All-Termination(SCP)

Aaron Turon
Northeastern University
turon@ccs.neu.edu

(joint work with Pete Manolios)

"Why, sometimes I've believed as many as six impossible things before breakfast."

-- Queen of Hearts, Alice in Wonderland

"Why, sometimes I've believed as many as six impossible things before breakfast."

-- Queen of Hearts, Alice in Wonderland

Slogan 0

All-Termination:

how to solve six impossible problems before lunch

```
(define (insert i x xs)
  (cond
    ((<= i 0)      (cons x xs))
    ((empty? xs) (list x))
    (else          (cons (car xs)
                     (insert (- i 1)
                             x
                             (cdr xs))))))
```

```
(define (insert i x xs)
  (cond
    ((<= i 0)      (cons x xs))
    ((empty? xs) (list x))
    (else          (cons (car xs)
                     (insert (- i 1)
                             x
                             (cdr xs))))))
```

How do we prove that **insert** terminates?

```
(define (insert i x xs)
  (cond
    ((<= i 0)      (cons x xs))
    ((empty? xs) (list x))
    (else          (cons (car xs)
                     (insert (- i 1)
                             x
                             (cdr xs))))))
```

How do we prove that **insert** terminates?

$m_1(i, x, xs) = i$

```
(define (insert i x xs)
  (cond
    ((<= i 0)      (cons x xs))
    ((empty? xs) (list x))
    (else          (cons (car xs)
                     (insert (- i 1)
                             x
                             (cdr xs))))
```

How do we prove that **insert** terminates?

$m_1(i, x, xs) = i$

$m_2(i, x, xs) = \text{length } xs$

```
(define (insert i x xs)
  (cond
    ((<= i 0)      (cons x xs))
    ((empty? xs) (list x))
    (else          (cons (car xs)
                     (insert (- i 1)
                             x
                             (cdr xs))))
```

How do we prove that **insert** terminates?

$$m_1(i, x, xs) = i$$

$$m_2(i, x, xs) = \text{length } xs$$

$$m_3(i, x, xs) = i + \text{length } xs$$

```
(define (insert i x xs)
  (cond
    ((<= i 0)      (cons x xs))
    ((empty? xs) (list x))
    (else          (cons (car xs)
                     (insert (- i 1)
                             x
                             (cdr xs))))))
```

How do we prove that **insert** terminates?

```
m1(i, x, xs) = i
m2(i, x, xs) = length xs
m3(i, x, xs) = i + length xs
```

```
(define (insert i x xs)
  (cond
    ((<= i 0)     (cons x xs))
    ((empty? xs) (list x))
    (else         (cons (car xs)
                    (insert (- i 1)
                            x
                            (cdr xs))))))
```

How do we prove that `insert` terminates?

Measured sets
for `insert`

```
{i}
{xs}
{i, xs}
```

Measured sets ↣ induction schemes [*Boyer&Moore, 1979*]

To prove $\forall i,x,xs :: \varphi(i,x,xs)$, show:

Measured sets ↣ induction schemes [*Boyer&Moore, 1979*]

To prove $\forall i,x,xs :: \varphi(i,x,xs)$, show:

Measured set {i}

$\varphi(\mathbf{0},x,xs)$

$[\forall y,ys :: \varphi(\mathbf{i},y,ys)] \Rightarrow \varphi(\mathbf{i+1},x,xs)$

Measured sets ↣ induction schemes [*Boyer&Moore, 1979*]

To prove $\forall i,x,xs :: \varphi(i,x,xs)$, show:

Measured set {i}

$\varphi(\mathbf{0},x,xs)$

$[\forall y,ys :: \varphi(\mathbf{i},y,ys)] \Rightarrow \varphi(\mathbf{i+1},x,xs)$

Measured set {xs}

$\varphi(i,x,\mathbf{nil})$

$[\forall j,y :: \varphi(j,y,\mathbf{xs})] \Rightarrow \varphi(i,x,\mathbf{cons(z,xs)})$

Measured sets ↣ induction schemes [*Boyer&Moore, 1979*]

To prove $\forall i,x,xs :: \varphi(i,x,xs)$, show:

Measured set {i}

$\varphi(\mathbf{0},x,xs)$

$[\forall y,ys :: \varphi(\mathbf{i},y,ys)] \Rightarrow \varphi(\mathbf{i+1},x,xs)$

Measured set {xs}

$\varphi(i,x,\mathbf{nil})$

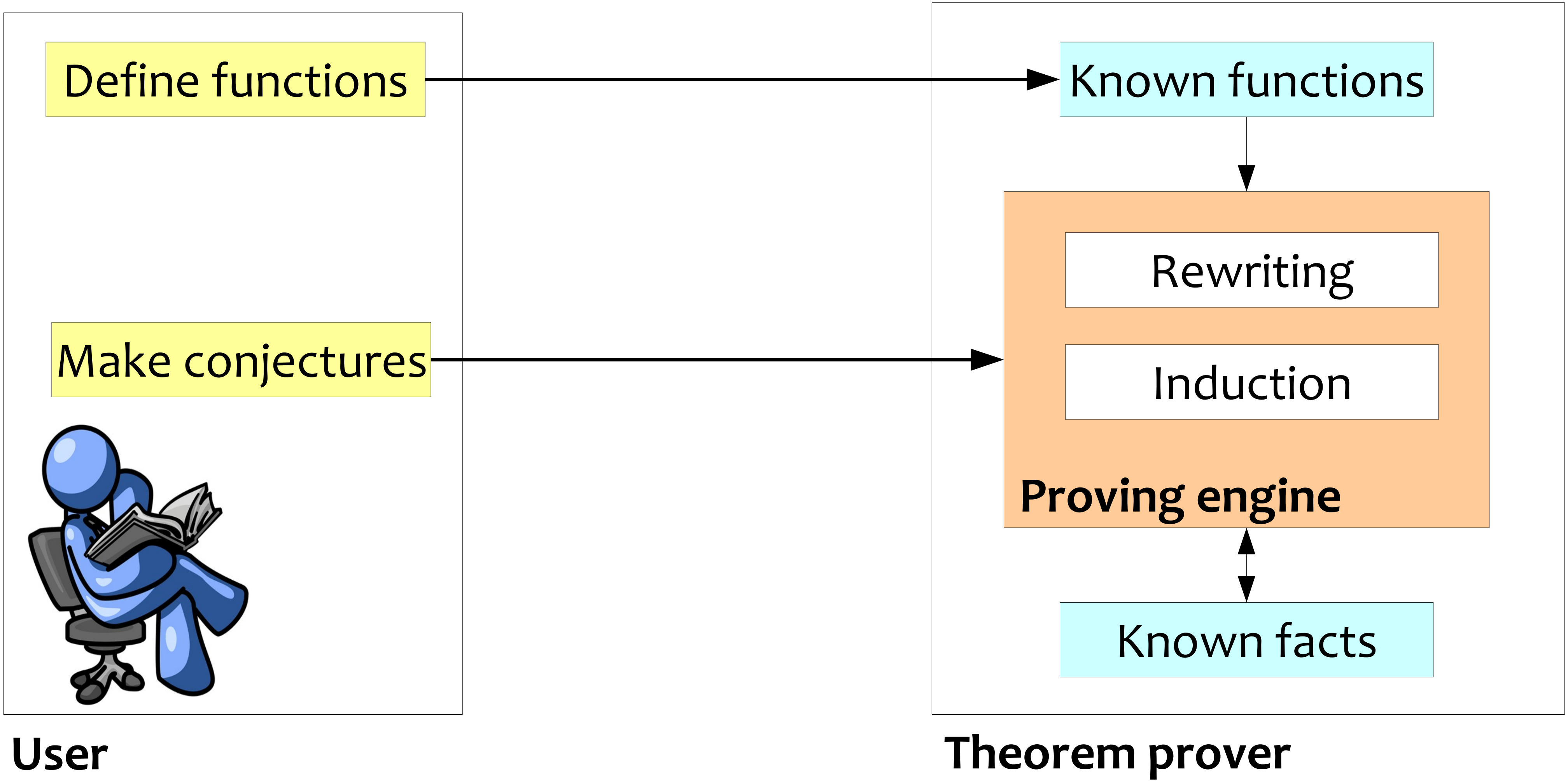
$[\forall j,y :: \varphi(j,y,\mathbf{xs})] \Rightarrow \varphi(i,x,\mathbf{cons(z,xs)})$

Measured set {i,xs}

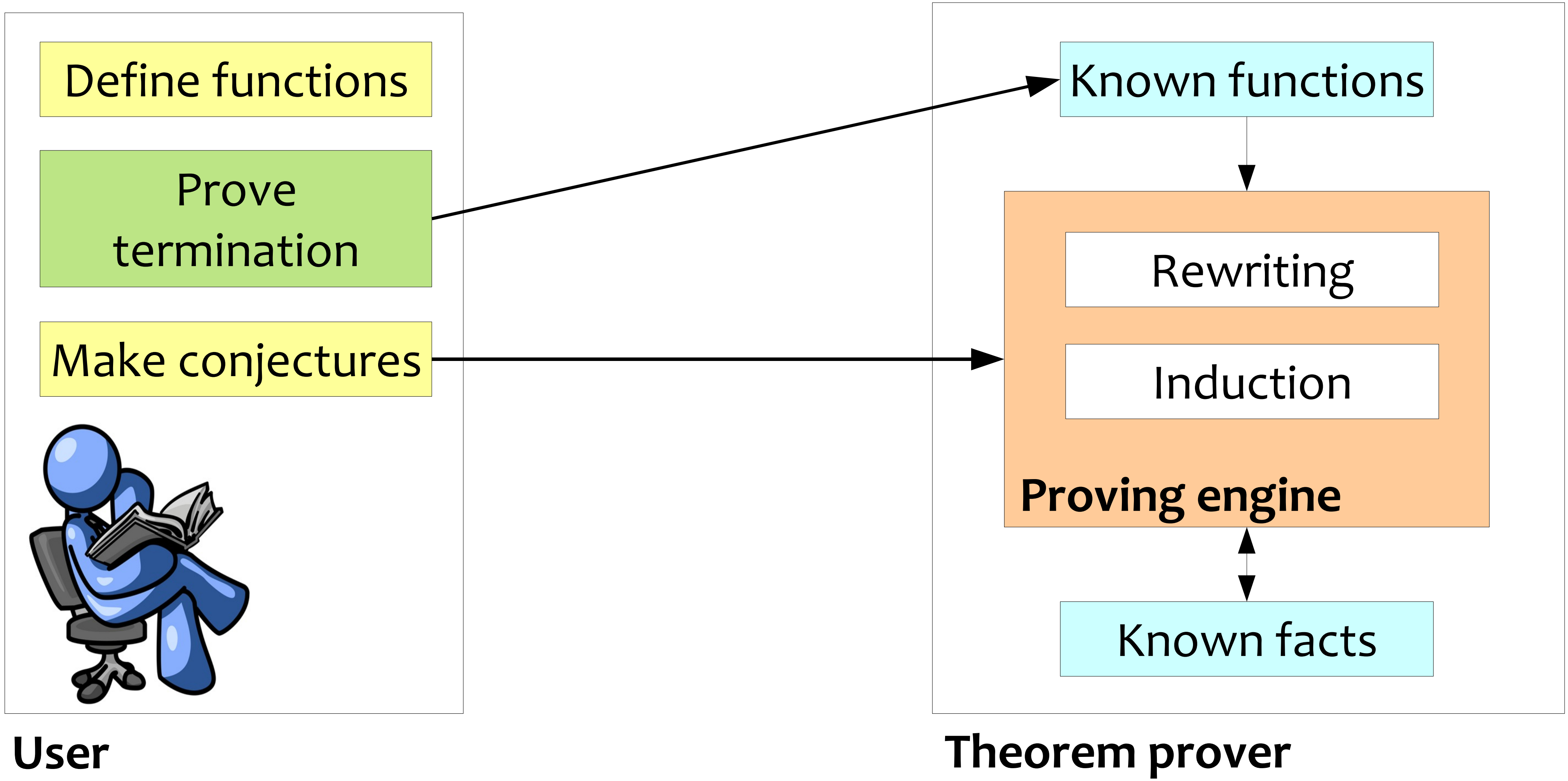
$\varphi(\mathbf{0},x,\mathbf{nil})$

$[\forall y :: \varphi(\mathbf{i},y,\mathbf{xs})] \Rightarrow \varphi(\mathbf{i+1},x,\mathbf{cons(z,xs)})$

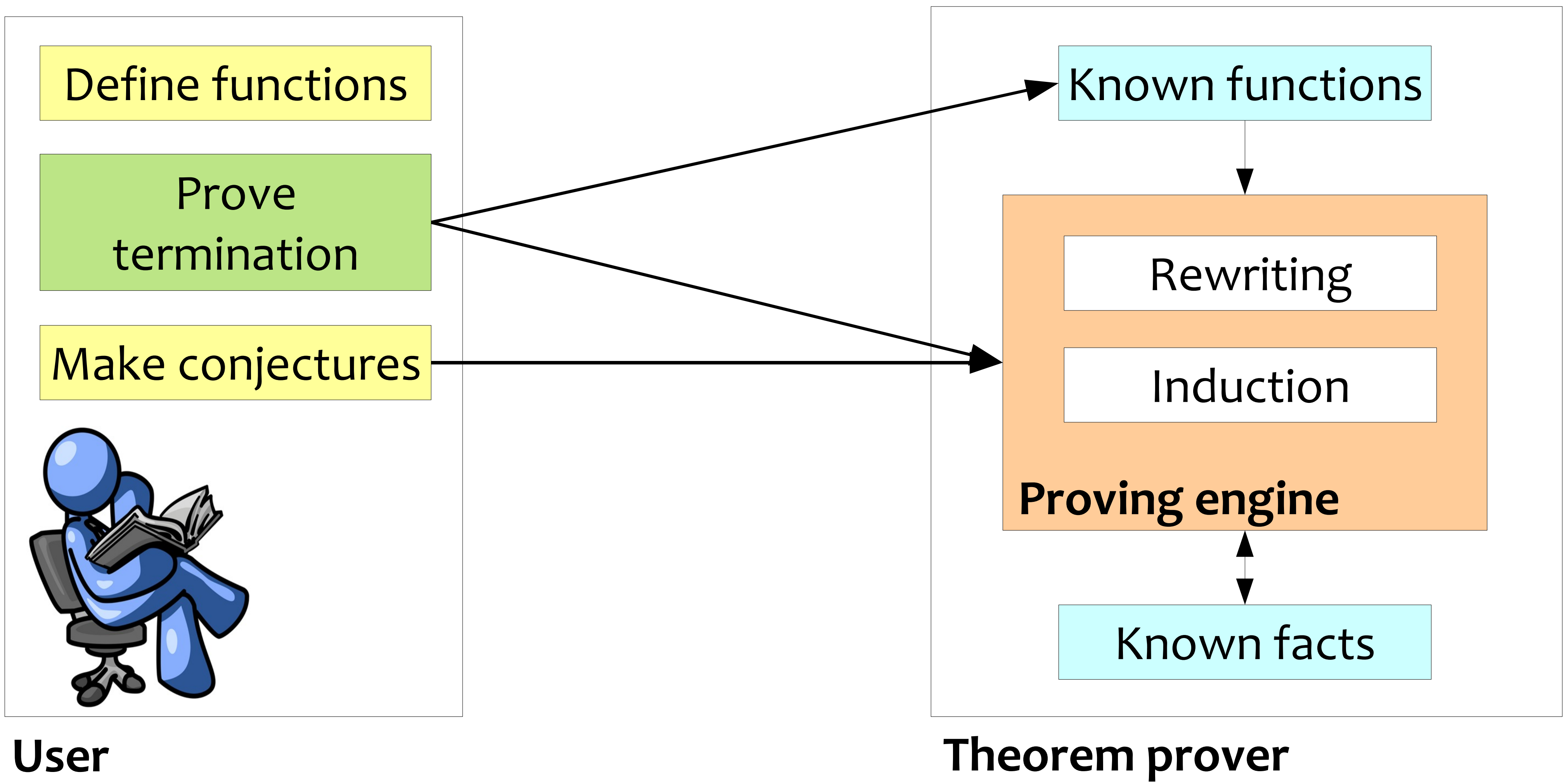
Life with a theorem prover:



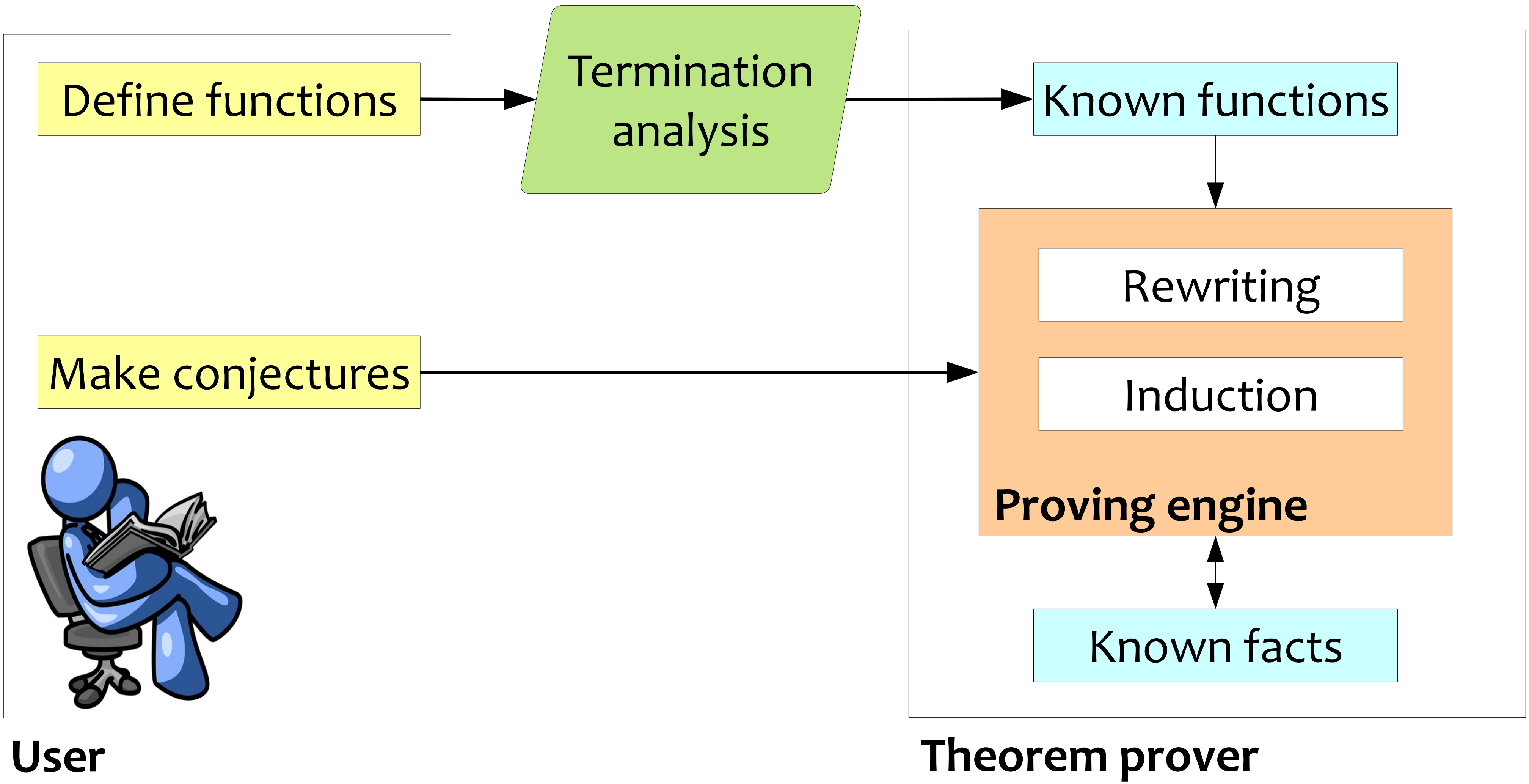
Life with a theorem prover:



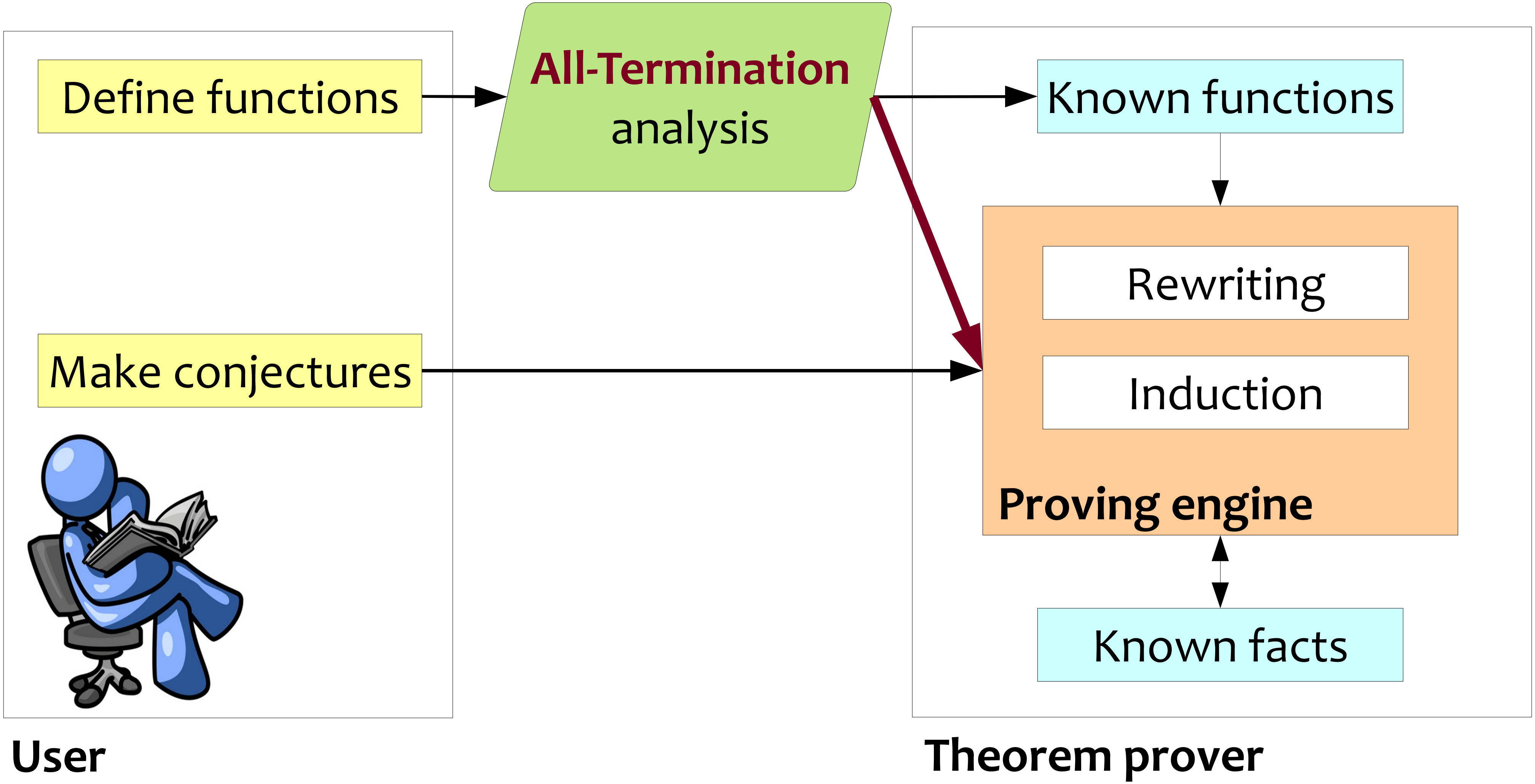
Life with ACL2:



Life with ACL2 Sedan:



A better life ACL2 Sedan:



Slogan 1

Termination is not a yes/no question –
it's multiple choice

The rest of the talk:

All-Termination(*T*)

- **definition**
- **research program**

Poly-time size-change termination (SCP)

All-Termination(SCP)

Termination analysis

Termination undecidable

Sound, incomplete analyses:

> **T** : Programs → Bool predicate such that
> if **T**(**P**) then **P** terminates on all inputs

Termination analysis

Termination undecidable

Sound, incomplete analyses:

T : Programs → Bool predicate such that
if **T**(**P**) then **P** terminates on all inputs

Restricted termination analysis

$$\mathbf{T} : \text{Programs} \times 2^{\text{Variables}} \to \text{Bool}$$

such that

if **T**(**P**,**V**) then **V** is a measured set for **P**

Measured sets are upward-closed:

if **U** $\subseteq$ **V** and **U** is a measured set for P then so is **V**

Measured sets are upward-closed:

if **U** $\subseteq$ **V** and **U** is a measured set for P then so is **V**

All-Termination(T) analysis

$$\text{All-Termination}(\mathbf{T})(\mathbf{P}) \overset{\text{def}}{=} \mathit{minimal}\{\mathbf{V} \mid \mathbf{T}(\mathbf{P},\mathbf{V})\}$$

where **T** is a restricted termination analysis.

Measured sets are upward-closed:

if $\mathbf{U} \subseteq \mathbf{V}$ and **U** is a measured set for P then so is **V**

All-Termination(T) analysis

$$\text{All-Termination}(\mathbf{T})(\mathbf{P}) \stackrel{\text{def}}{=} \mathit{minimal}\{\mathbf{V} \mid \mathbf{T}(\mathbf{P},\mathbf{V})\}$$

where **T** is a restricted termination analysis.

Warning

$|\text{All-Termination}(\mathbf{T})(\mathbf{P})|$ can be exponential in $|\mathbf{P}|$.

Theorem:

if **T** is in PSPACE then **AllTermination(T)** is in PSPACE.

Proof:

```
All-Termination(T)(P):
  for each V ⊆ vars(P)
     if T(P,V) then
        minimal := true
        for each U ⊊ V
           if T(P,U) then minimal := false
        if minimal then output(V)
```

Research program

Begin with standard termination analysis, **A**

Define restricted version, **T**, so that

$$[\exists\ \mathbf{V} :: \mathbf{T}(\mathbf{P},\mathbf{V})] \Leftrightarrow \mathbf{A}(\mathbf{P})$$

Instrument **A** to produce a "certificate"

Implement All-Termination(**T**)(**P**) by

- running **A** on **P** to produce certificate
- *extracting* measured sets from certificate

Slogan 2

All-Termination does not increase power –
it enriches results

The rest of the talk:

All-Termination(T)

Poly-time size-change termination (SCP)

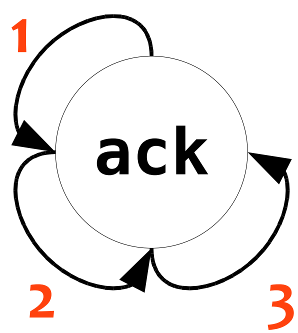
All-Termination(SCP)

Size-change termination [*Lee et al, POPL01*] works by analyzing a safe abstraction of the program.

```
ack(0,n) = n+1
ack(m,0) = ¹ack(m-1, 1)
ack(m,n) = ²ack(m-1, ³ack(m, n-1))
```

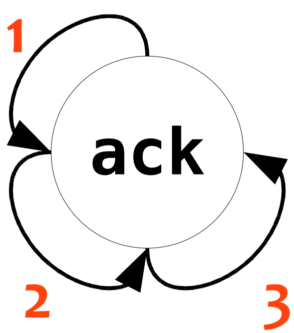
Size-change termination [*Lee et al, POPL01*] works by analyzing a safe abstraction of the program.

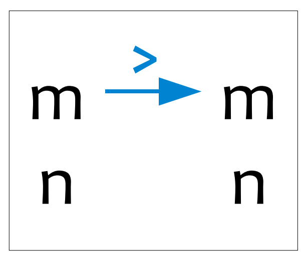
```
ack(0,n) = n+1
ack(m,0) = 1ack(m-1, 1)
ack(m,n) = 2ack(m-1, 3ack(m, n-1))
```



Size-change termination [*Lee et al, POPL01*] works by analyzing a safe abstraction of the program.

```
ack(0,n) = n+1
ack(m,0) = ¹ack(m-1, 1)
ack(m,n) = ²ack(m-1, ³ack(m, n-1))
```

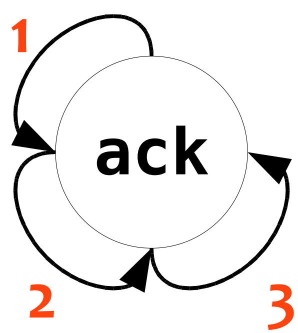


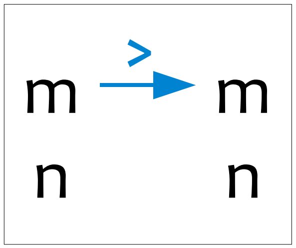


1

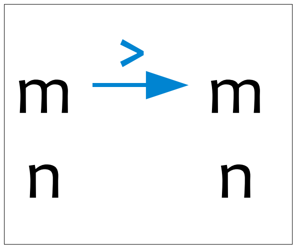
Size-change termination [*Lee et al, POPL01*] works by analyzing a safe abstraction of the program.

```
ack(0,n) = n+1
ack(m,0) = ¹ack(m-1, 1)
ack(m,n) = ²ack(m-1, ³ack(m, n-1))
```





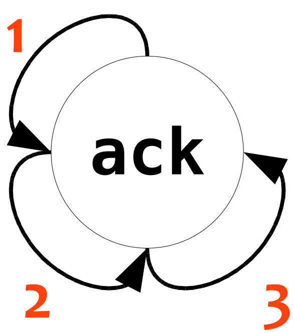
1

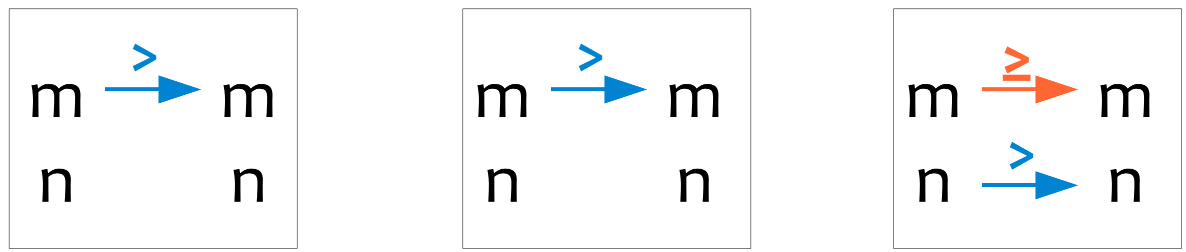


2

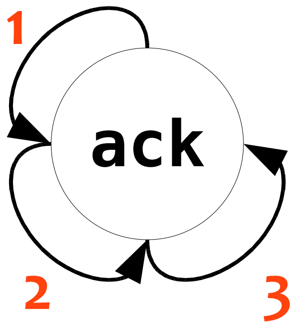
Size-change termination [*Lee et al, POPL01*] works by analyzing a safe abstraction of the program.

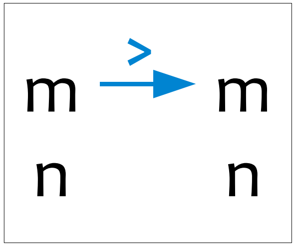
```
ack(0,n) = n+1
ack(m,0) = [1]ack(m-1, 1)
ack(m,n) = [2]ack(m-1, [3]ack(m, n-1))
```



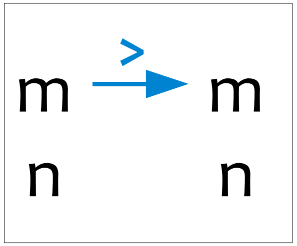


1 2 3

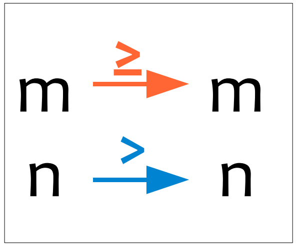




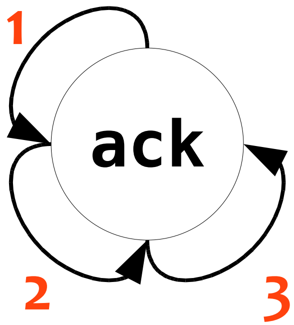
1

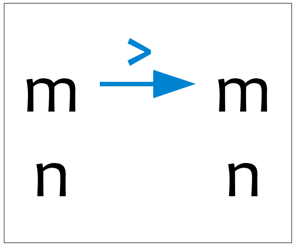


2

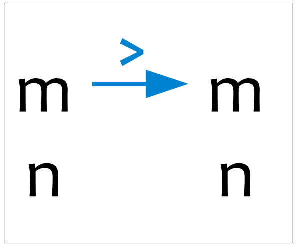


3

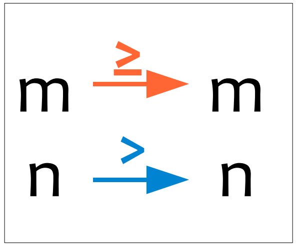




1



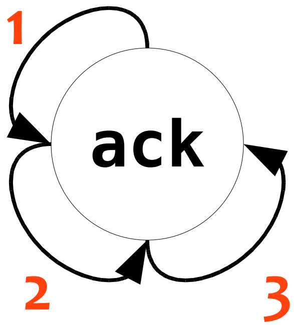
2

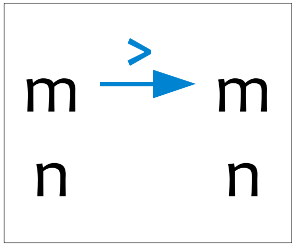


3

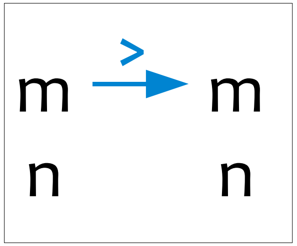
3 2 2 2 3 3 3 3 3 ...

Call sequence

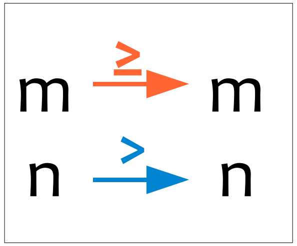




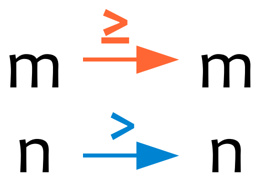
1



2

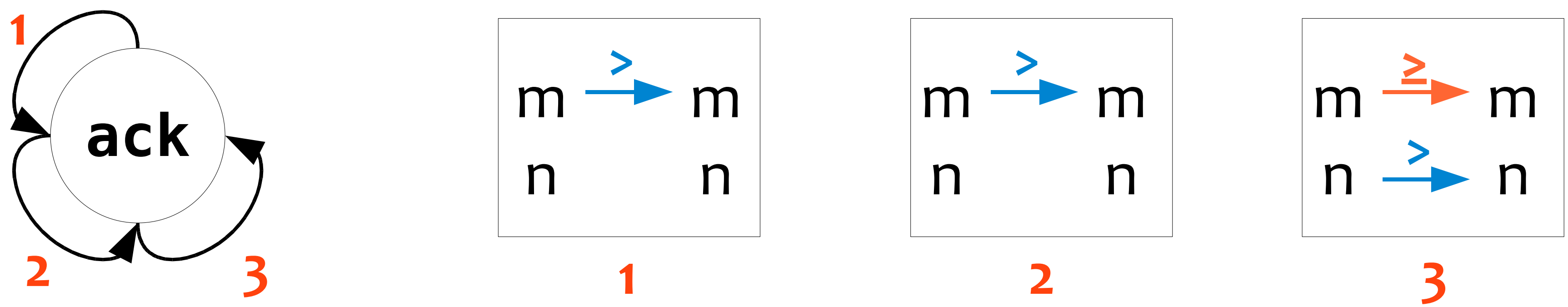


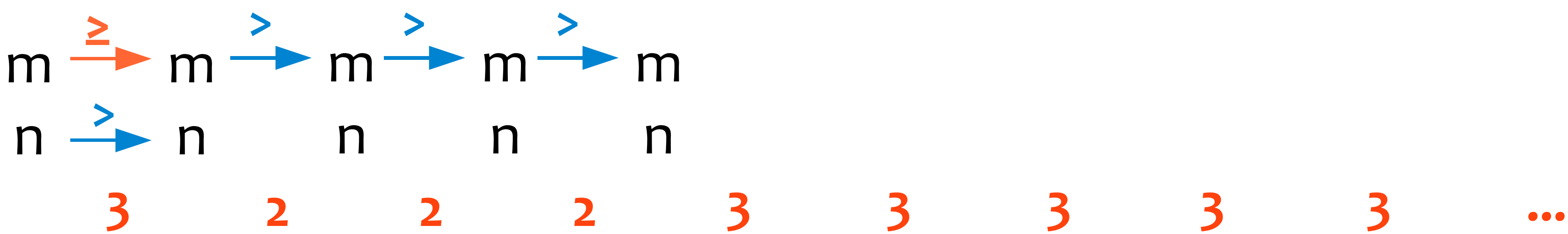
3



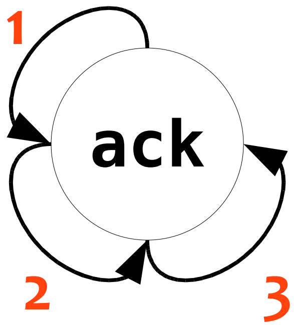
3 2 2 2 3 3 3 3 3 ...

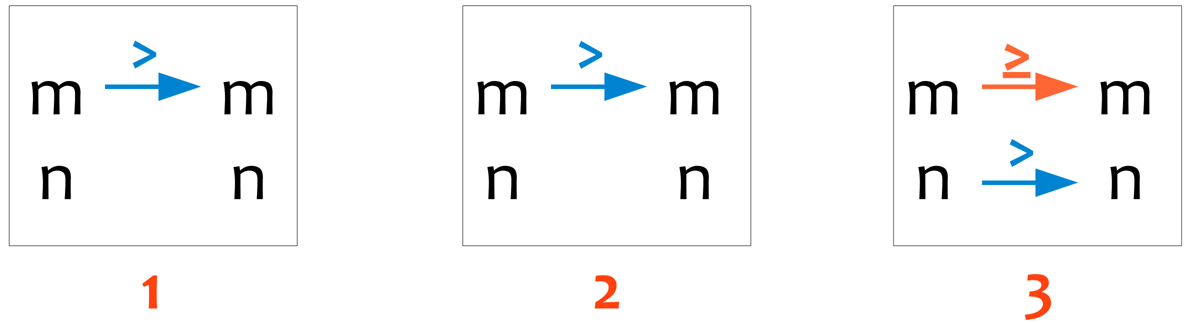
Call sequence

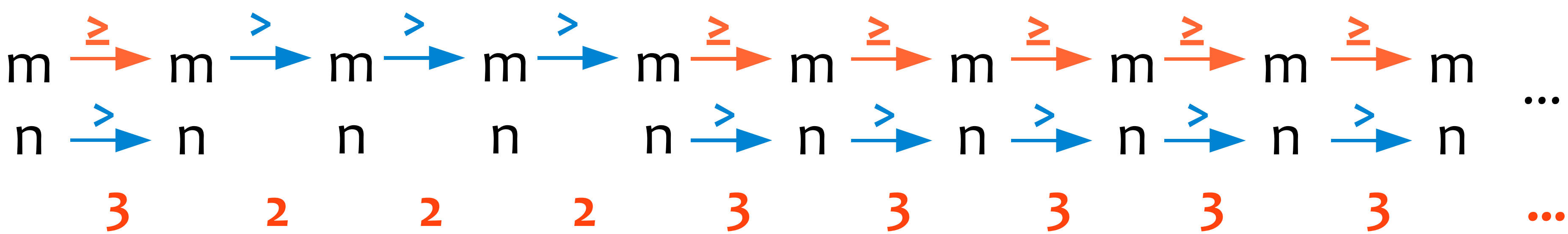




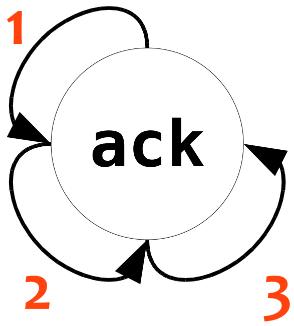
Call sequence

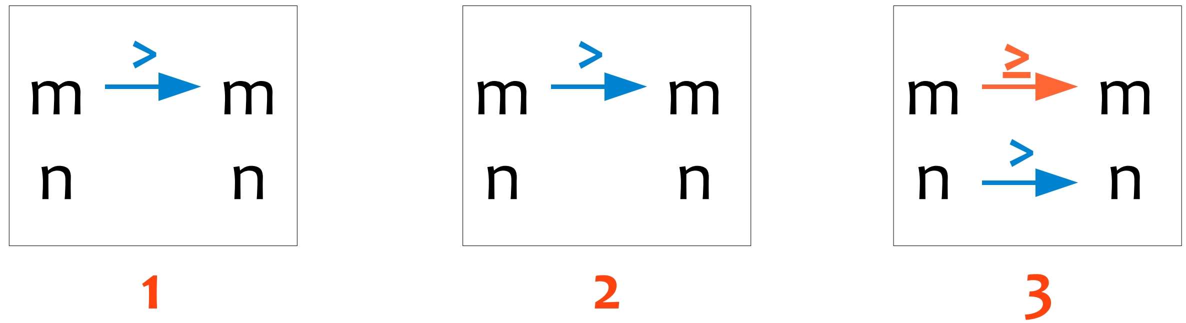


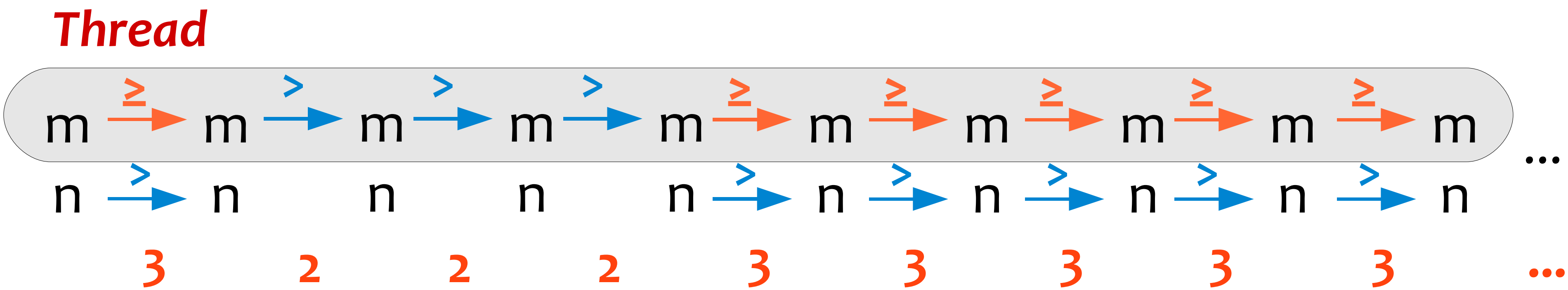




Call sequence







Call sequence

Program **P** is ***size-change terminating*** for graph **G** if:

each infinite path in **G** has a thread w/ infinite descent

Theorem [Lee et al, POPL01]

Deciding size-change termination is PSPACE-complete

Poly-time size-change analysis

Call site **C** is an ***anchor*** iff:

Passing through **C** infinitely often entails infinite descent

Algorithm [Ben-Amram, Lee 2007]**:**

```
SCP(G):
  for each H in SCC(G)
    A := FindAnchors(H)
    if empty(A) or SCP(H-A) = false
      then return false
  return true
```

The rest of the talk:

All-Termination(T)

Poly-time size-change termination (SCP)

All-Termination(SCP)

A (naïve!) restricted version of SCP

Let *restrict*(**G**, **V**) be **G**, but with only size-change edges relating variables in **V**.

Theorem: if

1. **G** is a valid annotated call graph for **P**
2. **SCP**(*restrict*(**G**,**V**))

then **V** is a measured subset for **P**.

Good, but ...

Good, but ...

Theorem:

Deciding $\exists$**V** :: **SCP**(*restrict*(**G**,**V**)) is NP-hard.

Good, but ...

Theorem:

Deciding ∃**V** :: **SCP**(*restrict*(**G**,**V**)) is NP-hard.

Corollary:

It is ***not*** true that **SCP**(**G**) iff ∃**V** :: **SCP**(*restrict*(**G**,**V**)).

What's going on?

Good, but ...

Theorem:

Deciding ∃**V** :: **SCP**(*restrict*(**G**,**V**)) is NP-hard.

Corollary:

It is ***not*** true that **SCP**(**G**) iff ∃**V** :: **SCP**(*restrict*(**G**,**V**)).

What's going on?

Non-monotonicity:

V ⊆ **W** and **SCP**(*restrict*(**G**,**V**)) does ***not*** imply **SCP**(*restrict*(**G**,**W**))

Slogan 3

Nonmonotonicity means trouble

What we do

- Instrument SCP to produce an ***anchor tree***
- Anchor tree is a certificate of termination
- Transform anchor tree to boolean constraint system φ
- φ captures which variables are *required* for the termination proof
  - small thread preservers are allowed!
- $|\varphi| = O(\ |G|\ )$

Finding the minimal solutions

Constraints φ are *dual-horn:* can find ψ that is

- equisatisfiable to φ
- conjunction of clauses,
- each clause a disjunction of literals
- at most one *negative* literal per clause

min solutions to φ can be found from ψ efficiently

Theorem:

After computing φ, we can find k elements of **All-Termination(SCP)(P)** in time $O(|G|^k)$

Pay-as-you-go algorithm

Slogan 4

To win, instrument and extract

Preliminary experimental results

ACL2 has a large regression suite:

- >100MB
- >11,000 function definitions (each of which must be proved terminating)

Code ranging from bit-vector libraries to model checkers

Preliminary experimental results

We have implemented, for ACL2,

- Poly-time size-change (SCP)
- Exp-time size-change (SCT)
- All-Termination(SCT)

We have ***not*** yet implemented

- All-Termination(SCP)

Preliminary experimental results

The setup: we ran CCG + All-Termination(SCT) on the entire regression suite.

Number of functions:	**>11,000**	
Proved terminating:	**98%**	***(note: same as CCG+SCT)***

Multiargument functions:

Proved terminating	1728
With "nontrivial" cores	90%
With multiple cores	7%
Maximum core count	3

Running time (not including CCG): 30 seconds

Preliminary experimental results

The setup: we ran CCG + All-Termination(SCT) on the entire regression suite.

Number of functions:	>11,000	
Proved terminating:	98%	(*note: same as* CCG+SCT)

Multiargument functions:

Proved terminating	**1728**
With "nontrivial" cores	**90%**
With multiple cores	**7%**
Maximum core count	**3**

Running time (not including CCG): 30 seconds

Preliminary experimental results

The setup: we ran CCG + All-Termination(SCT) on the entire regression suite.

Number of functions: >11,000
Proved terminating: 98% (*note: same as* CCG+SCT)

Multiargument functions:

- Proved terminating 1728
- **With "nontrivial" cores 90%**
- With multiple cores 7%
- **Maximum core count 3 (*the* k *parameter*)**

Running time (not including CCG): 30 seconds

Future work

Implement **All-Termination(SCP)**

Extend our prototype to the ACL2 Sedan

- Will help our freshman users at Northeastern

Study **All-Termination(*T*)** for additional ***T***

- e.g. dependency-pair termination analysis

Explore new applications of measured subsets

- We've got a few in mind, but want to hear yours

Contribution recap

- Proposed the All-Termination(*T*) problem
- Studied All-Termination(SCP)

Slogan recap

- Termination is not a yes/no question
- All-termination increases richness, not power
- Nonmonotonicity means trouble
- To win, instrument and extract