型标注**所决定

基于线性规划的类型推导

```
let rec append l1 l2 =
  match l1 with
  | [] ->
    l2
  | x::xs ->
    let () = tick(1) in
    let rest = append xs l2 in
    x::rest
```

基于线性规划的类型推导

append : $\langle L^p(\alpha) \times L^q(\alpha), r \rangle \to \langle L^s(\alpha), t \rangle$

```
let rec append l1 l2 =
  match l1 with
  | [] ->
    l2
  | x::xs ->
    let () = tick(1) in
    let rest = append xs l2 in
    x::rest
```

基于线性规划的类型推导

p,q,r,s,t 是未知数值量

append : $\langle L^p(\alpha) \times L^q(\alpha), r\rangle \to \langle L^s(\alpha), t\rangle$

```
let rec append l1 l2 =
  match l1 with
  | [] ->
    l2
  | x::xs ->
    let () = tick(1) in
    let rest = append xs l2 in
    x::rest
```

基于线性规划的类型推导

p,q,r,s,t 是未知数值量

append : $\langle L^p(\alpha) \times L^q(\alpha), r\rangle \to \langle L^s(\alpha), t\rangle$

```
let rec append l1 l2 =
  match l1 with
  | [] ->
    l2
  | x::xs ->
    let () = tick(1) in
    let rest = append xs l2 in
    x::rest
```

线性约束

$p \geq 0, q \geq 0, r \geq 0, s \geq 0, t \geq 0$

基于线性规划的类型推导

p,q,r,s,t 是未知数值量

append : $\langle L^p(\alpha) \times L^q(\alpha), r\rangle \to \langle L^s(\alpha), t\rangle$

```
let rec append l1 l2 =
  match l1 with
  | [] ->
    l2
  | x::xs ->
    let () = tick(1) in
    let rest = append xs l2 in
    x::rest
```

```
[l1: Lᵖ(a), l2: Lᵠ(a)]; r units
```

线性约束

```
p≥0,q≥0,r≥0,s≥0,t≥0
```

基于线性规划的类型推导

p,q,r,s,t 是未知数值量

append : $\langle L^p(\alpha) \times L^q(\alpha), r\rangle \to \langle L^s(\alpha), t\rangle$

```
let rec append l1 l2 =
  match l1 with
  | [] ->
    l2
  | x::xs ->
    let () = tick(1) in
    let rest = append xs l2 in
    x::rest
```

```
[l1: Lᵖ(a), l2: Lᵠ(a)]; r units
// l1 被消耗
```

线性约束

```
p≥0,q≥0,r≥0,s≥0,t≥0
```

基于线性规划的类型推导

p,q,r,s,t 是未知数值量

append : $\langle L^p(\alpha) \times L^q(\alpha), r\rangle \to \langle L^s(\alpha), t\rangle$

```
let rec append l1 l2 =
  match l1 with
  | [] ->
    l2
  | x::xs ->
    let () = tick(1) in
    let rest = append xs l2 in
    x::rest
```

```
[l1: Lᵖ(a), l2: Lᵠ(a)]; r units
// l1 被消耗
[l2: Lᵠ(a)]; r units
```

线性约束

```
p≥0,q≥0,r≥0,s≥0,t≥0
```

基于线性规划的类型推导

p,q,r,s,t 是未知数值量

append : $\langle L^p(\alpha) \times L^q(\alpha), r\rangle \to \langle L^s(\alpha), t\rangle$

```
let rec append l1 l2 =
  match l1 with
  | [] ->
    l2
  | x::xs ->
    let () = tick(1) in
    let rest = append xs l2 in
    x::rest
```

```
[l1: Lᵖ(a), l2: Lᵠ(a)]; r units
// l1 被消耗
[l2: Lᵠ(a)]; r units
// l2 被消耗且返回类型符合签名
```

线性约束

```
p≥0,q≥0,r≥0,s≥0,t≥0
```

基于线性规划的类型推导

p,q,r,s,t 是未知数值量

append : $\langle L^p(\alpha) \times L^q(\alpha), r\rangle \to \langle L^s(\alpha), t\rangle$

```
let rec append l1 l2 =
  match l1 with
  | [] ->
    l2
  | x::xs ->
    let () = tick(1) in
    let rest = append xs l2 in
    x::rest
```

```
[l1: Lᵖ(a), l2: Lᵠ(a)]; r units
// l1 被消耗
[l2: Lᵠ(a)]; r units
// l2 被消耗且返回类型符合签名
```

线性约束

```
p≥0,q≥0,r≥0,s≥0,t≥0

q≥s,r≥t
```

基于线性规划的类型推导

p,q,r,s,t 是未知数值量

append : $\langle L^p(\alpha) \times L^q(\alpha), r\rangle \to \langle L^s(\alpha), t\rangle$

```
let rec append l1 l2 =
  match l1 with
  | [] ->
    l2
  | x::xs ->
    let () = tick(1) in
    let rest = append xs l2 in
    x::rest
```

```
[l1: Lᵖ(a), l2: Lᵠ(a)]; r units
// l1 被消耗
[l2: Lᵠ(a)]; r units
// l2 被消耗且返回类型符合签名
[l2: Lᵠ(a), x: a, xs: Lᵖ(a)]; r+p units
```

线性约束

```
p≥0,q≥0,r≥0,s≥0,t≥0

q≥s,r≥t
```

基于线性规划的类型推导

p,q,r,s,t 是未知数值量

append : $\langle L^p(\alpha) \times L^q(\alpha), r\rangle \to \langle L^s(\alpha), t\rangle$

```
let rec append l1 l2 =
  match l1 with
  | [] ->
    l2
  | x::xs ->
    let () = tick(1) in
    let rest = append xs l2 in
    x::rest
```

```
[l1: Lᵖ(a), l2: Lᵠ(a)]; r units
// l1 被消耗
[l2: Lᵠ(a)]; r units
// l2 被消耗且返回类型符合签名
[l2: Lᵠ(a), x: a, xs: Lᵖ(a)]; r+p units
[l2: Lᵠ(a), x: a, xs: Lᵖ(a)]; r+p-1 units
```

线性约束

```
p≥0,q≥0,r≥0,s≥0,t≥0

q≥s,r≥t
```

基于线性规划的类型推导

p,q,r,s,t 是未知数值量

append : $\langle L^p(\alpha) \times L^q(\alpha), r\rangle \to \langle L^s(\alpha), t\rangle$

```
let rec append l1 l2 =
  match l1 with
  | [] ->
    l2
  | x::xs ->
    let () = tick(1) in
    let rest = append xs l2 in
    x::rest
```

```
[l1: Lᵖ(a), l2: Lᵠ(a)]; r units
// l1 被消耗
[l2: Lᵠ(a)]; r units
// l2 被消耗且返回类型符合签名
[l2: Lᵠ(a), x: a, xs: Lᵖ(a)]; r+p units
[l2: Lᵠ(a), x: a, xs: Lᵖ(a)]; r+p-1 units
```

线性约束

```
p≥0,q≥0,r≥0,s≥0,t≥0

q≥s,r≥t

r+p-1≥0
```

基于线性规划的类型推导

p,q,r,s,t 是未知数值量

append : $\langle L^p(\alpha) \times L^q(\alpha), r\rangle \to \langle L^s(\alpha), t\rangle$

```
let rec append l1 l2 =
  match l1 with
  | [] ->
    l2
  | x::xs ->
    let () = tick(1) in
    let rest = append xs l2 in
    x::rest
```

```
[l1: Lᵖ(a), l2: Lᵠ(a)]; r units
// l1 被消耗
[l2: Lᵠ(a)]; r units
// l2 被消耗且返回类型符合签名
[l2: Lᵠ(a), x: a, xs: Lᵖ(a)]; r+p units
[l2: Lᵠ(a), x: a, xs: Lᵖ(a)]; r+p-1 units
[x: a, rest: Lˢ(a)]; p-1+t units
```

线性约束

```
p≥0,q≥0,r≥0,s≥0,t≥0

q≥s,r≥t

r+p-1≥0
```

基于线性规划的类型推导

p,q,r,s,t 是未知数值量

append : $\langle L^p(\alpha) \times L^q(\alpha), r\rangle \to \langle L^s(\alpha), t\rangle$

```
let rec append l1 l2 =
  match l1 with
  | [] ->
    l2
  | x::xs ->
    let () = tick(1) in
    let rest = append xs l2 in
    x::rest
```

```
[l1: Lᵖ(a), l2: Lᵠ(a)]; r units
// l1 被消耗
[l2: Lᵠ(a)]; r units
// l2 被消耗且返回类型符合签名
[l2: Lᵠ(a), x: a, xs: Lᵖ(a)]; r+p units
[l2: Lᵠ(a), x: a, xs: Lᵖ(a)]; r+p-1 units
[x: a, rest: Lˢ(a)]; p-1+t units
```

线性约束

```
p≥0,q≥0,r≥0,s≥0,t≥0

q≥s,r≥t

r+p-1≥0
p≥p,q≥q,r+p-1≥r
```

基于线性规划的类型推导

p,q,r,s,t 是未知数值量

append : $\langle L^p(\alpha) \times L^q(\alpha), r\rangle \to \langle L^s(\alpha), t\rangle$

```
let rec append l1 l2 =
  match l1 with
  | [] ->
    l2
  | x::xs ->
    let () = tick(1) in
    let rest = append xs l2 in
    x::rest
```

```
[l1: Lᵖ(a), l2: Lᵠ(a)]; r units
// l1 被消耗
[l2: Lᵠ(a)]; r units
// l2 被消耗且返回类型符合签名
[l2: Lᵠ(a), x: a, xs: Lᵖ(a)]; r+p units
[l2: Lᵠ(a), x: a, xs: Lᵖ(a)]; r+p-1 units
[x: a, rest: Lˢ(a)]; p-1+t units
// x 和 rest 被消耗且返回类型符合签名
```

线性约束

```
p≥0,q≥0,r≥0,s≥0,t≥0

q≥s,r≥t

r+p-1≥0
p≥p,q≥q,r+p-1≥r
```

基于线性规划的类型推导

p,q,r,s,t 是未知数值量

append : $\langle L^p(\alpha) \times L^q(\alpha), r\rangle \to \langle L^s(\alpha), t\rangle$

```
let rec append l1 l2 =
  match l1 with
  | [] ->
    l2
  | x::xs ->
    let () = tick(1) in
    let rest = append xs l2 in
    x::rest
```

```
[l1: Lᵖ(a), l2: Lᵠ(a)]; r units
// l1 被消耗
[l2: Lᵠ(a)]; r units
// l2 被消耗且返回类型符合签名
[l2: Lᵠ(a), x: a, xs: Lᵖ(a)]; r+p units
[l2: Lᵠ(a), x: a, xs: Lᵖ(a)]; r+p-1 units
[x: a, rest: Lˢ(a)]; p-1+t units
// x 和 rest 被消耗且返回类型符合签名
```

线性约束

```
p≥0,q≥0,r≥0,s≥0,t≥0

q≥s,r≥t

r+p-1≥0
p≥p,q≥q,r+p-1≥r
p-1+t≥s+t
```

基于线性规划的类型推导

p,q,r,s,t 是未知数值量

append : $\langle L^p(\alpha) \times L^q(\alpha), r\rangle \to \langle L^s(\alpha), t\rangle$

```
let rec append l1 l2 =
  match l1 with
  | [] ->
    l2
  | x::xs ->
    let () = tick(1) in
    let rest = append xs l2 in
    x::rest
```

```
[l1: Lᵖ(a), l2: Lᵠ(a)]; r units
// l1 被消耗
[l2: Lᵠ(a)]; r units
// l2 被消耗且返回类型符合签名
[l2: Lᵠ(a), x: a, xs: Lᵖ(a)]; r+p units
[l2: Lᵠ(a), x: a, xs: Lᵖ(a)]; r+p-1 units
[x: a, rest: Lˢ(a)]; p-1+t units
// x 和 rest 被消耗且返回类型符合签名
```

线性约束

```
p≥0,q≥0,r≥0,s≥0,t≥0

q≥s,r≥t

r+p-1≥0
p≥p,q≥q,r+p-1≥r
p-1+t≥s+t
```

p=1, q=r=s=t=0

基于线性规划的类型推导

p,q,r,s,t 是未知数值量

append : $\langle L^p(\alpha) \times L^q(\alpha), r\rangle \to \langle L^s(\alpha), t\rangle$

```
let rec append l1 l2 =
  match l1 with
  | [] ->
    l2
  | x::xs ->
    let () = tick(1) in
    let rest = append xs l2 in
    x::rest
```

```
[l1: Lᵖ(a), l2: Lᵠ(a)]; r units
// l1 被消耗
[l2: Lᵠ(a)]; r units
// l2 被消耗且返回类型符合签名
[l2: Lᵠ(a), x: a, xs: Lᵖ(a)]; r+p units
[l2: Lᵠ(a), x: a, xs: Lᵖ(a)]; r+p-1 units
[x: a, rest: Lˢ(a)]; p-1+t units
// x 和 rest 被消耗且返回类型符合签名
```

线性约束

```
p≥0,q≥0,r≥0,s≥0,t≥0

q≥s,r≥t

r+p-1≥0
p≥p,q≥q,r+p-1≥r
p-1+t≥s+t
```

append : $\langle L^1(\alpha) \times L^0(\alpha), 0\rangle \to \langle L^0(\alpha), 0\rangle$ ⟵ `p=1, q=r=s=t=0`

基于线性规划的类型推导

p,q,r,s,t 是未知数值量

append : $\langle L^p(\alpha) \times L^q(\alpha), r\rangle \to \langle L^s(\alpha), t\rangle$

```
let rec append l1 l2 =
  match l1 with
  | [] ->
    l2
  | x::xs ->
    let () = tick(1) in
    let rest = append xs l2 in
    x::rest
```

```
[l1: Lᵖ(a), l2: Lᵠ(a)]; r units
// l1 被消耗
[l2: Lᵠ(a)]; r units
// l2 被消耗且返回类型符合签名
[l2: Lᵠ(a), x: a, xs: Lᵖ(a)]; r+p units
[l2: Lᵠ(a), x: a, xs: Lᵖ(a)]; r+p-1 units
[x: a, rest: Lˢ(a)]; p-1+t units
// x 和 rest 被消耗且返回类型符合签名
```

线性约束

```
p≥0,q≥0,r≥0,s≥0,t≥0

q≥s,r≥t

r+p-1≥0
p≥p,q≥q,r+p-1≥r
p-1+t≥s+t
```

append : $\langle L^2(\alpha) \times L^1(\alpha), 3\rangle \to \langle L^1(\alpha), 3\rangle$ ← `p=2, q=s=1, r=t=3`

势能方法的研究现状

[HDW17]	多元多项式形式的资源消耗上界，均摊资源分析
[Atkey10]	命令式编程语言，支持堆操作
[JHL+10]	函数式编程语言，支持高阶函数
[HM18]	对数形式的资源消耗上界（可分析伸展树）
[KH20]	指数形式的资源消耗上界
[WKH20]	**我的研究：对概率程序的期望资源消耗分析**

[Atkey10] R. Atkey. 2010. Amortised Resource Analysis with Separation Logic. In *ESOP'10*.
[JHL+10] S. Jost, K. Hammond, H.-W. Loidl, and M. Hofmann. 2010. Static Determination of Quantitative Resource Usage for Higher-Order Programs. In *POPL'10*.
[HM18] M. Hofmann and G. Moser. 2018. Analysis of Logarithmic Amortised Complexity. Available on: https://arxiv.org/abs/1807.08242.
[KH20] D. M. Kahn and J. Hoffmann. 2020. Exponential Automatic Amortized Resource Analysis. In *FoSSaCS'20*.

静态资源分析的研究展望

静态资源分析的研究展望

学术界  工业界

静态资源分析的研究展望

分歧

学术界  工业界

“可以严格证明正确性的形式化方法”

静态资源分析的研究展望

分歧

学术界

“可以严格证明正确性的形式化方法”

工业界

“工具实用的话，可以没有严格的正确性”

静态资源分析的研究展望

分歧

学术界  工业界

“可以严格证明正确性的形式化方法”

“工具实用的话，可以没有严格的正确性”

适度放宽对正确性的要求可以引出很多的研究课题

静态资源分析的研究展望

学术界

“可以严格证明正确性的形式化方法”

工业界

“工具实用的话，可以没有严格的正确性”

适度放宽对正确性的要求可以引出很多的研究课题

工业级语言

静态资源分析的研究展望

“可以严格证明正确性的形式化方法”

“工具实用的话，可以没有严格的正确性”

适度放宽对正确性的要求可以引出很多的研究课题

工业级语言

利用 Rust 语言精确内存抽象
进行实用的资源分析

静态资源分析的研究展望

“可以严格证明正确性的形式化方法”

“工具实用的话，可以没有严格的正确性”

适度放宽对正确性的要求可以引出很多的研究课题

工业级语言

利用 Rust 语言精确内存抽象
进行实用的资源分析

数据驱动

静态资源分析的研究展望

“可以严格证明正确性的形式化方法”

“工具实用的话，可以没有严格的正确性”

适度放宽对正确性的要求可以引出很多的研究课题

工业级语言

利用 Rust 语言精确内存抽象
进行实用的资源分析

数据驱动

针对**输入**的经验分布的
平均情况资源分析

静态资源分析的研究展望

分歧

学术界 ⟷ 工业界

“可以严格证明正确性的形式化方法”

“工具实用的话，可以没有严格的正确性”

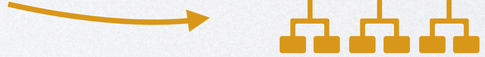
适度放宽对正确性的要求可以引出很多的研究课题

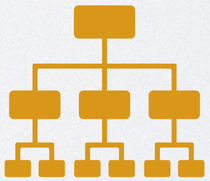
工业级语言

利用 Rust 语言精确内存抽象
进行实用的资源分析

数据驱动

针对**输入**的经验分布的
平均情况资源分析





网络请求

用户输入

...

静态资源分析的研究展望

“可以严格证明正确性的形式化方法”

“工具实用的话，可以没有严格的正确性”

适度放宽对正确性的要求可以引出很多的研究课题

工业级语言

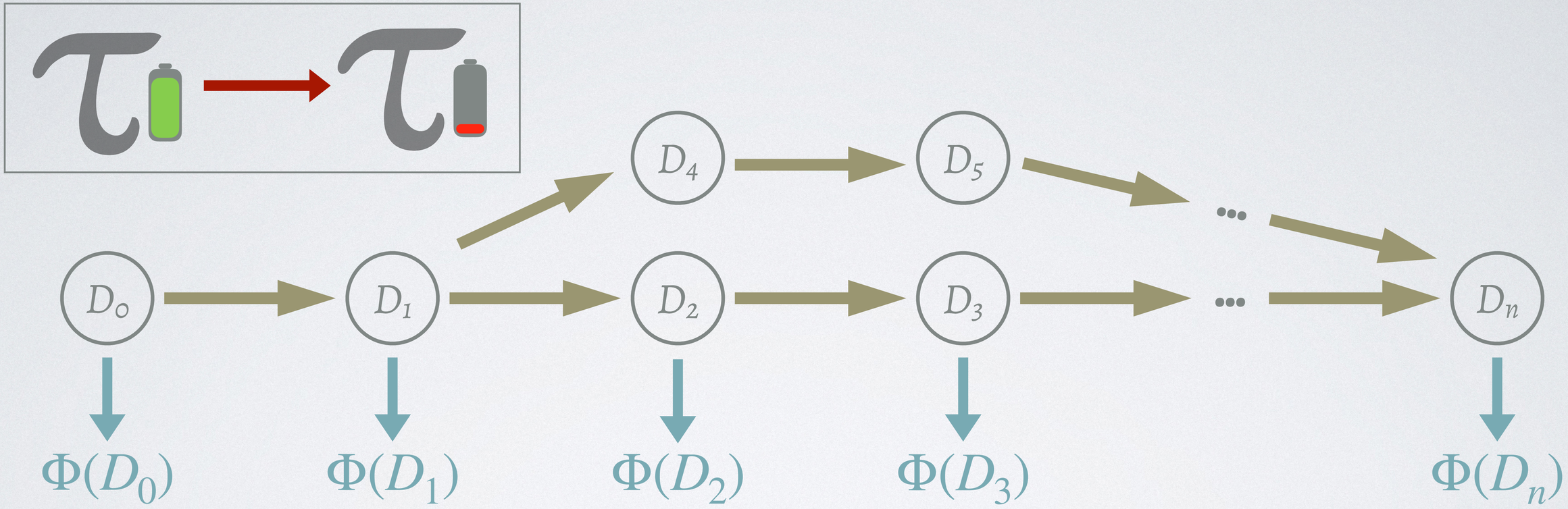
利用 Rust 语言精确内存抽象
进行实用的资源分析

数据驱动

针对**输入**的经验分布的
平均情况资源分析

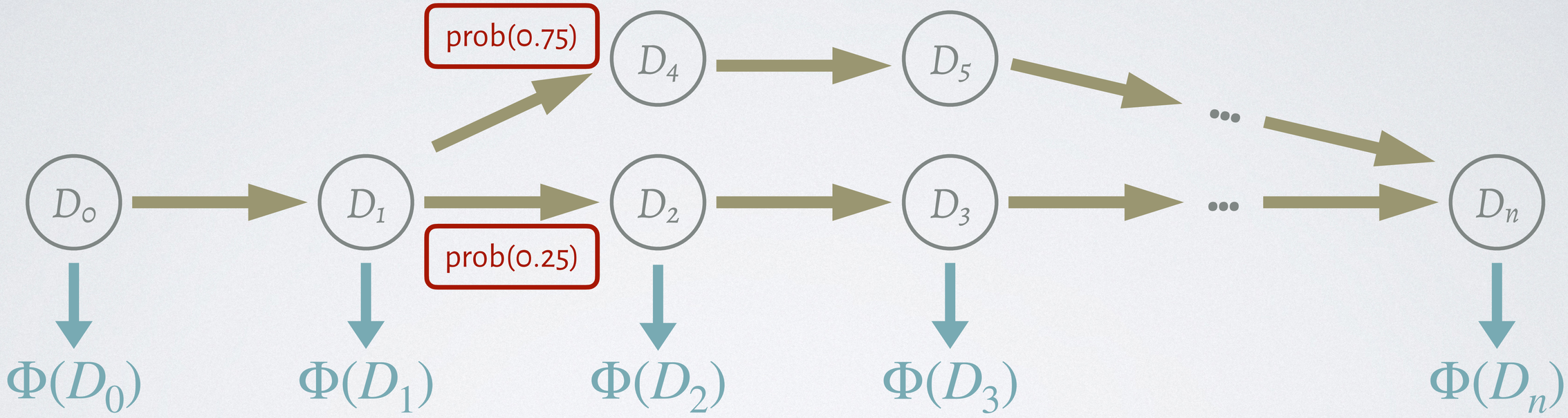
概率分析

我的研究：基于期望势能的分析



我的研究：基于期望势能的分析

D. Wang, D. M. Kahn, and J. Hoffmann. Raising Expectations: Automating Expected Cost Analysis with Types. In *ICFP'20*.



$$\Phi_E(D_1) \geq 0.75 \cdot (Cost(D_1, D_4) + \Phi_E(D_4)) + 0.25 \cdot (Cost(D_1, D_2) + \Phi_E(D_2))$$

两个例子

```
(* Gambler's ruin *)
let rec gr Alice Bob =
  match Alice with
  | [] -> ()
  | ha::ta ->
    match Bob with
    | [] -> ()
    | hb::tb ->
      let _ = tick(1) in
      if flip(0.5)
      then gr ta (ha::Bob)
      else gr (hb::Alice) tb
```

```
(* Makes a fair coin from a biased one *)
let rec make_fair (p:prob) =
  let _ = tick p*(1-p) in
  if flip(p)
  then (* H *)
    let _ = tick p*(1-p) in
    if flip(p)
    then make_fair p (* HH *)
    else H (* HT *)
  else (* T *)
    let _ = tick p*(1-p) in
    if flip(p)
    then T (* TH *)
    else make_fair p (* TT *)
```

两个例子

```
(* Gambler's ruin *)
let rec gr Alice Bob =
  match Alice with
  | [] -> ()
  | ha::ta ->
    match Bob with
    | [] -> ()
    | hb::tb ->
      let _ = tick(1) in
      if flip(0.5)
      then gr ta (ha::Bob)
      else gr (hb::Alice) tb
```

$$|\text{Alice}| * |\text{Bob}|$$

```
(* Makes a fair coin from a biased one *)
let rec make_fair (p:prob) =
  let _ = tick p*(1-p) in
  if flip(p)
  then (* H *)
    let _ = tick p*(1-p) in
    if flip(p)
    then make_fair p (* HH *)
    else H (* HT *)
  else (* T *)
    let _ = tick p*(1-p) in
    if flip(p)
    then T (* TH *)
    else make_fair p (* TT *)
```

两个例子

```
(* Gambler's ruin *)
let rec gr Alice Bob =
  match Alice with
  | [] -> ()
  | ha::ta ->
    match Bob with
    | [] -> ()
    | hb::tb ->
      let _ = tick(1) in
      if flip(0.5)
      then gr ta (ha::Bob)
      else gr (hb::Alice) tb
```

$$|\text{Alice}| * |\text{Bob}|$$

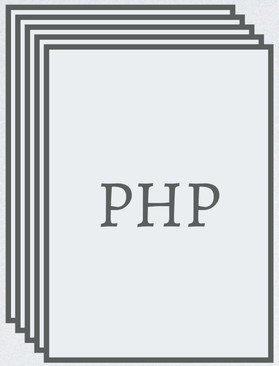
```
(* Makes a fair coin from a biased one *)
let rec make_fair (p:prob) =
  let _ = tick p*(1-p) in
  if flip(p)
  then (* H *)
    let _ = tick p*(1-p) in
    if flip(p)
    then make_fair p (* HH *)
    else H (* HT *)
  else (* T *)
    let _ = tick p*(1-p) in
    if flip(p)
    then T (* TH *)
    else make_fair p (* TT *)
```

$$1 / (p * (1 - p))$$

提纲

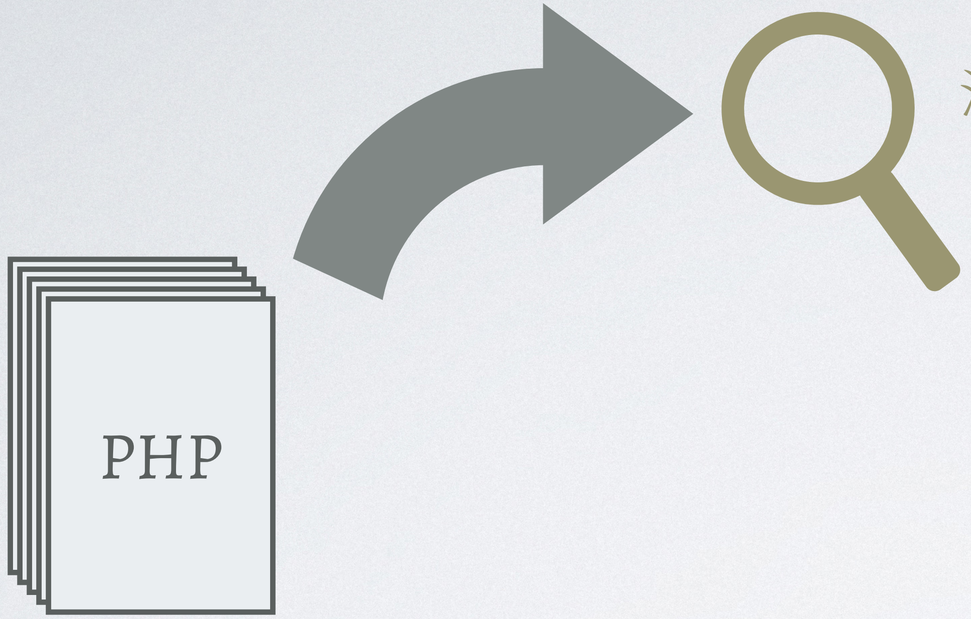
- [x] 目标阐述
- [x] 静态分析：基于势能方法的均摊分析
- [ ] 动态分析：最坏情况测试生成
- [ ] 程序验证：量化霍尔逻辑

最坏情况测试生成



[1] CVE - CVE-2011-4885: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-4885
[2] PHP 5.3.8 - Hashtables Denial of Service: https://www.exploit-db.com/exploits/18296/
[3] PHP: PHP 5 ChangeLog: http://www.php.net/ChangeLog-5.php#5.3.9

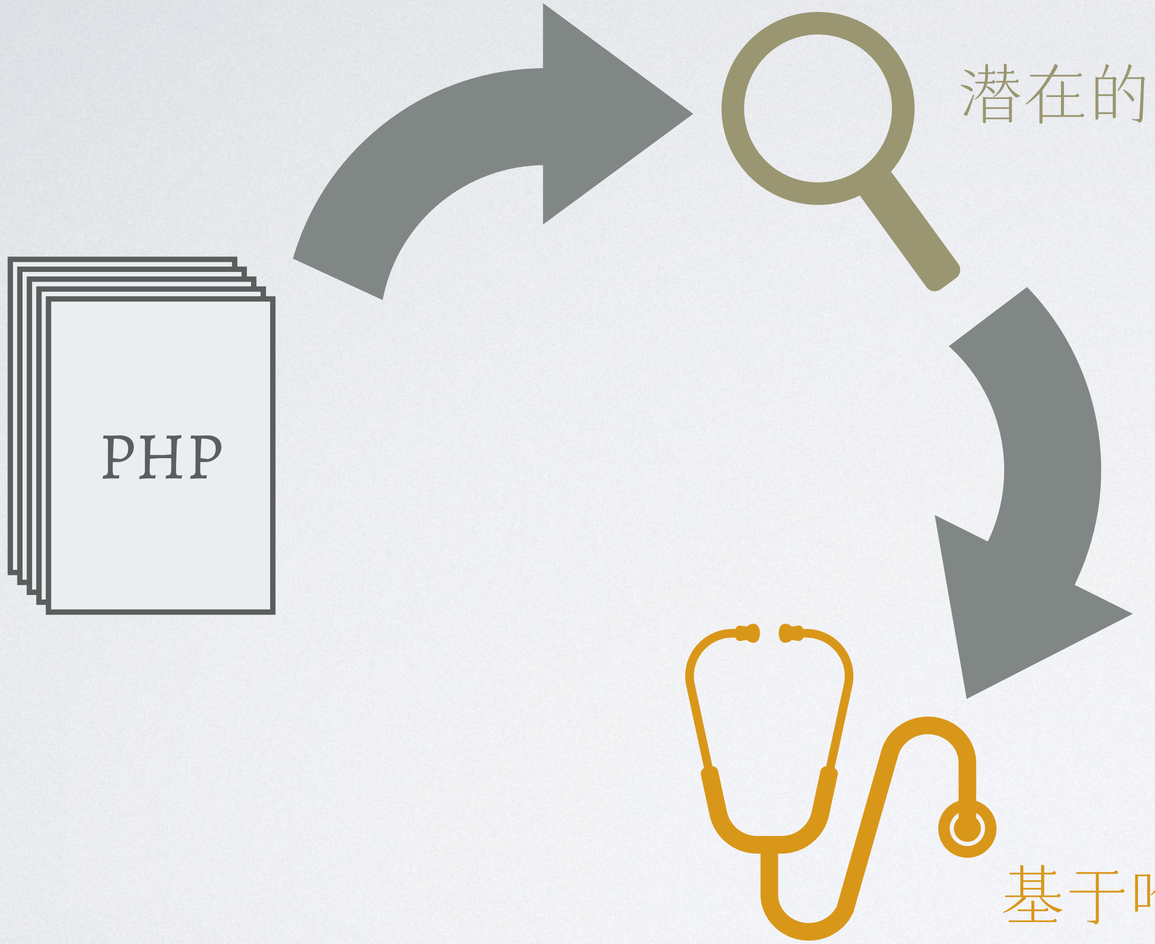
最坏情况测试生成



潜在的拒绝服务（Denial-of-Service）漏洞[1]

[1] CVE - CVE-2011-4885: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-4885
[2] PHP 5.3.8 - Hashtables Denial of Service: https://www.exploit-db.com/exploits/18296/
[3] PHP: PHP 5 ChangeLog: http://www.php.net/ChangeLog-5.php#5.3.9

最坏情况测试生成

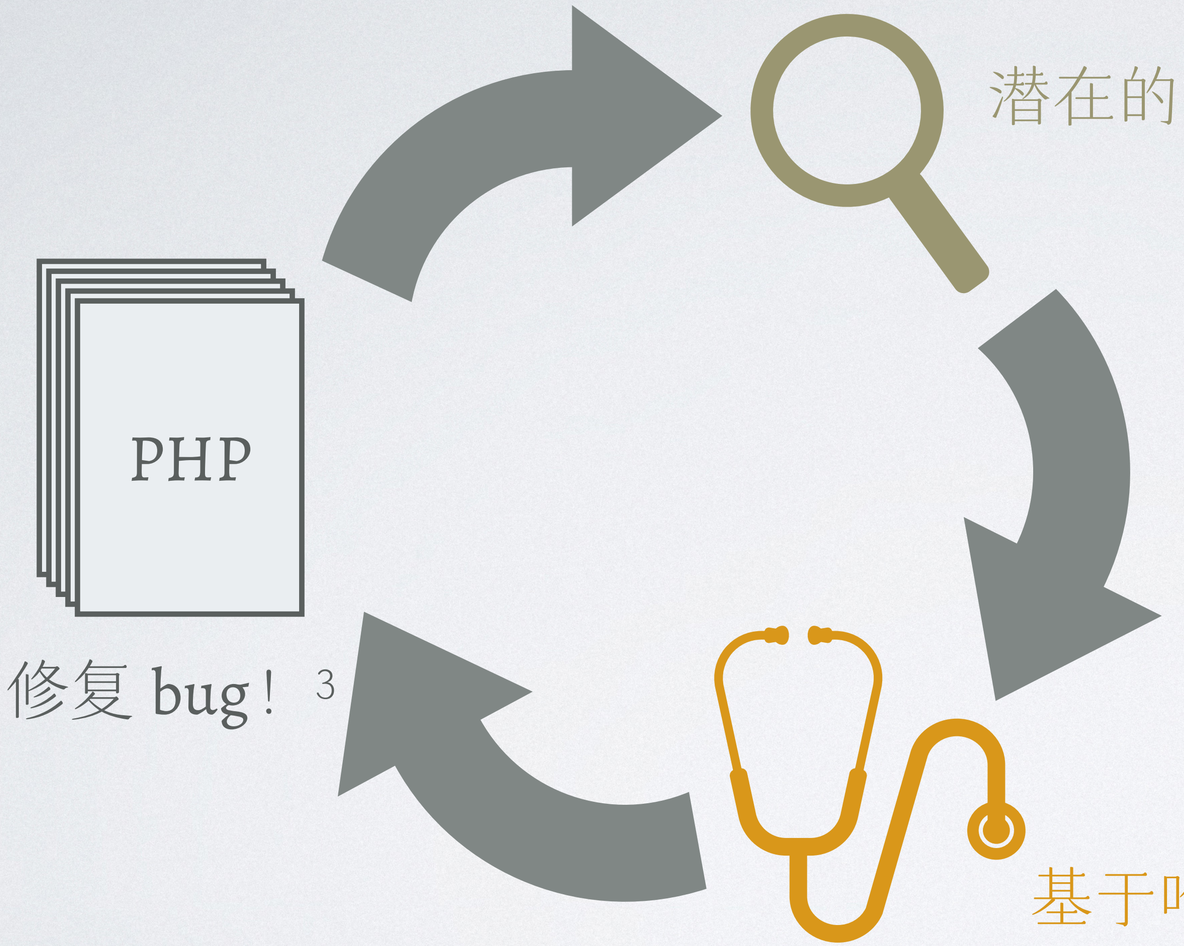


潜在的拒绝服务（Denial-of-Service）漏洞 [1]

基于哈希冲突的具体攻击实例 [2]

[1] CVE - CVE-2011-4885: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-4885
[2] PHP 5.3.8 - Hashtables Denial of Service: https://www.exploit-db.com/exploits/18296/
[3] PHP: PHP 5 ChangeLog: http://www.php.net/ChangeLog-5.php#5.3.9

最坏情况测试生成



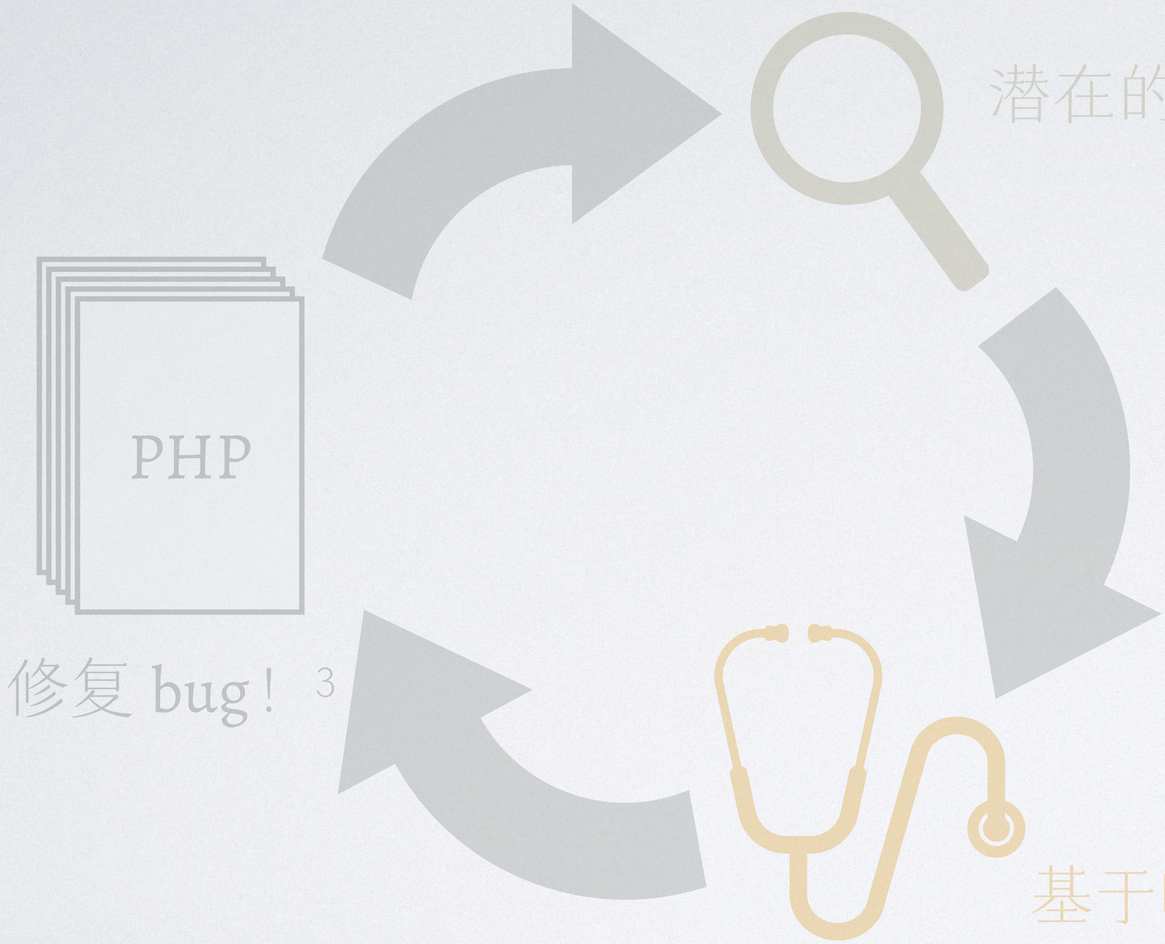
潜在的拒绝服务（Denial-of-Service）漏洞 1

基于哈希冲突的具体攻击实例 2

1 CVE - CVE-2011-4885: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-4885
2 PHP 5.3.8 - Hashtables Denial of Service: https://www.exploit-db.com/exploits/18296/
3 PHP: PHP 5 ChangeLog: http://www.php.net/ChangeLog-5.php#5.3.9

最坏情况测试生成

潜在的拒绝服务（Denial-of-Service）漏洞 [1]



修复 bug！ [3]

基于哈希冲突的具体攻击实例 [2]

最坏情况测试用例可以帮助开发者发现并修复**性能**相关的 bug！

[1] CVE - CVE-2011-4885: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-4885
[2] PHP 5.3.8 - Hashtables Denial of Service: https://www.exploit-db.com/exploits/18296/
[3] PHP: PHP 5 ChangeLog: http://www.php.net/ChangeLog-5.php#5.3.9

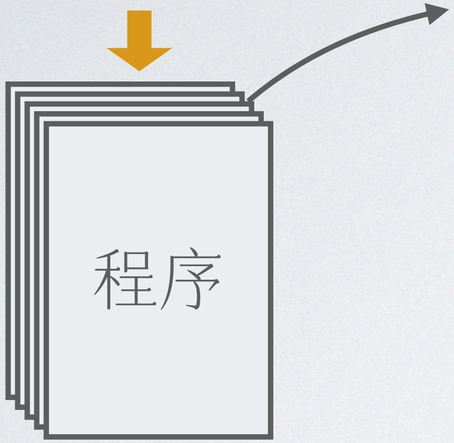
基于符号执行的最坏情况测试生成

基于符号执行的最坏情况测试生成

输入的规约
（例如列表长度）

基于符号执行的最坏情况测试生成

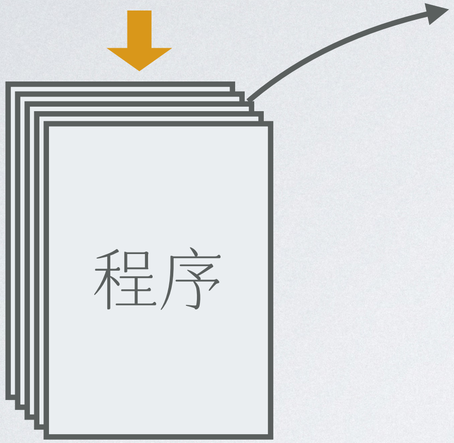
输入的规约
（例如列表长度）



可能的执行路径 1

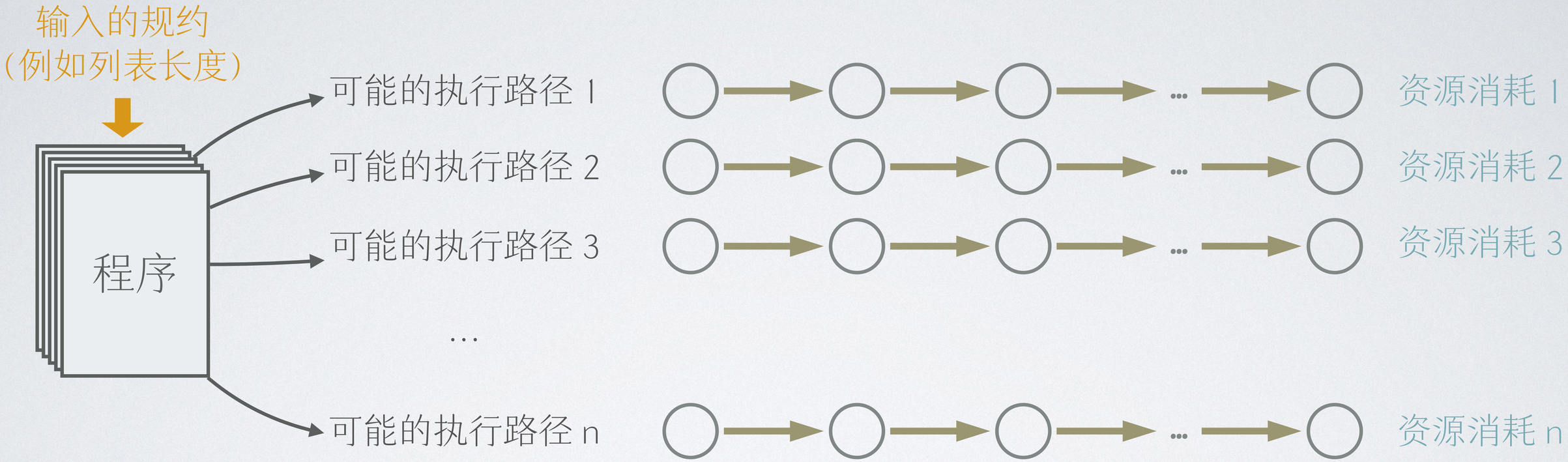
基于符号执行的最坏情况测试生成

输入的规约
（例如列表长度）

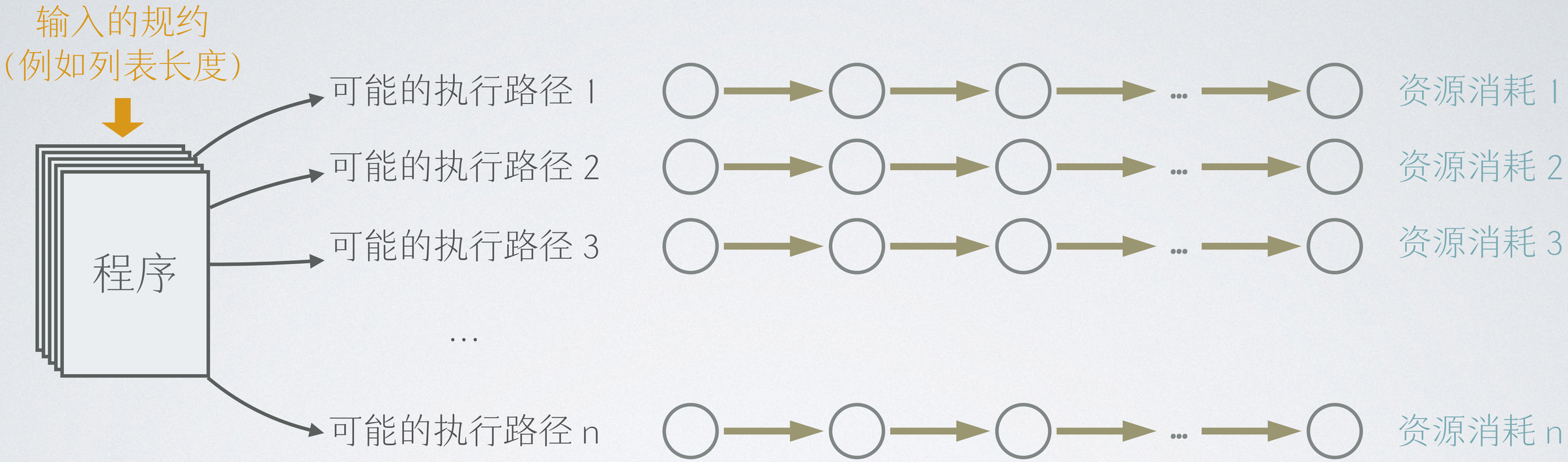


可能的执行路径 |  资源消耗 |

基于符号执行的最坏情况测试生成

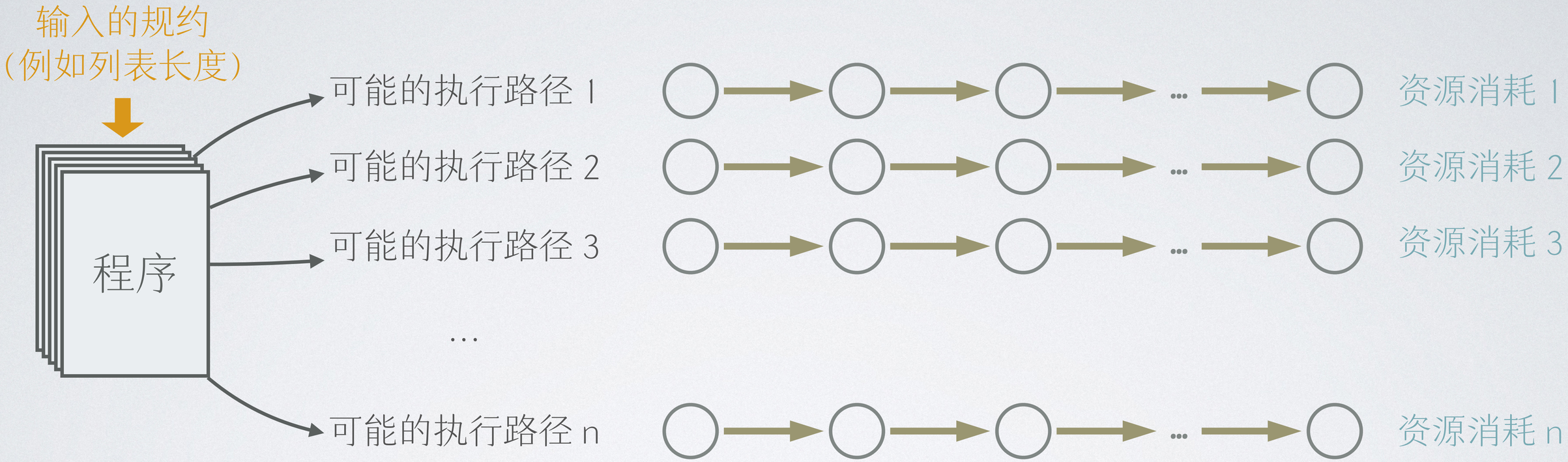


基于符号执行的最坏情况测试生成



- 给定一个输入的规约，搜索一条具有最大资源消耗的程序执行路径

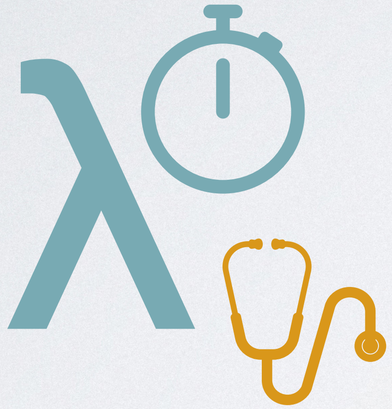
基于符号执行的最坏情况测试生成



- 给定一个输入的规约，搜索一条具有最大资源消耗的程序执行路径
- 是否可以利用静态资源分析的结果来指导符号执行的搜索过程呢？

我的研究：资源制导最坏情况输入生成

D. Wang and J. Hoffmann. Type-Guided Worst-Case Input Generation. In *POPL'19*.



我的研究：资源制导最坏情况输入生成

D. Wang and J. Hoffmann. Type-Guided Worst-Case Input Generation. In *POPL'19*.

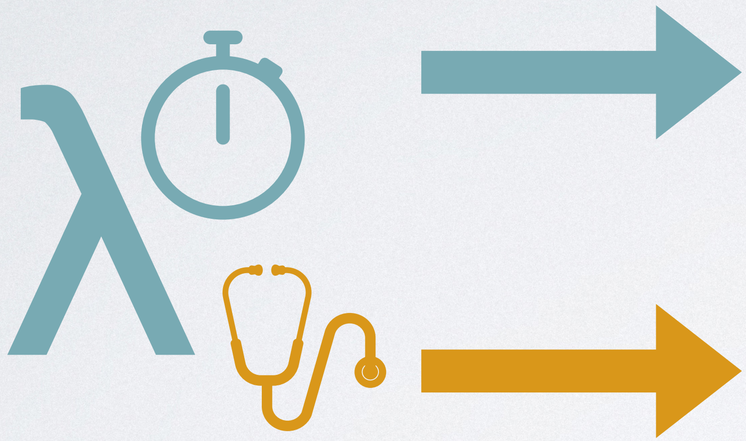
基于势能的自动资源分析

指导

基于符号执行的最坏情况输入生成

我的研究：资源制导最坏情况输入生成

D. Wang and J. Hoffmann. Type-Guided Worst-Case Input Generation. In *POPL'19*.



基于势能的自动资源分析

指导

基于符号执行的最坏情况输入生成

首个**有理论正确性**保证的，基于**静态资源分析**的
最坏情况输入生成算法

资源制导的符号执行

```
let rec lpairs l =
  match l with
  | [] -> []
  | x1::xs ->
    match xs with
    | [] -> []
    | x2::xs' ->
      if x1 < x2 then
        let () = tick(2) in
        (x1,x2)::(lpairs xs')
      else
        lpairs xs'
```

资源制导的符号执行

$\Phi = |\ell|$

```
let rec lpairs l =
  match l with
  | [] -> []
  | x1::xs ->
    match xs with
    | [] -> []
    | x2::xs' ->
      if x1 < x2 then
        let () = tick(2) in
        (x1,x2)::(lpairs xs')
      else
        lpairs xs'
```

资源制导的符号执行

$\Phi = |\ell|$

$\ell \mapsto [\text{int}^1, \text{int}^2, \text{int}^3, \text{int}^4]$

```
let rec lpairs l =
  match l with
  | [] -> []
  | x1::xs ->
    match xs with
    | [] -> []
    | x2::xs' ->
      if x1 < x2 then
        let () = tick(2) in
        (x1,x2)::(lpairs xs')
      else
        lpairs xs'
```

资源制导的符号执行

$\Phi = |\ell|$

```
let rec lpairs l =
  match l with
  | [] -> []
  | x1::xs ->
    match xs with
    | [] -> []
    | x2::xs' ->
      if x1 < x2 then
        let () = tick(2) in
        (x1,x2)::(lpairs xs')
      else
        lpairs xs'
```

$\ell \mapsto [\text{int}^1, \text{int}^2, \text{int}^3, \text{int}^4]$

$x_1 \mapsto \text{int}^1, x_2 \mapsto \text{int}^2,$

$xs' \mapsto [\text{int}^3, \text{int}^4]$

资源制导的符号执行

$\Phi = |\ell|$

```
let rec lpairs l =
  match l with
  | [] -> []
  | x1::xs ->
    match xs with
    | [] -> []
    | x2::xs' ->
      if x1 < x2 then
        let () = tick(2) in
        (x1,x2)::(lpairs xs')
      else
        lpairs xs'
```

$\ell \mapsto [\text{int}^1, \text{int}^2, \text{int}^3, \text{int}^4]$

$\Phi = |xs'| + 2 = 4$

$x_1 \mapsto \text{int}^1, x_2 \mapsto \text{int}^2,$

$xs' \mapsto [\text{int}^3, \text{int}^4]$

资源制导的符号执行

$\Phi = |\ell|$

$\ell \mapsto [\text{int}^1, \text{int}^2, \text{int}^3, \text{int}^4]$

```
let rec lpairs l =
  match l with
  | [] -> []
  | x1::xs ->
    match xs with
    | [] -> []
    | x2::xs' ->
      if x1 < x2 then
        let () = tick(2) in
        (x1,x2)::(lpairs xs')
      else
        lpairs xs'
```

$\Phi = |xs'| + 2 = 4$

$x_1 \mapsto \text{int}^1, x_2 \mapsto \text{int}^2,$

$xs' \mapsto [\text{int}^3, \text{int}^4]$

Cost = 2

资源制导的符号执行

$\Phi = |\ell|$

$\ell \mapsto [\text{int}^1, \text{int}^2, \text{int}^3, \text{int}^4]$

```
let rec lpairs l =
  match l with
  | [] -> []
  | x1::xs ->
    match xs with
    | [] -> []
    | x2::xs' ->
      if x1 < x2 then
        let () = tick(2) in
        (x1,x2)::(lpairs xs')
      else
        lpairs xs'
```

$\Phi = |xs'| + 2 = 4$

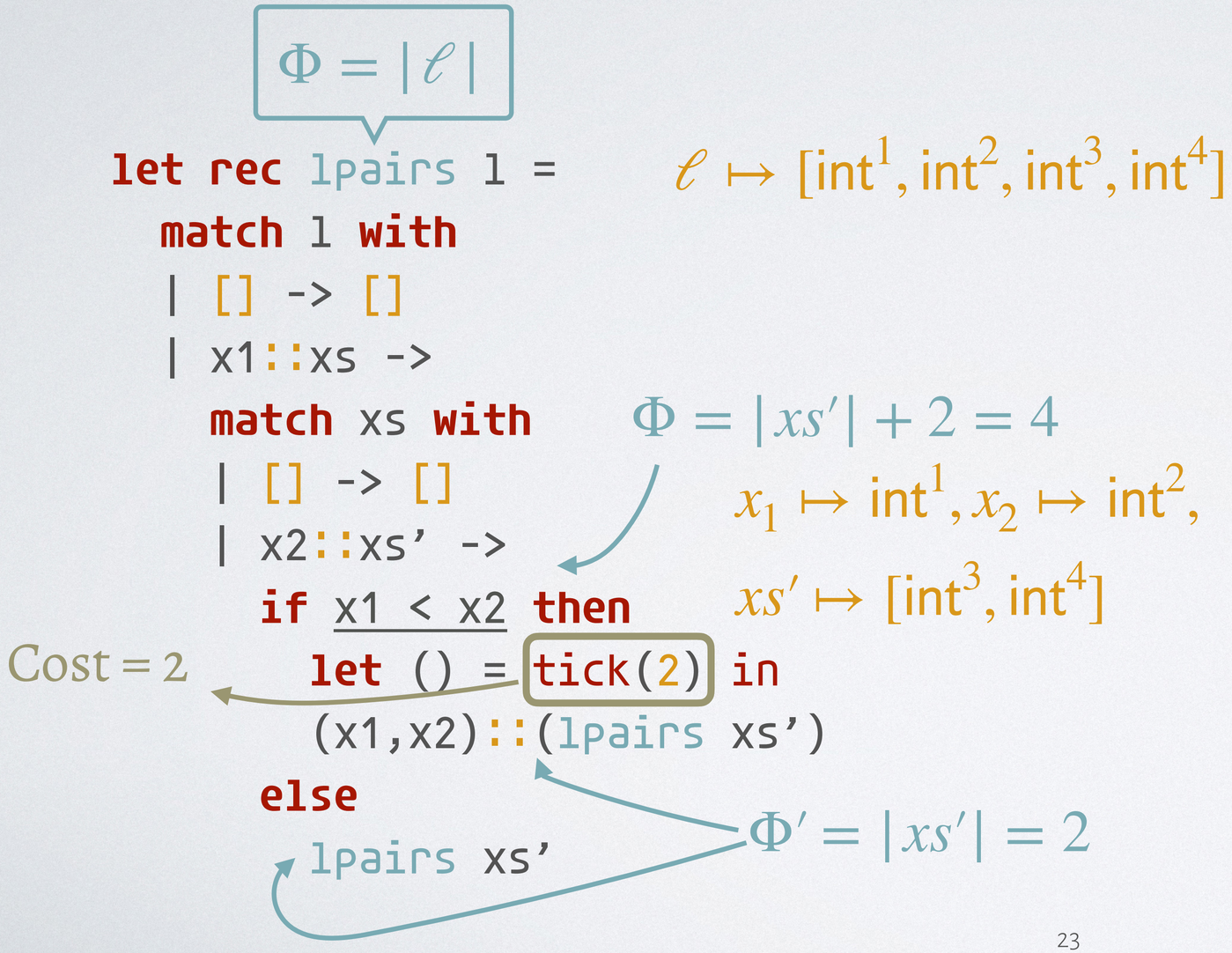
$x_1 \mapsto \text{int}^1, x_2 \mapsto \text{int}^2,$

$xs' \mapsto [\text{int}^3, \text{int}^4]$

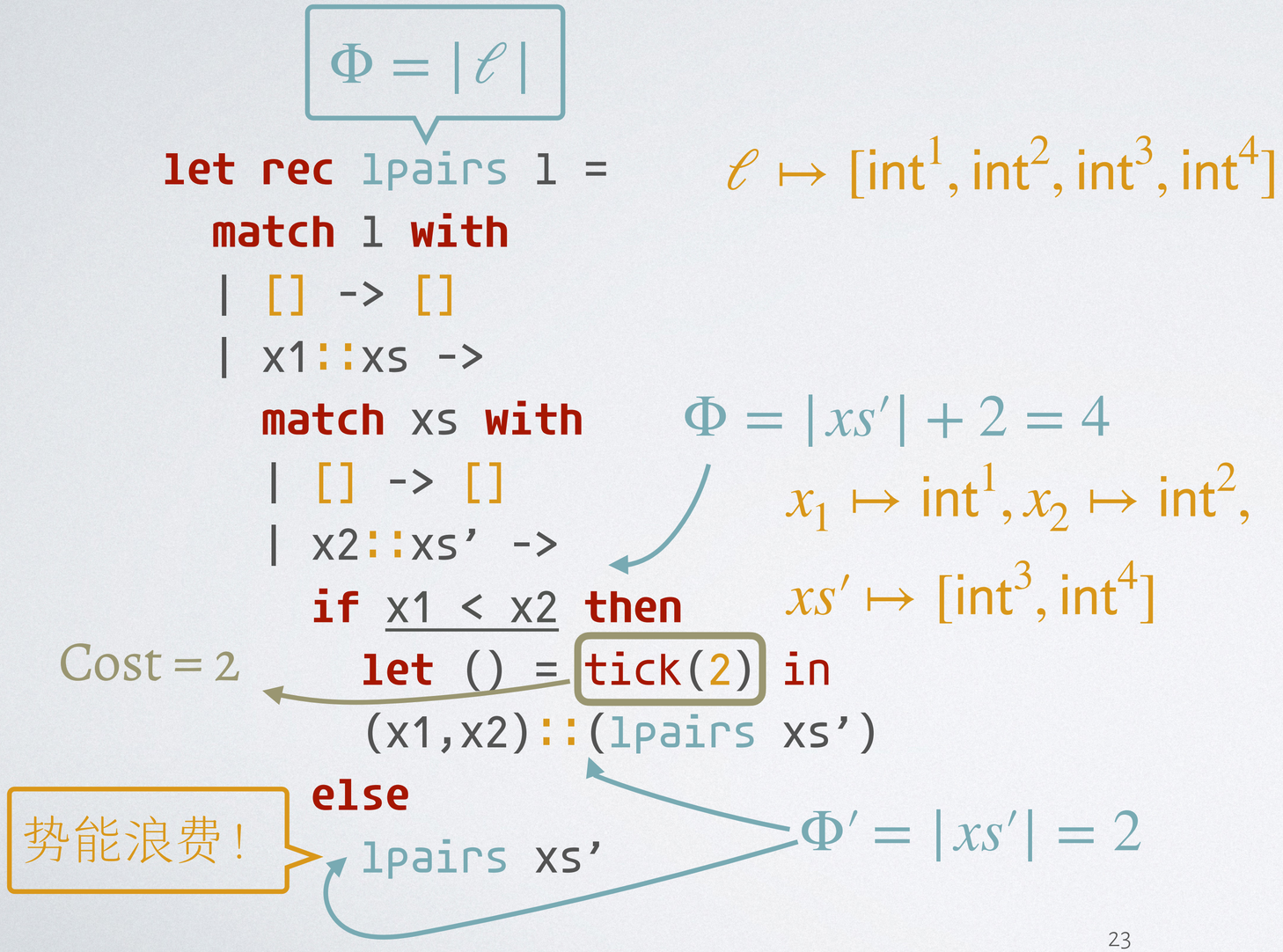
Cost = 2

$\Phi' = |xs'| = 2$

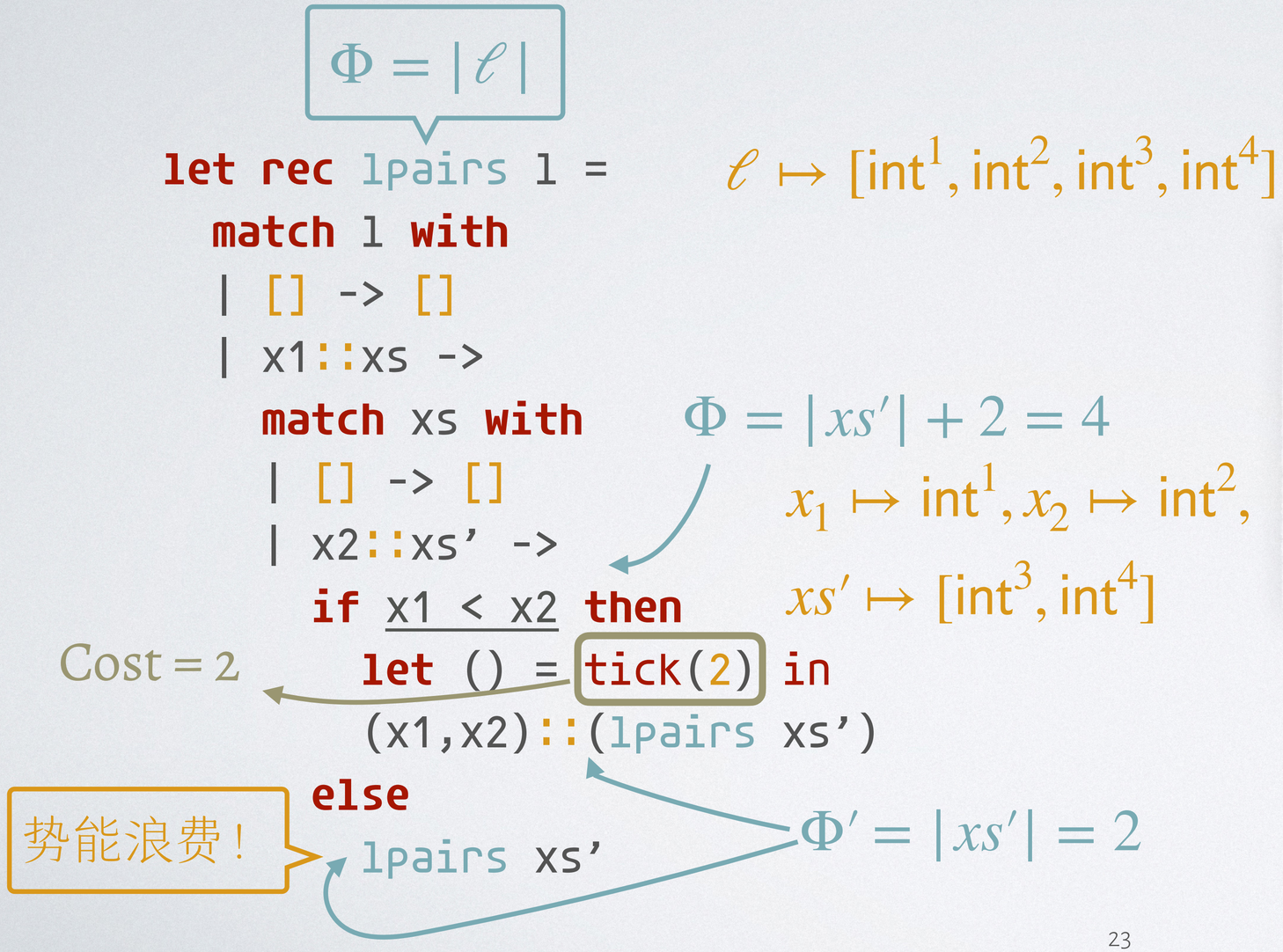
资源制导的符号执行


$\Phi = |\ell|$
let rec lpairs l =
match l with
| [] -> []
| x1::xs ->
match xs with
| [] -> []
| x2::xs' ->
if x1 < x2 then
let () = tick(2) in
(x1,x2)::(lpairs xs')
else
lpairs xs'
$\ell \mapsto [\text{int}^1, \text{int}^2, \text{int}^3, \text{int}^4]$
$\Phi = |xs'| + 2 = 4$
$x_1 \mapsto \text{int}^1, x_2 \mapsto \text{int}^2,$
$xs' \mapsto [\text{int}^3, \text{int}^4]$
Cost = 2
$\Phi' = |xs'| = 2$

资源制导的符号执行

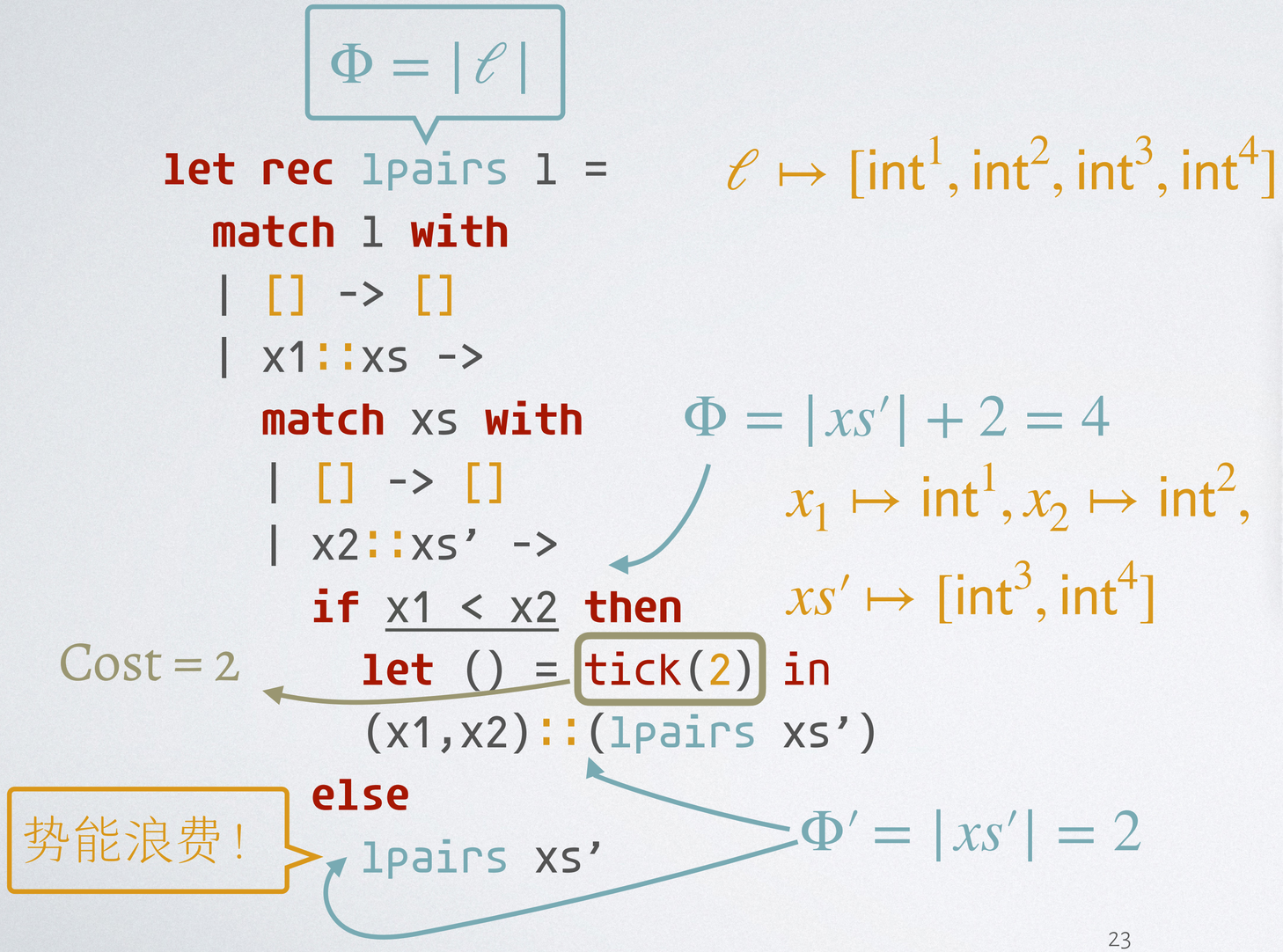

$\Phi = |\ell|$
let rec lpairs l =
match l with
| [] -> []
| x1::xs ->
match xs with
| [] -> []
| x2::xs' ->
if x1 < x2 then
let () = tick(2) in
(x1,x2)::(lpairs xs')
else
lpairs xs'
$\ell \mapsto [\text{int}^1, \text{int}^2, \text{int}^3, \text{int}^4]$
$\Phi = |xs'| + 2 = 4$
$x_1 \mapsto \text{int}^1, x_2 \mapsto \text{int}^2,$
$xs' \mapsto [\text{int}^3, \text{int}^4]$
Cost = 2
$\Phi' = |xs'| = 2$
势能浪费！

资源制导的符号执行



如果一条程序执行路径**没有**任何**势能浪费**，那么它一定具有最坏的资源消耗

资源制导的符号执行



如果一条程序执行路径**没有**任何**势能浪费**，那么它一定具有最坏的资源消耗

通过这个信息减少搜索量！

理论结果

理论结果

正确性：如果算法生成了一个输入，那么被分析程序在该输入上的资源消耗精确等于静态资源分析所得出的上界

理论结果

正确性：如果算法生成了一个输入，那么被分析程序在该输入上的资源消耗精确等于静态资源分析所得出的上界

完备性：给定一个输入规约，如果存在一个满足此规约的输入，使得被分析程序在该输入上的资源消耗精确等于静态资源分析所得出的上界，那么算法一定能搜索到该输入所对应的程序执行路径

部分测试程序

描述	输入规约	算法	算法 + 优化
插入排序	200 个整数	7.74s	6.97s
快速排序	200 个整数	T/O	53.23s
快速排序（字典序）	长度为 100，99，…，1 的子列表	439.35s	438.79s
基于双栈的队列	200 个队列操作	444.64s	T/O
在树上找之字形路径	200 个树节点	T/O	4.87s
键值为字符串的哈希表	64 个插入操作	7.64s	7.62s

动态资源分析的研究展望

动态资源分析的研究展望

概率方法

动态资源分析的研究展望

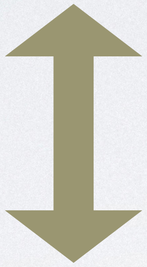
最坏情况程序运行路径

概率方法

动态资源分析的研究展望

概率方法

最坏情况程序运行路径



最大似然样本

动态资源分析的研究展望

概率方法

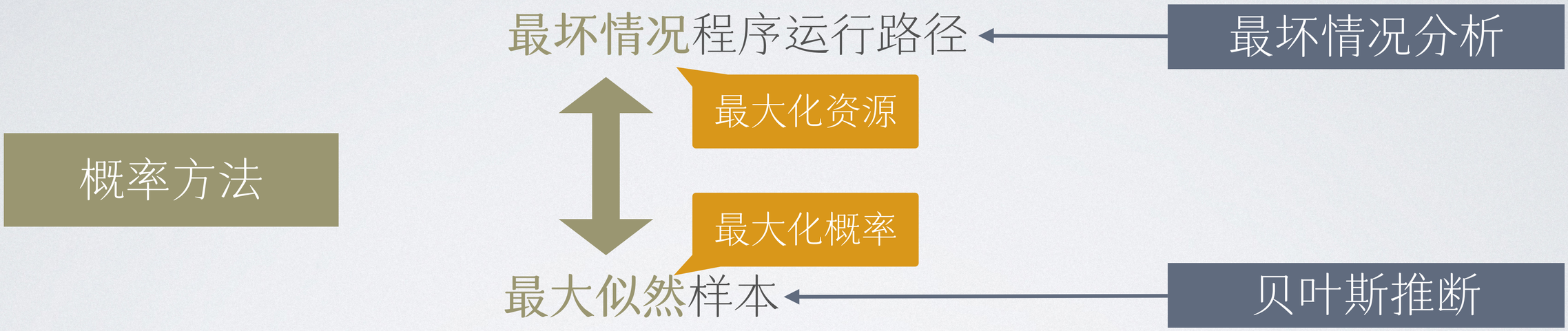
最坏情况程序运行路径

最大化资源

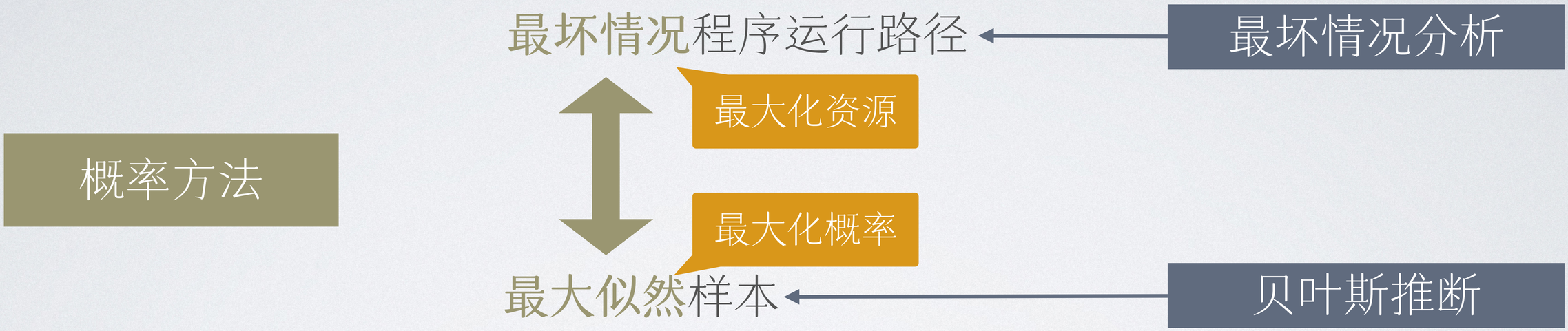
最大化概率

最大似然样本

动态资源分析的研究展望



动态资源分析的研究展望



是否可以迁移贝叶斯推断的方法用于更大规模的测试生成？

通过贝叶斯推断生成测试

```
var quickSort = function(arr) {
  if (arr.length < 2) {
    return arr;
  }
  const pivot = arr[0];

  let left = [];
  let right = [];
  let equal = [];

  map(function(val) {
    if (val < pivot) {
      factor(3);
      left.push(val);
    } else if (val > pivot) {
      factor(2);
      right.push(val);
    } else {
      factor(1);
      equal.push(val);
    }
  }, arr);

  return quickSort(left).concat(equal, quickSort(right));
}
```

在 WebPPL 中实现的快速排序

通过贝叶斯推断生成测试

```
var quickSort = function(arr) {
  if (arr.length < 2) {
    return arr;
  }
  const pivot = arr[0];

  let left = [];
  let right = [];
  let equal = [];

  map(function(val) {
    if (val < pivot) {
      factor(3);
      left.push(val);
    } else if (val > pivot) {
      factor(2);
      right.push(val);
    } else {
      factor(1);
      equal.push(val);
    }
  }, arr);

  return quickSort(left).concat(equal, quickSort(right));
}
```

在 WebPPL 中实现的快速排序

通过贝叶斯推断生成测试

```
var quickSort = function(arr) {
  if (arr.length < 2) {
    return arr;
  }
  const pivot = arr[0];

  let left = [];
  let right = [];
  let equal = [];

  map(function(val) {
    if (val < pivot) {
      factor(3);
      left.push(val);
    } else if (val > pivot) {
      factor(2);
      right.push(val);
    } else {
      factor(1);
      equal.push(val);
    }
  }, arr);

  return quickSort(left).concat(equal, quickSort(right));
}
```

factor(c) 表示将当前执行路径的**对数概率**增加 **c**

在 WebPPL 中实现的快速排序

通过贝叶斯推断生成测试

```
var quickSort = function(arr) {
  if (arr.length < 2) {
    return arr;
  }
  const pivot = arr[0];

  let left = [];
  let right = [];
  let equal = [];

  map(function(val) {
    if (val < pivot) {
      factor(3);
      left.push(val);
    } else if (val > pivot) {
      factor(2);
      right.push(val);
    } else {
      factor(1);
      equal.push(val);
    }
  }, arr);

  return quickSort(left).concat(equal, quickSort(right));
}
```

在 WebPPL 中实现的快速排序

factor(c) 表示将当前执行路径的**对数概率**增加 **c**

本质上与资源分析中的 **tick(c)** 效果相同！

通过贝叶斯推断生成测试

```
var quickSort = function(arr) {
  if (arr.length < 2) {
    return arr;
  }
  const pivot = arr[0];

  let left = [];
  let right = [];
  let equal = [];

  map(function(val) {
    if (val < pivot) {
      factor(3);
      left.push(val);
    } else if (val > pivot) {
      factor(2);
      right.push(val);
    } else {
      factor(1);
      equal.push(val);
    }
  }, arr);

  return quickSort(left).concat(equal, quickSort(right));
}
```

在 WebPPL 中实现的快速排序

factor(c) 表示将当前执行路径的对数概率增加 **c**

本质上与资源分析中的 **tick(c)** 效果相同！

- 设定输入为长度为 5 的实数数组，采样 100 次后得到的最大似然样本为：
  - [0.936, 0.548, 0.519, 0.139, 0.093]

提纲

- [x] 目标阐述
- [x] 静态分析：基于势能方法的均摊分析
- [x] 动态分析：最坏情况测试生成
- [ ] 程序验证：量化霍尔逻辑

量化霍尔逻辑

Hoare Logic

$\{\Gamma\}\ S\ \{\Gamma'\}$

量化霍尔逻辑

Hoare Logic

$$\{\Gamma\}\ S\ \{\Gamma'\}$$

$$\Gamma, \Gamma' : \mathit{State} \to \mathit{bool}$$

量化霍尔逻辑

Hoare Logic

$\{\Gamma\}\ S\ \{\Gamma'\}$

$\Gamma, \Gamma' : State \to bool$

如果初始状态 σ 满足 $\Gamma(\sigma)$ 为真
且运行程序 S 到达终止状态 σ'
那么 $\Gamma'(\sigma')$ 也为真

量化霍尔逻辑

Hoare Logic	Quantitative Hoare Logic
$\{\Gamma\}\ S\ \{\Gamma'\}$	$\{\Phi\}\ S\ \{\Phi'\}$
$\Gamma, \Gamma' : State \to bool$	$\Phi, \Phi' : State \to \mathbb{Q}_{\geq 0} \cup \{\infty\}$
如果初始状态 σ 满足 $\Gamma(\sigma)$ 为真 且运行程序 S 到达终止状态 σ' 那么 $\Gamma'(\sigma')$ 也为真	

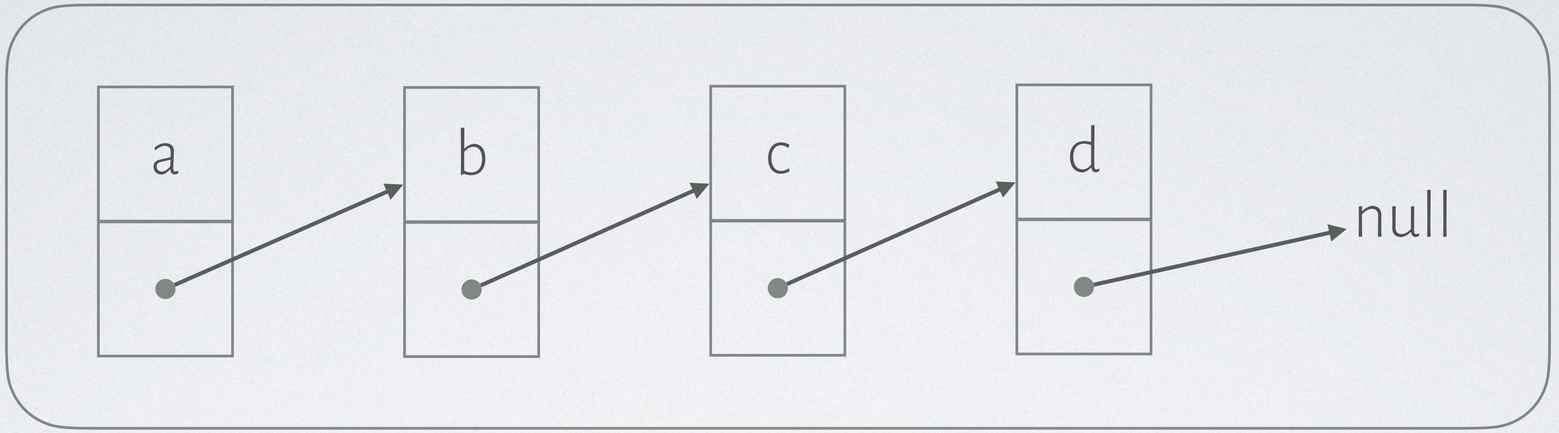
量化霍尔逻辑

Hoare Logic	**Quantitative Hoare Logic**
$\{\Gamma\}\ S\ \{\Gamma'\}$	$\{\Phi\}\ S\ \{\Phi'\}$
$\Gamma, \Gamma' : State \to bool$	$\Phi, \Phi' : State \to \mathbb{Q}_{\geq 0} \cup \{\infty\}$
如果初始状态 σ 满足 $\Gamma(\sigma)$ 为真 且运行程序 S 到达终止状态 σ' 那么 $\Gamma'(\sigma')$ 也为真	如果初始有至少 $\Phi(\sigma)$ 单位的资源 且运行程序 S 从 σ 到达 σ' 那么终止有至少 $\Phi'(\sigma')$ 单位的资源

量化霍尔逻辑

Hoare Logic	**Quantitative Hoare Logic**
$\{\Gamma\}\ S\ \{\Gamma'\}$	$\{\Phi\}\ S\ \{\Phi'\}$
$\Gamma, \Gamma' : State \to bool$	$\Phi, \Phi' : State \to \mathbb{Q}_{\geq 0} \cup \{\infty\}$
$false$	∞
如果初始状态 σ 满足 $\Gamma(\sigma)$ 为真 且运行程序 S 到达终止状态 σ' 那么 $\Gamma'(\sigma')$ 也为真	如果初始有至少 $\Phi(\sigma)$ 单位的资源 且运行程序 S 从 σ 到达 σ' 那么终止有至少 $\Phi'(\sigma')$ 单位的资源

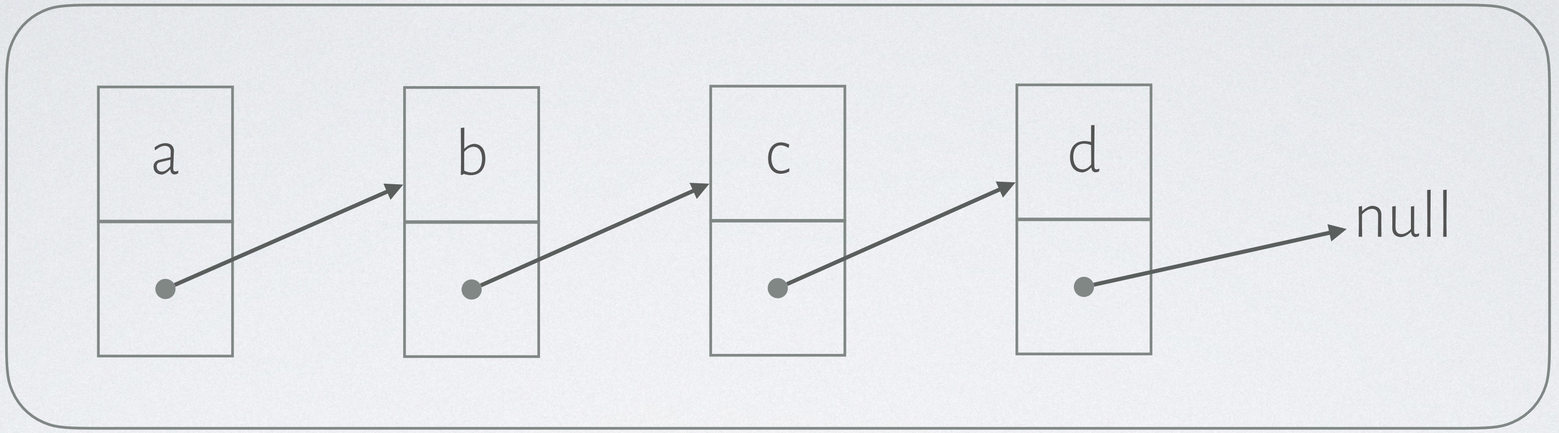
量化霍尔逻辑 + 分离逻辑



堆

[Atkey10] R. Atkey. 2010. Amortised Resource Analysis with Separation Logic. In *ESOP'10*.

量化霍尔逻辑 + 分离逻辑

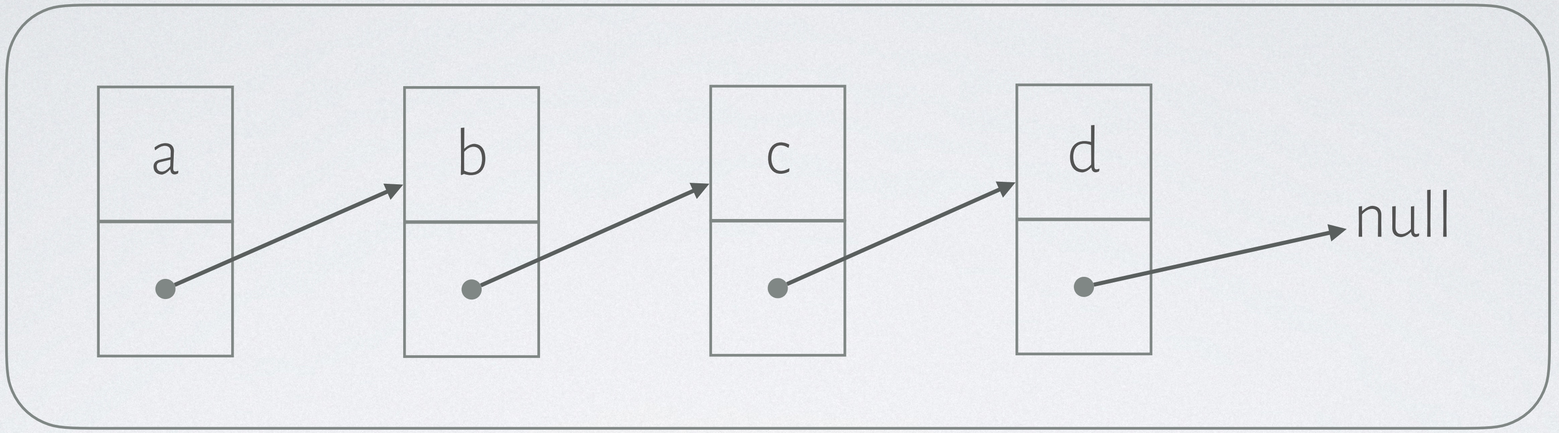


堆

$$\mathsf{lseg}(x, y) \equiv (x = y \wedge \mathsf{emp}) \vee \exists d, z\,.\,[x \mapsto_{\mathsf{data}} d] * [x \mapsto_{\mathsf{next}} z] * \mathsf{lseg}(z, y)$$

[Atkey10] R. Atkey. 2010. Amortised Resource Analysis with Separation Logic. In *ESOP'10*.

量化霍尔逻辑 + 分离逻辑



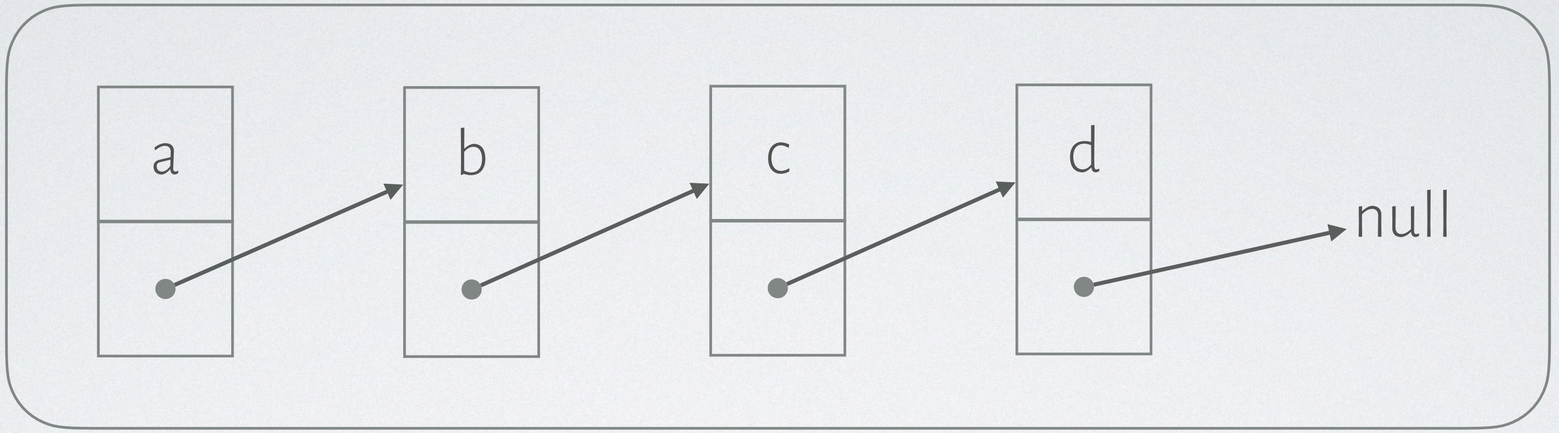
堆

$$\mathsf{lseg}(x, y) \equiv (x = y \wedge \mathsf{emp}) \vee \exists d, z\,.\,[x \mapsto_{\mathsf{data}} d] * [x \mapsto_{\mathsf{next}} z] * \mathsf{lseg}(z, y)$$

以 x 为头 y 为尾的单链表

[Atkey10] R. Atkey. 2010. Amortised Resource Analysis with Separation Logic. In *ESOP'10*.

量化霍尔逻辑 + 分离逻辑



堆

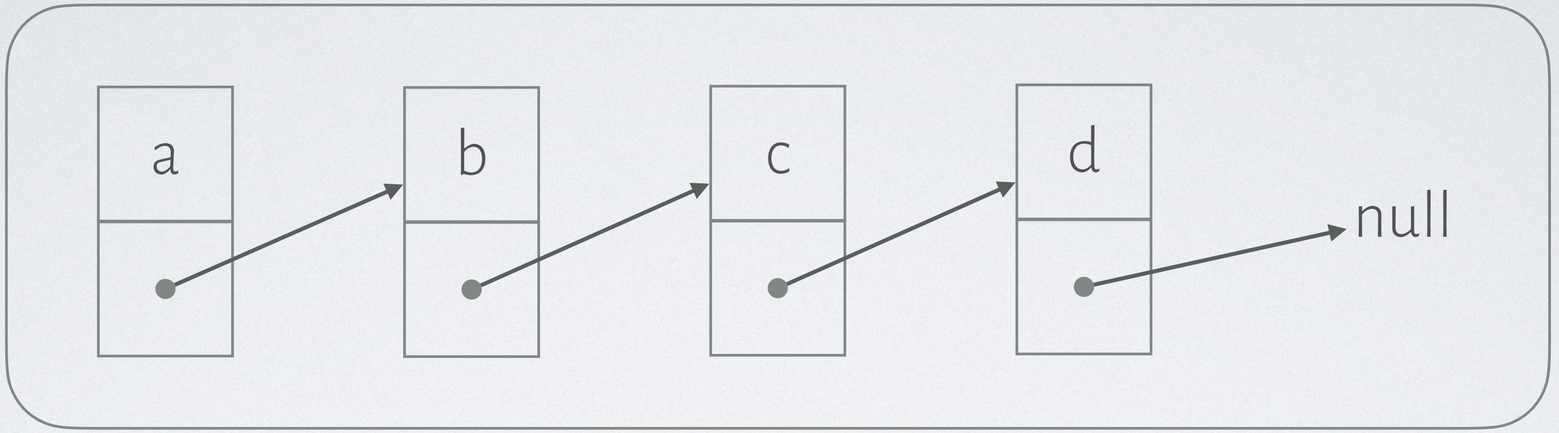
$$\mathsf{lseg}(x, y) \equiv (x = y \wedge \mathsf{emp}) \vee \exists d, z\,.\,[x \mapsto_{\mathsf{data}} d] * [x \mapsto_{\mathsf{next}} z] * \mathsf{lseg}(z, y)$$

以 x 为头 y 为尾的单链表

空堆

[Atkey10] R. Atkey. 2010. Amortised Resource Analysis with Separation Logic. In *ESOP'10*.

量化霍尔逻辑 + 分离逻辑



堆

$$\mathsf{lseg}(x, y) \equiv (x = y \wedge \mathsf{emp}) \vee \exists d, z\,.\,[x \mapsto_{\mathsf{data}} d] * [x \mapsto_{\mathsf{next}} z] * \mathsf{lseg}(z, y)$$

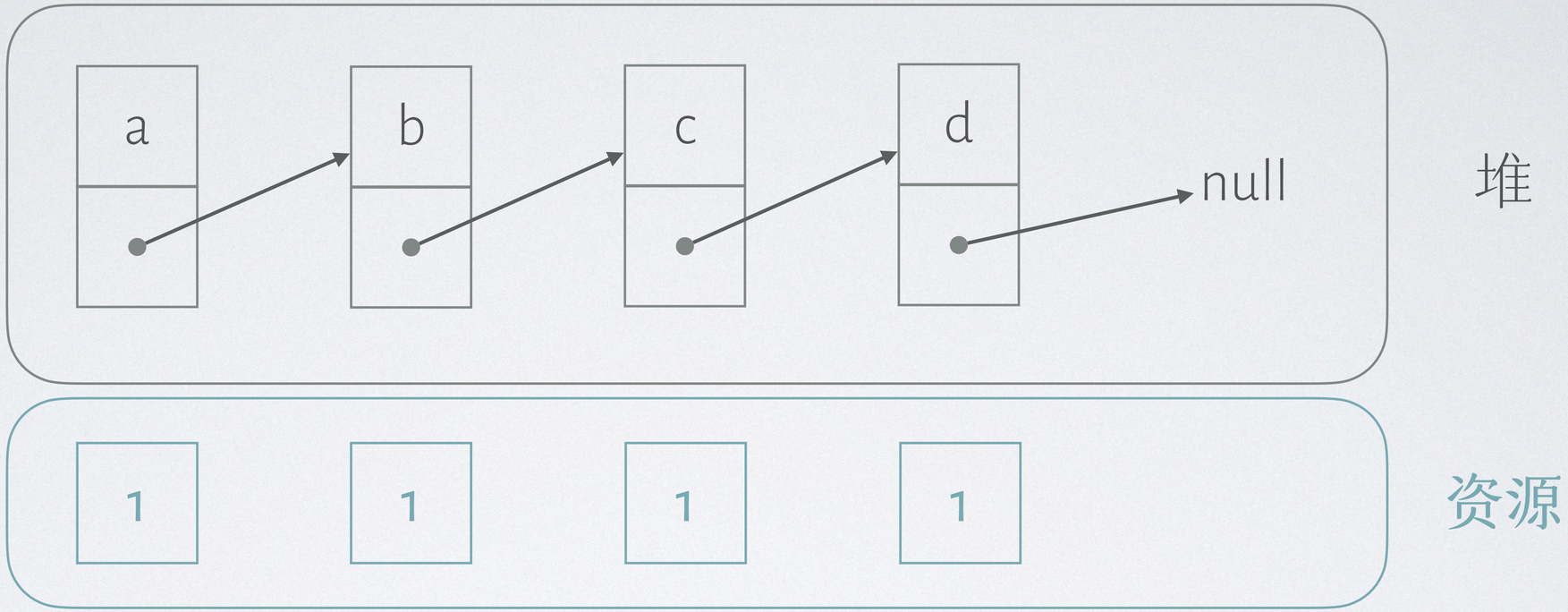
以 x 为头 y 为尾的单链表

空堆

分离合取（不相交的堆）

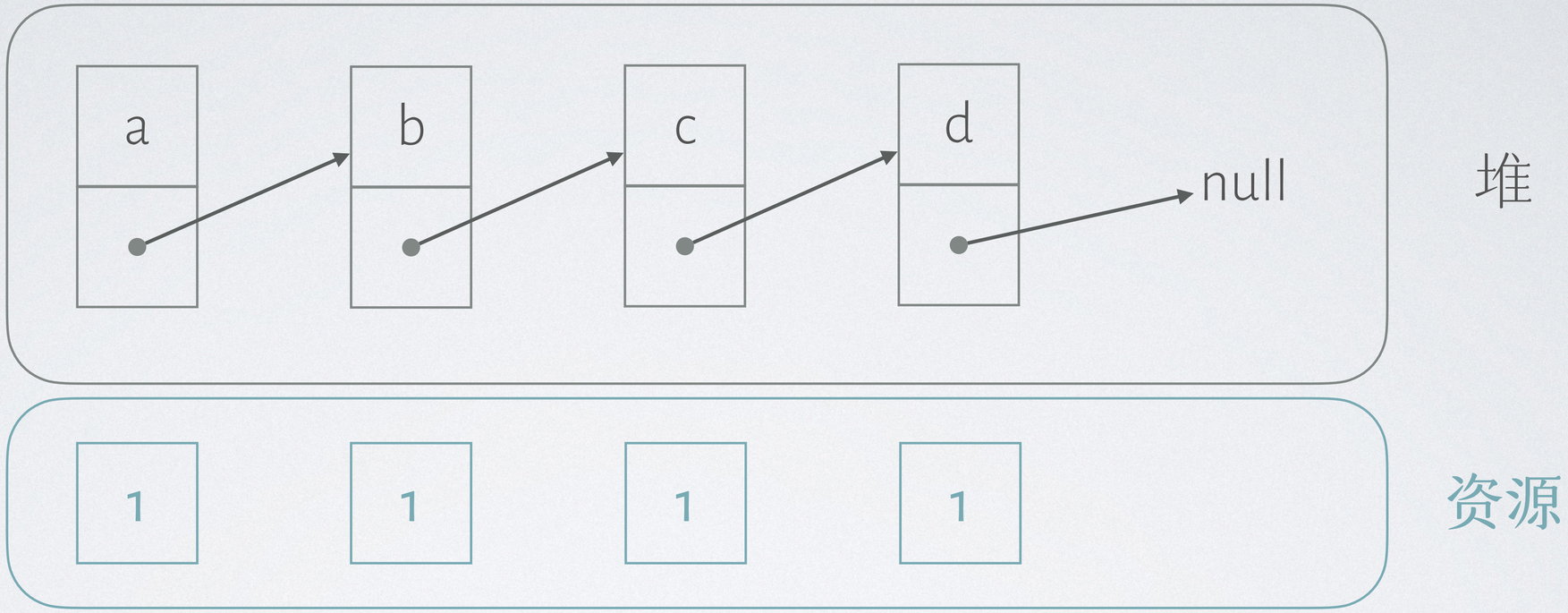
[Atkey10] R. Atkey. 2010. Amortised Resource Analysis with Separation Logic. In *ESOP'10*.

量化霍尔逻辑 + 分离逻辑



[Atkey10] R. Atkey. 2010. Amortised Resource Analysis with Separation Logic. In *ESOP'10*.

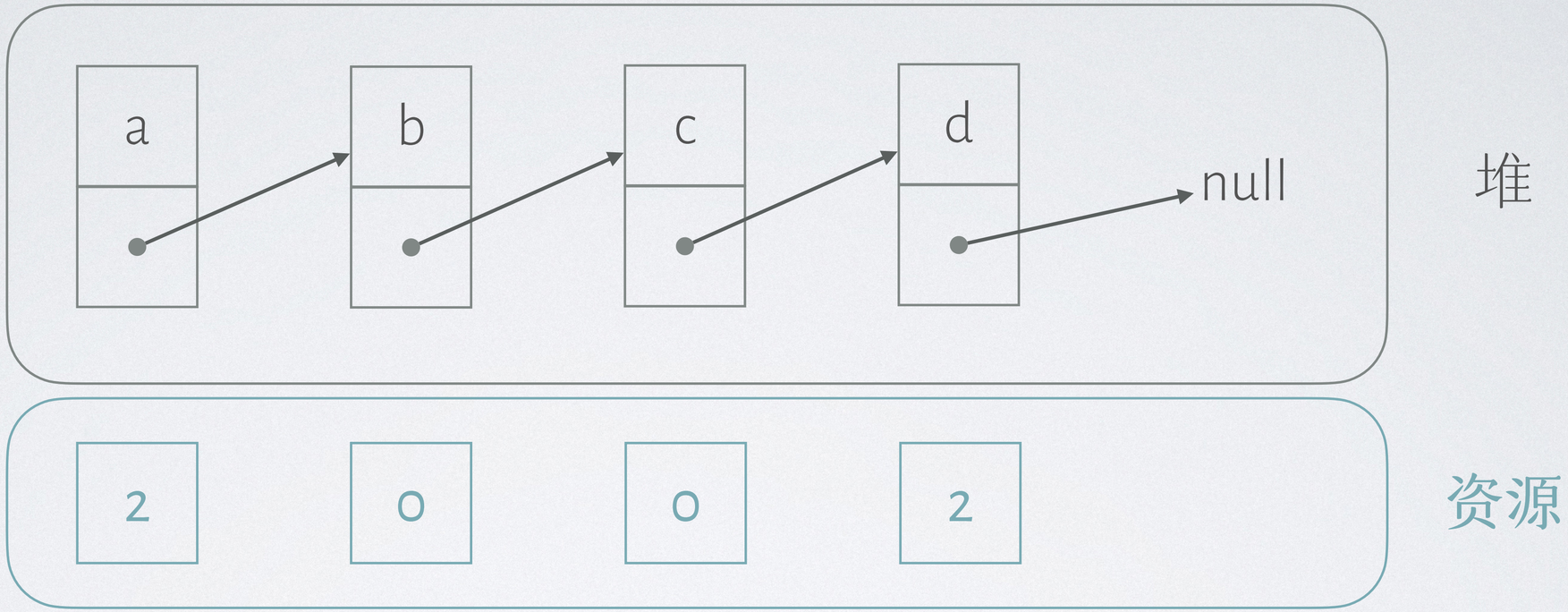
量化霍尔逻辑 + 分离逻辑



$$\mathsf{lseg}^1(x,y) \equiv (x = y \wedge \mathsf{emp}) \vee \exists d, z\,.\,[x \mapsto_{\mathsf{data}} d] * [x \mapsto_{\mathsf{next}} z] * \mathsf{lseg}^1(z,y) * \Diamond^1$$

[Atkey10] R. Atkey. 2010. Amortised Resource Analysis with Separation Logic. In *ESOP'10*.

量化霍尔逻辑 + 分离逻辑



$$\mathsf{lseg}^{p}(x,y) \equiv (x = y \wedge \mathsf{emp}) \vee \exists d, z\,.\,[x \mapsto_{\mathsf{data}} d] * [x \mapsto_{\mathsf{next}} z] * \mathsf{lseg}^{p}(z,y) * \Diamond^{p(d)}$$

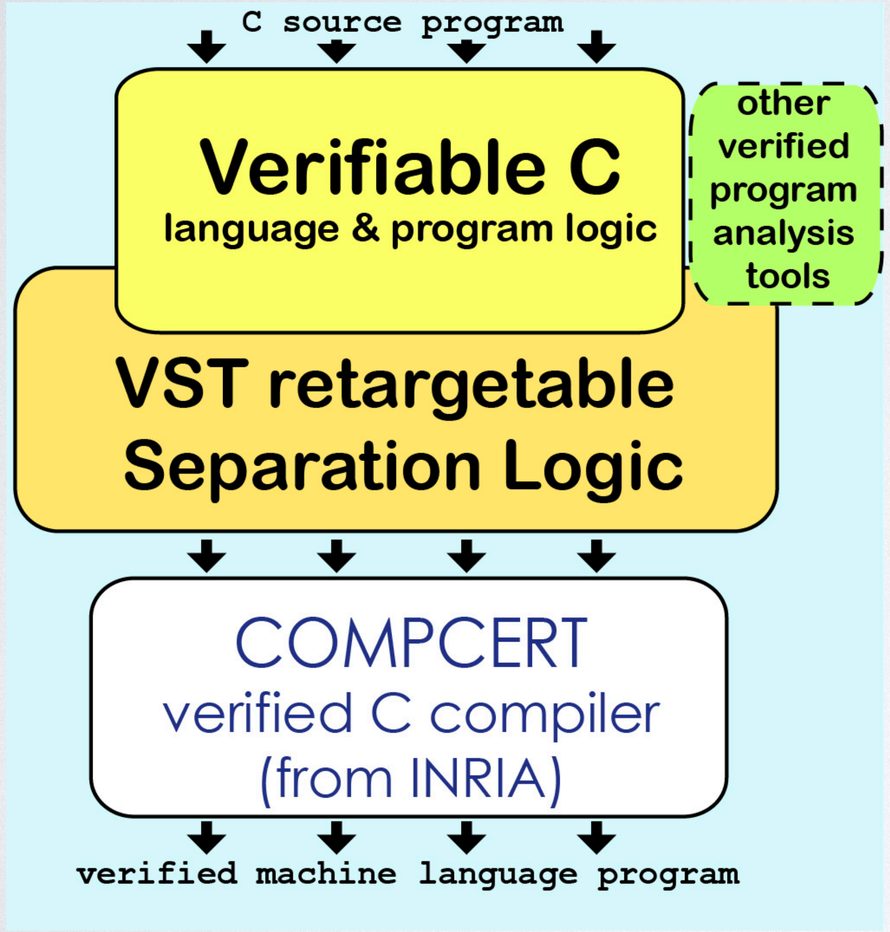
$$p(d) \equiv \mathsf{ite}(d \neq 0,2,0)$$

[Atkey10] R. Atkey. 2010. Amortised Resource Analysis with Separation Logic. In *ESOP'10*.

研究展望：VST + 资源验证

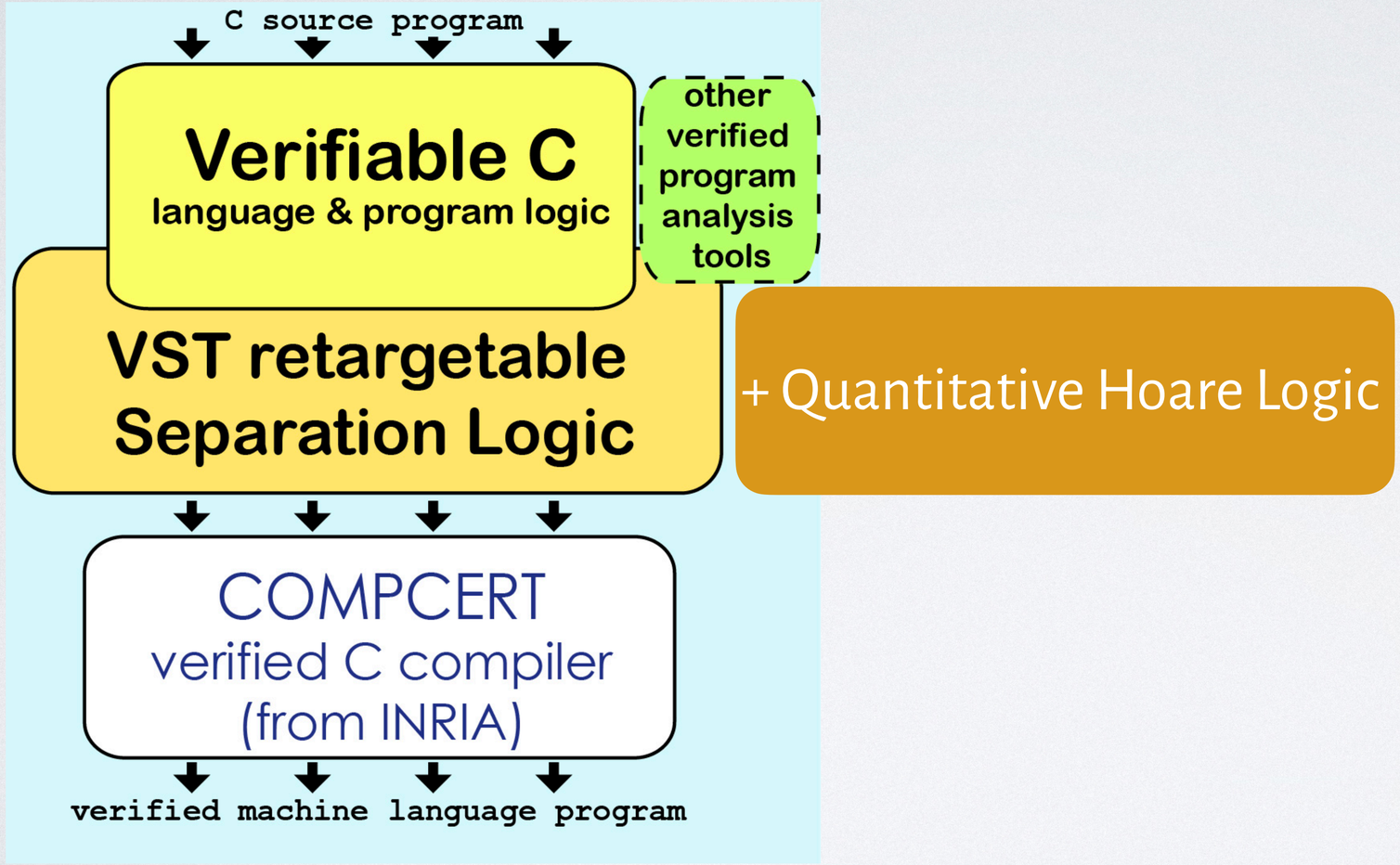
[Appel11] A. Appel. 2011. Verified Software Toolchain. In *ESOP'11*.

研究展望：VST + 资源验证



[Appel11] A. Appel. 2011. Verified Software Toolchain. In *ESOP'11*.

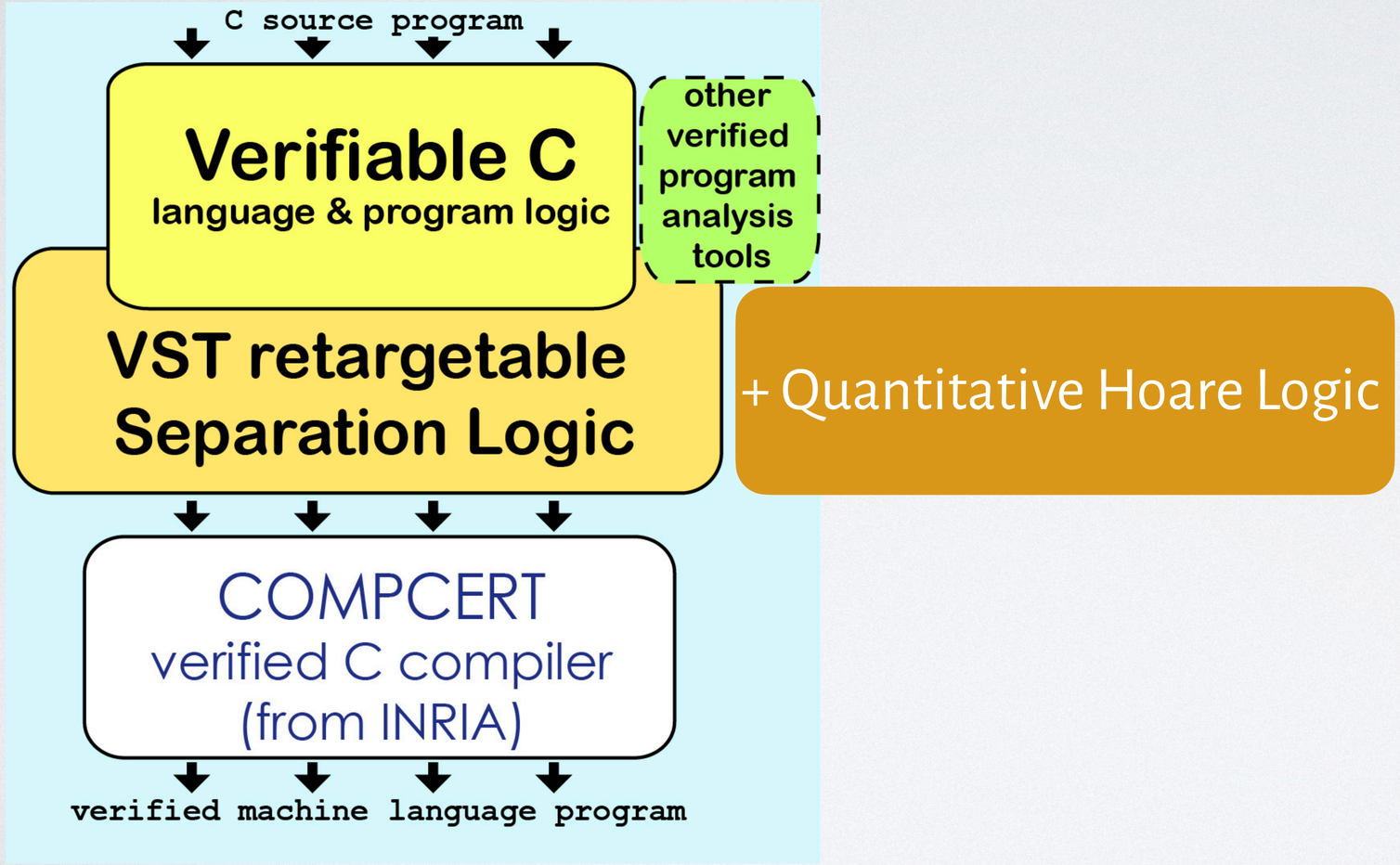
研究展望：VST + 资源验证



+ Quantitative Hoare Logic

[Appel11] A. Appel. 2011. Verified Software Toolchain. In *ESOP'11*.

研究展望：VST + 资源验证



+ Quantitative Hoare Logic

- 除了验证 C 程序的功能正确性/安全性外，通过量化霍尔逻辑验证 C 程序的资源消耗的正确性/安全性
- 可以开发为 VST 框架的扩展

[Appel11] A. Appel. 2011. Verified Software Toolchain. In *ESOP'11*.

量化程序分析与验证

量化程序分析与验证

- 静态分析：基于势能方法的均摊分析
- 动态分析：最坏情况测试生成
- 程序验证：量化霍尔逻辑

量化程序分析与验证

- [x] 静态分析：基于势能方法的均摊分析
- [x] 动态分析：最坏情况测试生成
- [x] 程序验证：量化霍尔逻辑
- [ ] 程序合成：自动生成满足资源消耗规约的程序

谢谢

wangdi95@pku.edu.cn