形式化方法导引

第 5 章 模型检测

5.2 理论

5.2.3 BMC Algorithm | 5.2.4 K-induction

黄文超

http://staff.ustc.edu.cn/~huangwc/fm.html

Bounded model checking (有界模型检测)

- A way to exploit *SAT/SMT* for verifying properties of programs

例: a marble puzzle

We do steps in which either

- *one marble* is added, or
- the number of marbles is *doubled*

What is the *smallest number* of steps required to end up in *exactly 1000* marbles?

How to solve this by NuSMV?

```
MODULE main
VAR
  M : 1..1000;
INIT
  M=1
TRANS
  case M<=999 : next(M)=M+1; TRUE : next(M)=M; esac |
  case M<=500: next(M)=2*M; TRUE : next(M)=M; esac
LTLSPEC
  G !(M=1000)
```

A parameter for NuSMV: *-bmc* or *-bmc -bmc_length 11*

- Looks for counterexamples of length 1, 2, ...
- So it is incomplete

How to solve this by NuSMV?

```
MODULE main
VAR
  M : 1..1000;
INIT
  M=1
TRANS
  case M<=999 : next(M)=M+1; TRUE : next(M)=M; esac |
  case M<=500: next(M)=2*M; TRUE : next(M)=M; esac
LTLSPEC
  G !(M=1000)
```

A parameter for NuSMV: *-bmc* or *-bmc -bmc_length 11*

- Looks for counterexamples of length 1, 2, ...
- So it is incomplete

How to solve this by NuSMV?

```
MODULE main
VAR
  M : 1..1000;
INIT
  M=1
TRANS
  case M<=999 : next(M)=M+1; TRUE : next(M)=M; esac |
  case M<=500: next(M)=2*M; TRUE : next(M)=M; esac
LTLSPEC
  G !(M=1000)
```

A parameter for NuSMV: *-bmc* or *-bmc -bmc_length 11*

- Looks for counterexamples of length 1, 2, ...
- So it is incomplete

How to solve this by NuSMV?

```
MODULE main
VAR
  M : 1..1000;
INIT
  M=1
TRANS
  case M<=999 : next(M)=M+1; TRUE : next(M)=M; esac |
  case M<=500: next(M)=2*M; TRUE : next(M)=M; esac
LTLSPEC
  G !(M=1000)
```

A parameter for NuSMV: *-bmc* or *-bmc -bmc_length 11*

- Looks for counterexamples of length 1, 2, ...
- So it is incomplete

运行结果: ./NuSMV *-bmc* c-sample1-bmc.smv

```
-- no counterexample found with bound 0
-- no counterexample found with bound 1
-- no counterexample found with bound 2
-- no counterexample found with bound 3
-- no counterexample found with bound 4
-- no counterexample found with bound 5
-- no counterexample found with bound 6
-- no counterexample found with bound 7
-- no counterexample found with bound 8
-- no counterexample found with bound 9
-- no counterexample found with bound 10
```

运行结果: ./NuSMV *-bmc -bmc_length 13* c-sample1-bmc.smv

```
-- no counterexample found with bound 0
-- no counterexample found with bound 1
-- no counterexample found with bound 2
-- no counterexample found with bound 3
-- no counterexample found with bound 4
-- no counterexample found with bound 5
-- no counterexample found with bound 6
-- no counterexample found with bound 7
-- no counterexample found with bound 8
-- no counterexample found with bound 9
-- no counterexample found with bound 10
-- no counterexample found with bound 11
-- no counterexample found with bound 12
-- no counterexample found with bound 13
```

运行结果: ./NuSMV *-bmc -bmc_length 14* c-sample1-bmc.smv

```
-- specification  G !(M = 1000)    is false
-- as demonstrated by the following execution sequence
Trace Description: BMC Counterexample
Trace Type: Counterexample
```

```
-> State: 1.1 <-
  M = 1
-> State: 1.2 <-
  M = 2
-> State: 1.3 <-
  M = 3
-> State: 1.4 <-
  M = 6
-> State: 1.5 <-
  M = 7
```

```
-> State: 1.6 <-
  M = 14
-> State: 1.7 <-
  M = 15
-> State: 1.8 <-
  M = 30
-> State: 1.9 <-
  M = 31
-> State: 1.10 <-
  M = 62
```

```
-> State: 1.11 <-
  M = 124
-> State: 1.12 <-
  M = 125
-> State: 1.13 <-
  M = 250
-> State: 1.14 <-
  M = 500
-> State: 1.15 <-
  M = 1000
```

How to solve this by SAT/SMT?

Fix number k: try for k steps

Introduce $M[i]$ to represent the number of marbles after i steps, for $i = 0, \ldots, k$

Start by one marble:

$$M[0] = 1$$

At the end exactly 1000 marbles:

$$M[k] = 1000$$

Requirements for the steps:

$$(M[i] = M[i-1] + 1) \vee (M[i] = 2M[i-1])$$

for $i = 1, \ldots, k$

How to solve this by SAT/SMT?

Fix number k: try for k steps

Introduce $M[i]$ to represent the number of marbles after i steps, for $i = 0, \ldots, k$

Start by one marble:

$$M[0] = 1$$

At the end exactly 1000 marbles:

$$M[k] = 1000$$

Requirements for the steps:

$$(M[i] = M[i-1] + 1) \vee (M[i] = 2M[i-1])$$

for $i = 1, \ldots, k$

How to solve this by SAT/SMT?

Fix number k: try for k steps

Introduce $M[i]$ to represent the number of marbles after i steps, for $i = 0, \ldots, k$

Start by one marble:

$$M[0] = 1$$

At the end exactly 1000 marbles:

$$M[k] = 1000$$

Requirements for the steps:

$$(M[i] = M[i-1] + 1) \vee (M[i] = 2M[i-1])$$

for $i = 1, \ldots, k$

How to solve this by SAT/SMT?

Fix number k: try for k steps

Introduce $M[i]$ to represent the number of marbles after i steps, for $i = 0, \ldots, k$

Start by one marble:

$$M[0] = 1$$

At the end exactly 1000 marbles:

$$M[k] = 1000$$

Requirements for the steps:

$$(M[i] = M[i-1] + 1) \vee (M[i] = 2M[i-1])$$

for $i = 1, \ldots, k$

How to solve this by SAT/SMT?

Fix number k: try for k steps

Introduce $M[i]$ to represent the number of marbles after i steps, for $i = 0, \ldots, k$

Start by one marble:

$$M[0] = 1$$

At the end exactly 1000 marbles:

$$M[k] = 1000$$

Requirements for the steps:

$$(M[i] = M[i-1] + 1) \vee (M[i] = 2M[i-1])$$

for $i = 1, \ldots, k$

How to solve this by SAT/SMT?

Fix number k: try for k steps

Introduce $M[i]$ to represent the number of marbles after i steps, for $i = 0, \ldots, k$

Start by one marble:

$$M[0] = 1$$

At the end exactly 1000 marbles:

$$M[k] = 1000$$

Requirements for the steps:

$$(M[i] = M[i-1] + 1) \vee (M[i] = 2M[i-1])$$

for $i = 1, \ldots, k$

Resulting formula

$$M[0] = 1 \wedge M[k] = 1000 \wedge$$

$$\bigwedge_{i=1}^{k} (M[i] = M[i-1] + 1) \vee (M[i] = 2M[i-1])$$

for $i = 1, \ldots, k$ is *satisfiable* if and only if there is a solution in k steps

For $k = 1, 2, \ldots, 13$, this formula is unsatisfiable

For $k = 14$, it yields the satisfying assignment

$$M[i] = 1, 2, 3, 6, 7, 14, 15, 30, 31, 62, 124, 125, 250, 500, 1000$$

for $i = 0, \ldots, 14$

So 14 is the smallest number of steps

Resulting formula

$$M[0] = 1 \wedge M[k] = 1000 \wedge$$

$$\bigwedge_{i=1}^{k} (M[i] = M[i-1] + 1) \vee (M[i] = 2M[i-1])$$

for $i = 1, \ldots, k$ is *satisfiable* if and only if there is a solution in k steps

For $k = 1, 2, \ldots, 13$, this formula is unsatisfiable

For $k = 14$, it yields the satisfying assignment

$$M[i] = 1, 2, 3, 6, 7, 14, 15, 30, 31, 62, 124, 125, 250, 500, 1000$$

for $i = 0, \ldots, 14$

So 14 is the smallest number of steps

Resulting formula

$$M[0] = 1 \wedge M[k] = 1000 \wedge$$

$$\bigwedge_{i=1}^{k} (M[i] = M[i-1] + 1) \vee (M[i] = 2M[i-1])$$

for $i = 1, \ldots, k$ is *satisfiable* if and only if there is a solution in k steps

For $k = 1, 2, \ldots, 13$, this formula is unsatisfiable

For $k = 14$, it yields the satisfying assignment

$$M[i] = 1, 2, 3, 6, 7, 14, 15, 30, 31, 62, 124, 125, 250, 500, 1000$$

for $i = 0, \ldots, 14$

So 14 is the smallest number of steps

Resulting formula

$$M[0] = 1 \wedge M[k] = 1000 \wedge$$

$$\bigwedge_{i=1}^{k} (M[i] = M[i-1] + 1) \vee (M[i] = 2M[i-1])$$

for $i = 1, \ldots, k$ is *satisfiable* if and only if there is a solution in k steps

For $k = 1, 2, \ldots, 13$, this formula is unsatisfiable

For $k = 14$, it yields the satisfying assignment

$$M[i] = 1, 2, 3, 6, 7, 14, 15, 30, 31, 62, 124, 125, 250, 500, 1000$$

for $i = 0, \ldots, 14$

So 14 is the smallest number of steps

Concluding: BMC 算法的优缺点

- 优点: runs fast in finding counterexamples
- 缺点: incompleteness: hard to return True if the property is satisfied

回顾: 定义: Transition system

A transition system $\mathcal{M} = (S, \rightarrow, L)$ is

- S: a set of states
- $\rightarrow$: a transition relation.
  - every $s \in S$ has some $s' \in S$ with $s \rightarrow s'$
- L: a label function.
  - $L : S \rightarrow \mathcal{P}(\text{Atoms})$

定义: Inductive

Denote s_0 as the initial state of a transition system. A state property P is *inductive*, iff

- P holds in the initial states, i.e., $L(s_0) \vDash P(s_0)$
- P holds in all states reachable from states that satisfy P, i.e.,

$$P(s), (s \rightarrow s') \vDash P(s')$$

回顾: 定义: Transition system

A transition system $\mathcal{M} = (S, \rightarrow, L)$ is

- S: a set of states
- $\rightarrow$: a transition relation.
  - every $s \in S$ has some $s' \in S$ with $s \rightarrow s'$
- L: a label function.
  - $L : S \rightarrow \mathcal{P}(\text{Atoms})$

定义: Inductive

Denote s_0 as the initial state of a transition system. A state property P is *inductive*, iff

- P holds in the initial states, i.e., $L(s_0) \vDash P(s_0)$
- P holds in all states reachable from states that satisfy P, i.e.,

$$P(s), (s \rightarrow s') \vDash P(s')$$

定义: Inductive

P is *inductive*, iff $L(s_0) \models P(s_0)$ and $P(s), (s \rightarrow s') \models P(s')$

```
MODULE main
VAR
  y : 0..15000;
ASSIGN
  init(y) := 0;
TRANS
  case
    y=70  : next(y)=0;
    TRUE : next(y)=y+1;
  esac
LTLSPEC
  G (y in (0..70))
```

- $L(s_0) = \{y = 0\}$, so $0 \leq y \leq 70$
- Assume $0 \leq y \leq 70$, is $0 \leq y' \leq 70$ satisfied?
  - if $y = 70$, $y' = 0$
  - if $0 \leq y < 70$, $0 < y' \leq 70$

So we can prove the property by hand

运行结果: ./NuSMV c-sample2-induct.smv
– specification G y in (0 .. 70) is true
Time cost: 4.26s

定义: Inductive

P is *inductive*, iff $L(s_0) \models P(s_0)$ and $P(s), (s \rightarrow s') \models P(s')$

```
MODULE main
VAR
  y : 0..15000;
ASSIGN
  init(y) := 0;
TRANS
  case
    y=70  : next(y)=0;
    TRUE  : next(y)=y+1;
  esac
LTLSPEC
  G (y in (0..70))
```

- $L(s_0) = \{y = 0\}$, so $0 \leq y \leq 70$
- Assume $0 \leq y \leq 70$, is $0 \leq y' \leq 70$ satisfied?
  - if $y = 70$, $y' = 0$
  - if $0 \leq y < 70$, $0 < y' \leq 70$

So we can prove the property by hand

运行结果: ./NuSMV c-sample2-induct.smv
– specification G y in (0 .. 70) is true
Time cost: 4.26s

定义: Inductive

P is *inductive*, iff $L(s_0) \models P(s_0)$ and $P(s), (s \rightarrow s') \models P(s')$

```
MODULE main
VAR
  y : 0..15000;
ASSIGN
  init(y) := 0;
TRANS
  case
    y=70  : next(y)=0;
    TRUE  : next(y)=y+1;
  esac
LTLSPEC
  G (y in (0..70))
```

- $L(s_0) = \{y = 0\}$, so $0 \leq y \leq 70$
- Assume $0 \leq y \leq 70$, is $0 \leq y' \leq 70$ satisfied?
  - if $y = 70$, $y' = 0$
  - if $0 \leq y < 70$, $0 < y' \leq 70$

So we can prove the property by hand

运行结果: ./NuSMV c-sample2-induct.smv
– specification G y in (0 .. 70) is true
Time cost: 4.26s

定义: Inductive

P is *inductive*, iff $L(s_0) \models P(s_0)$ and $P(s), (s \rightarrow s') \models P(s')$

```
MODULE main
VAR
  y : 0..15000;
ASSIGN
  init(y) := 0;
TRANS
  case
    y=70  : next(y)=0;
    TRUE  : next(y)=y+1;
  esac
LTLSPEC
  G (y in (0..70))
```

- $L(s_0) = \{y = 0\}$, so $0 \leq y \leq 70$
- Assume $0 \leq y \leq 70$, is $0 \leq y' \leq 70$ satisfied?
  - if $y = 70$, $y' = 0$
  - if $0 \leq y < 70$, $0 < y' \leq 70$

So we can prove the property by hand

运行结果: ./NuSMV c-sample2-induct.smv
– specification G y in (0 .. 70) is true
Time cost: 4.26s

定义: Inductive

P is *inductive*, iff $L(s_0) \models P(s_0)$ and $P(s), (s \to s') \models P(s')$

```
MODULE main
VAR
  y : 0..15000;
ASSIGN
  init(y) := 0;
TRANS
  case
    y=70  : next(y)=0;
    TRUE  : next(y)=y+1;
  esac
LTLSPEC
  G (y in (0..70))
```

- $L(s_0) = \{y = 0\}$, so $0 \le y \le 70$
- Assume $0 \le y \le 70$, is $0 \le y' \le 70$ satisfied?
  - if $y = 70$, $y' = 0$
  - if $0 \le y < 70$, $0 < y' \le 70$

So we can prove the property by hand

运行结果: ./NuSMV c-sample2-induct.smv
– specification G y in (0 .. 70) is true
Time cost: 4.26s

定义: Inductive

P is *inductive*, iff $L(s_0) \models P(s_0)$ and $P(s), (s \rightarrow s') \models P(s')$

```
MODULE main
VAR
  y : 0..15000;
ASSIGN
  init(y) := 0;
TRANS
  case
    y=70  : next(y)=0;
    TRUE  : next(y)=y+1;
  esac
LTLSPEC
  G (y in (0..70))
```

- $L(s_0) = \{y = 0\}$, so $0 \leq y \leq 70$
- Assume $0 \leq y \leq 70$, is $0 \leq y' \leq 70$ satisfied?
  - if $y = 70$, $y' = 0$
  - if $0 \leq y < 70$, $0 < y' \leq 70$

So we can prove the property by hand

运行结果: ./NuSMV c-sample2-induct.smv
– specification G y in (0 .. 70) is true
Time cost: 4.26s

定义: Inductive

P is *inductive*, iff $L(s_0) \models P(s_0)$ and $P(s), (s \rightarrow s') \models P(s')$

```
MODULE main
VAR
  y : 0..15000;
ASSIGN
  init(y) := 0;
TRANS
  case
    y=70  : next(y)=0;
    TRUE  : next(y)=y+1;
  esac
LTLSPEC
  G (y in (0..70))
```

- $L(s_0) = \{y = 0\}$, so $0 \leq y \leq 70$
- Assume $0 \leq y \leq 70$, is $0 \leq y' \leq 70$ satisfied?
  - if $y = 70$, $y' = 0$
  - if $0 \leq y < 70$, $0 < y' \leq 70$

So we can prove the property by hand

运行结果: ./NuSMV c-sample2-induct.smv
– specification G y in (0 .. 70) is true
Time cost: 4.26s

定义: Inductive

P is *inductive*, iff $L(s_0) \models P(s_0)$ and $P(s), (s \rightarrow s') \models P(s')$

```
MODULE main
VAR
  y : 0..15000;
ASSIGN
  init(y) := 0;
TRANS
  case
    y=70  : next(y)=0;
    TRUE  : next(y)=y+1;
  esac
LTLSPEC
  G (y in (0..70))
```

- $L(s_0) = \{y = 0\}$, so $0 \leq y \leq 70$
- Assume $0 \leq y \leq 70$, is $0 \leq y' \leq 70$ satisfied?
  - if $y = 70$, $y' = 0$
  - if $0 \leq y < 70$, $0 < y' \leq 70$

  So we can prove the property by hand

运行结果: ./NuSMV c-sample2-induct.smv
– specification G y in (0 .. 70) is true
Time cost: 4.26s

定义: Inductive

P is *inductive*, iff $L(s_0) \vDash P(s_0)$ and $P(s), (s \to s') \vDash P(s')$

```
MODULE main
VAR
  y : 0..15000;
ASSIGN
  init(y) := 0;
TRANS
  case
    y=70  : next(y)=0;
    TRUE : next(y)=y+1;
  esac
LTLSPEC
  G (y in (0..70))
```

尝试其它实验:

- G (y in (0..69))
  - return false
- G (y in (0..71))
  - *NOT* inductive now
- for all 70, change into 700
  - The extra time cost is *low*
- change 15000 into 150000
  - The extra time cost is *extremely high*

定义: Inductive

P is *inductive*, iff $L(s_0) \vDash P(s_0)$ and $P(s), (s \rightarrow s') \vDash P(s')$

```
MODULE main
VAR
  y : 0..15000;
ASSIGN
  init(y) := 0;
TRANS
  case
    y=70  : next(y)=0;
    TRUE : next(y)=y+1;
  esac
LTLSPEC
  G (y in (0..70))
```

尝试其它实验:

- G (y in (0..69))
  - return false
- G (y in (0..71))
  - *NOT* inductive now
- for all 70, change into 700
  - The extra time cost is *low*
- change 15000 into 150000
  - The extra time cost is *extremely high*

定义: Inductive

P is *inductive*, iff $L(s_0) \models P(s_0)$ and $P(s), (s \rightarrow s') \models P(s')$

```
MODULE main
VAR
  y : 0..15000;
ASSIGN
  init(y) := 0;
TRANS
  case
    y=70  : next(y)=0;
    TRUE : next(y)=y+1;
  esac
LTLSPEC
  G (y in (0..70))
```

尝试其它实验:

- G (y in (0..69))
  - return false
- G (y in (0..71))
  - *NOT* inductive now
- for all 70, change into 700
  - The extra time cost is *low*
- change 15000 into 150000
  - The extra time cost is *extremely high*

定义: Inductive

P is *inductive*, iff $L(s_0) \models P(s_0)$ and $P(s), (s \rightarrow s') \models P(s')$

```
MODULE main
VAR
  y : 0..15000;
ASSIGN
  init(y) := 0;
TRANS
  case
    y=70  : next(y)=0;
    TRUE : next(y)=y+1;
  esac
LTLSPEC
  G (y in (0..70))
```

尝试其它实验:

- G (y in (0..69))
  - return false
- G (y in (0..71))
  - *NOT* inductive now
- for all 70, change into 700
  - The extra time cost is *low*
- change 15000 into 150000
  - The extra time cost is *extremely high*

定义: Inductive

P is *inductive*, iff $L(s_0) \models P(s_0)$ and $P(s), (s \rightarrow s') \models P(s')$

```
MODULE main
VAR
  y : 0..15000;
ASSIGN
  init(y) := 0;
TRANS
  case
    y=70   : next(y)=0;
    TRUE : next(y)=y+1;
  esac
LTLSPEC
  G (y in (0..70))
```

尝试其它实验:

- G (y in (0..69))
  - return false
- G (y in (0..71))
  - *NOT* inductive now
- for all 70, change into 700
  - The extra time cost is *low*
- change 15000 into 150000
  - The extra time cost is *extremely high*

定义: Inductive

P is *inductive*, iff $L(s_0) \models P(s_0)$ and $P(s), (s \rightarrow s') \models P(s')$

```
MODULE main
VAR
  y : 0..15000;
ASSIGN
  init(y) := 0;
TRANS
  case
    y=70  : next(y)=0;
    TRUE  : next(y)=y+1;
  esac
LTLSPEC
  G (y in (0..70))
```

尝试其它实验:

- G (y in (0..69))
  - return false
- G (y in (0..71))
  - *NOT* inductive now
- for all 70, change into 700
  - The extra time cost is *low*
- change 15000 into 150000
  - The extra time cost is *extremely high*

定义: Inductive

P is *inductive*, iff $L(s_0) \models P(s_0)$ and $P(s), (s \rightarrow s') \models P(s')$

```
MODULE main
VAR
  y : 0..15000;
ASSIGN
  init(y) := 0;
TRANS
  case
    y=70  : next(y)=0;
    TRUE  : next(y)=y+1;
  esac
LTLSPEC
  G (y in (0..70))
```

尝试其它实验:

- G (y in (0..69))
  - return false
- G (y in (0..71))
  - *NOT* inductive now
- for all 70, change into 700
  - The extra time cost is *low*
- change 15000 into 150000
  - The extra time cost is *extremely high*

回顾定义: Inductive

P is *inductive*, iff $L(s_0) \vDash P(s_0)$ and $P(s), (s \to s') \vDash P(s')$

一种简单的解释 (standard induction over *natural numbers*):

$$P(0) \wedge \forall n(P(n) \to P(n+1)) \vDash \forall n P(n)$$

2-induction

$$P(0) \wedge P(1) \wedge \forall n((P(n) \wedge P(n+1)) \to P(n+2)) \vDash \forall n P(n)$$

扩充定义: *k-induction*

$$\left(\bigwedge_{i=0}^{k-1} P(i)\right) \wedge \forall n \left(\left(\bigwedge_{i=0}^{k-1} P(n+i)\right) \to P(n+k)\right) \vDash \forall n P(n)$$

回顾定义: Inductive

P is *inductive*, iff $L(s_0) \vDash P(s_0)$ and $P(s), (s \to s') \vDash P(s')$

一种简单的解释 (standard induction over *natural numbers*):

$$P(0) \wedge \forall n(P(n) \to P(n+1)) \vDash \forall n P(n)$$

2-induction

$$P(0) \wedge P(1) \wedge \forall n((P(n) \wedge P(n+1)) \to P(n+2)) \vDash \forall n P(n)$$

扩充定义: *k-induction*

$$\left(\bigwedge_{i=0}^{k-1} P(i)\right) \wedge \forall n \left(\left(\bigwedge_{i=0}^{k-1} P(n+i)\right) \to P(n+k)\right) \vDash \forall n P(n)$$

回顾定义: Inductive

P is *inductive*, iff $L(s_0) \vDash P(s_0)$ and $P(s), (s \to s') \vDash P(s')$

一种简单的解释 (standard induction over *natural numbers*):

$$P(0) \wedge \forall n(P(n) \to P(n+1)) \vDash \forall n P(n)$$

2-induction

$$P(0) \wedge P(1) \wedge \forall n((P(n) \wedge P(n+1)) \to P(n+2)) \vDash \forall n P(n)$$

扩充定义: *k-induction*

$$\left(\bigwedge_{i=0}^{k-1} P(i)\right) \wedge \forall n \left(\left(\bigwedge_{i=0}^{k-1} P(n+i)\right) \to P(n+k)\right) \vDash \forall n P(n)$$

回顾定义: Inductive

P is *inductive*, iff $L(s_0) \vDash P(s_0)$ and $P(s), (s \to s') \vDash P(s')$

一种简单的解释 (standard induction over *natural numbers*):

$$P(0) \wedge \forall n(P(n) \to P(n+1)) \vDash \forall n P(n)$$

2-induction

$$P(0) \wedge P(1) \wedge \forall n((P(n) \wedge P(n+1)) \to P(n+2)) \vDash \forall n P(n)$$

扩充定义: *k-induction*

$$\left(\bigwedge_{i=0}^{k-1} P(i)\right) \wedge \forall n \left(\left(\bigwedge_{i=0}^{k-1} P(n+i)\right) \to P(n+k)\right) \vDash \forall n P(n)$$

问题: Is k-induction "better" than standard induction?
Maybe yes.

例: Consider the Fibonacci sequence, defined by

$$fib(n) = \begin{cases} n & \text{if } n \leq 1 \\ fib(n-1) + fib(n-2) & \text{otherwise} \end{cases}$$

Suppose we want to prove $fib(n) \geq n$ for $n \geq 5$

- 1-induction? No
- 2-induction? Yes

问题: Is k-induction "better" than standard induction?
Maybe yes.

例: Consider the Fibonacci sequence, defined by

$$fib(n) = \begin{cases} n & \text{if } n \leq 1 \\ fib(n-1) + fib(n-2) & \text{otherwise} \end{cases}$$

Suppose we want to prove $fib(n) \geq n$ for $n \geq 5$

- 1-induction? No
- 2-induction? Yes

问题: Is k-induction "better" than standard induction?
Maybe yes.

例: Consider the Fibonacci sequence, defined by

$$fib(n) = \begin{cases} n & \text{if } n \leq 1 \\ fib(n-1) + fib(n-2) & \text{otherwise} \end{cases}$$

Suppose we want to prove $fib(n) \geq n$ for $n \geq 5$

- 1-induction? No
- 2-induction? Yes