TATA CONSULTANCY SERVICES
Experience certainty.

Verifying Array Manipulating Programs

Thanks to Shrawan Kumar

Array manipulating programs

- Program manipulates/access array elements in a loop
  - Finding minimum
  - Searching for a value
  - Initialising/Copying/reversing
  - …

```
#define N 100000
int a[N]

for(i=0; i< N; i++)
    if (i != N/2)
       a[i] = 42

for (i=0; i< N; i++)
    if (i != N/2)
        assert(a[i]==42)
```

What is the difficulty?

- Bounded model checkers do not scale (either want of memory or time)
- Theorem provers will need loop invariant
  - $\forall i. 0 \leq i < N \wedge i \neq \frac{N}{2} \Longrightarrow a[i] = 42$
  - Getting such quantified loop invariant is hard

A different idea

- Can we analyze the loop for a few elements and extrapolate
- When can we do that?
- If it is possible then program can be transformed accordingly
- A bounded model checker then can be pressed into service

An example

```
int i, m;
int a[N]  ={2,4,6,8,11,2,2}
m = a[0]
i=0

while(i < N)
   if  (m >= a[i]-1)      m = a[i]
   i++

assert  ∀j ∈ [0..N-1] . (m <= a[j])
```

- Intended to compute minimum
- Property being checked accordingly
- Erroneously min is checked against a[i]-1 instead of a[i]
- It computes the last value in longest subsequence a[i1], a[i2],…
  - Such that a[i1]=min and a[ik+1]<= a[ik]+1
- Here min=2 and such a sequence is [2,2,2] with m being computed as 2
- Therefore property holds
- However changing last element to 3 will make the sequence as [2,2,3] with m being 3 and property will fail

An example

```
int i, m;
int a[N]  ={2,4,6,8,11,2,3}
m = a[0]
i=0

while(i < N)
   if  (m >= a[i]-1)     m = a[i]
   i++

assert  ∀j ∈ [0..N-1] . (m <= a[j])
```

- Intended to compute minimum
- Property being checked accordingly
- Erroneously min is checked against a[i]-1 instead of a[i]
- It computes the last value in longest subsequence a[i1], a[i2],…
  - Such that a[i1]=min and a[ik+1]<= a[ik]+1
- Here min=2 and such a sequence is [2,2,2] with m being computed as 2
- Therefore property holds
- However changing last element to 3 will make the sequence as [2,2,3] with m being 3 and property will fail

…Example

```
int i, m
int a[N]={2,4,6,8,11,2,2}

m = a[0]   i=0

while(i < N)
        if  (m >= a[i]-1)     m = a[i]
        i++

assert  ∀j ∈ [0..N-1] . (m <= a[j])
```

Some observations

- min+1 should not occur after last min
- Conservative check: no two elements differ by 1
- compute m from any two elements and check it against those two elements
- Rather than array elements we would think in terms of loop iterations
- So m from iterations 2 and 3 (i.e. from elements a[1] and a[2]) would be 4 and $4 \leq a[1]$ and $4 \leq a[2]$
- However, after changing last element to 3, the iterations 1 and 7 will make m to be 3 and $3 \leq a[0]$

Observation summary

- Only two iterations are needed to establish the property

- Shrinkable loops and Shrink factor
  - Loops for which only k iterations are needed are called shrinkable loops
  - and k is the shrink factor for the loop

Abstraction

```
int i, m, a[N]={2,4,6,8,11,2,2}
unsigned li, it[2]
 m = a[0] i=0
 it[0]=nondet() it[1]=nondet()
 assume(1 <= it[0] && it[0]<it[1])
 for (li=0; li < 2 ; li++)
      i = it[li] - 1
      if (!(i < N)) break
      if(m >= a[i]-1) m = a[i]
      i++

 assume(li==2)
 assert ∀t ∈ it . (m <= a[t-1])
```

- New *for* loop equivalent to two unrolling of the original loop for two iterations it[0] and it[1]
  - Residual loop
- The new property is checked over array elements corresponding to chosen iterations
  - Residual property

A Couple of More Examples

```
// property is violated for K > 1
//     but not for K=1
int a[N]
for(i=0; i< N-1; i++)
     if (a[i] > a[i+1])
          swap(a[i], a[i+1])
for (i=0; i< N-1; i++)
     assert(a[i] <= a[i+1])
```

```
// property is violated for K=1
// but holds for any K > 1
int a[N], min = -1, max = -1, f = 0, i
for (i=0; i< N; i++)
  assume (a[i] > 0)
  if (a[0]!=a[i])
      f =1
  if (min==-1 || min > a[i])
       min = a[i]
  if (max==-1 || max < a[i])
       max = a[i]

assert (f==0 && max==min ||
                          f==1 && max > min)
```

Program Characteristics

- For technique to be amenable
  - The array processing loop and property checking loop have same number of iterations
    - A loop free property can be seen in a loop of any iterations
- For technique to work
  - The accelerable index expressions of an array in array processing loop are a super set of accelerable index expressions of same array in property checking loop

Some Notations

- Program state
  - A map σ : Var → Val
- Iteration sequence
  - A sequence of integers (> 0) in ascending order representing the iterations to be considered
  - Given a sequence T of size >k , $P_k(T)$ represents set of subsequences of T of size k
- Loop acceleration
  - Variables whose value in an iteration j can be expressed as f(j). e.g. i=j-1 in the example program

Residual Loop and Property

Program : while (C) { B } ψ

T = $[j_1, j_2, \ldots, j_n]$ is an iteration sequence

Residual loop for T

$\{S_{j_1}, S_{j_2}, \ldots, S_{j_n}$; loop_exit:$\}$

Where S_{j_r} is

$\{A_{j_r}$; if (C) { B } else goto loop_exit $\}$

A_{j_r} assignments of accelerable variables

residual property is conjunction/disjunction of clauses corresponding to each iteration

```
i=3-1
if (i < N)
  if (m >= a[i]-1) m = a[i]
  i++
else    goto loop_exit
i=5-1
if (i < N)
  if (m >= a[i]-1) m = a[i]
  i++
else goto loop_exit
assert ∀ t ∈ {3,5} . (m <= a[t-1])
loop_exit:
```

Over Approximation

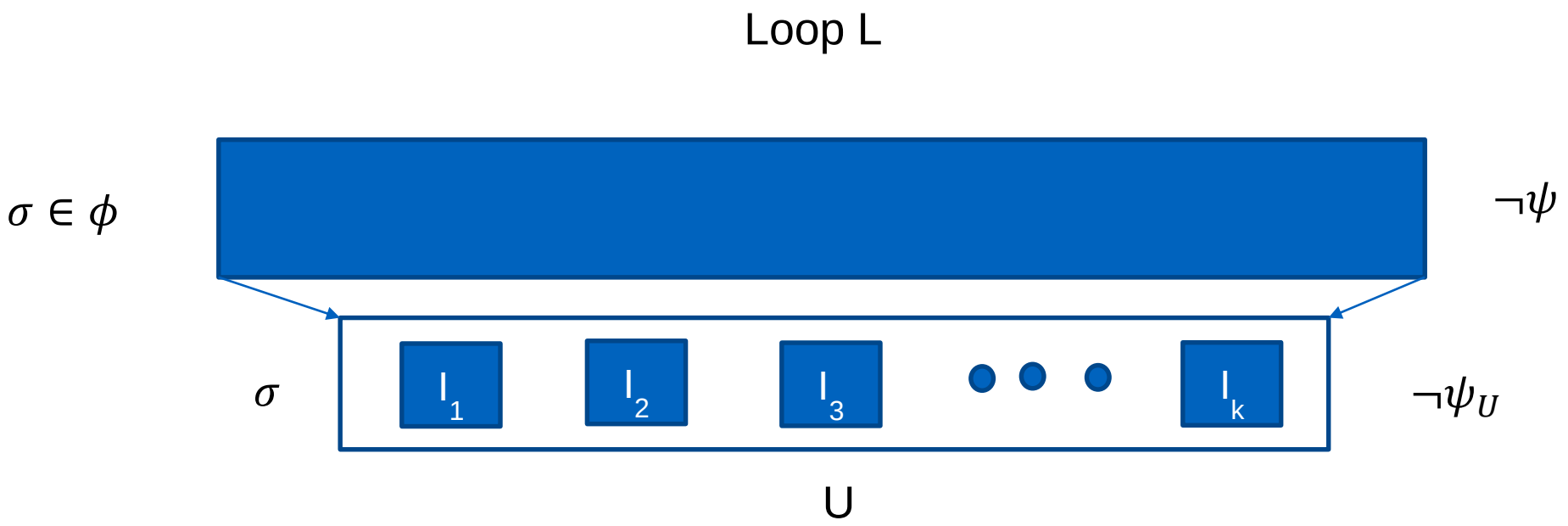
- Program (L , ψ) to be over-approximated (abstracted) by (L_U, ψ_U)
- Where U is a small sub-sequence of iterations chosen non deterministically
- Assume ϕ is set of states before L

Shrinkable loops

- Loop L is k-shrinkable wrt property ψ, iff starting form any state $\sigma \in \phi$, L satisfies ψ whenever all residuals of k-sized sequences satisfy their corresponding residual property

$$\forall\, \sigma \in \phi : ((\forall\, U \in P_k(T) : \{\sigma\}\, L_U\, \{\psi_U\}) \Longrightarrow \{\sigma\} L\, \{\psi\})$$

Loop k-Shrinkability



Verifying programs with shrinkable loop

- To verify a k shrinkable loop for ψ,
- Verify an abstraction (L_U, ψ_U) with a k-sized iteration sequence U chosen non-deterministically
- Abstraction has loops of size k CBMC can be used

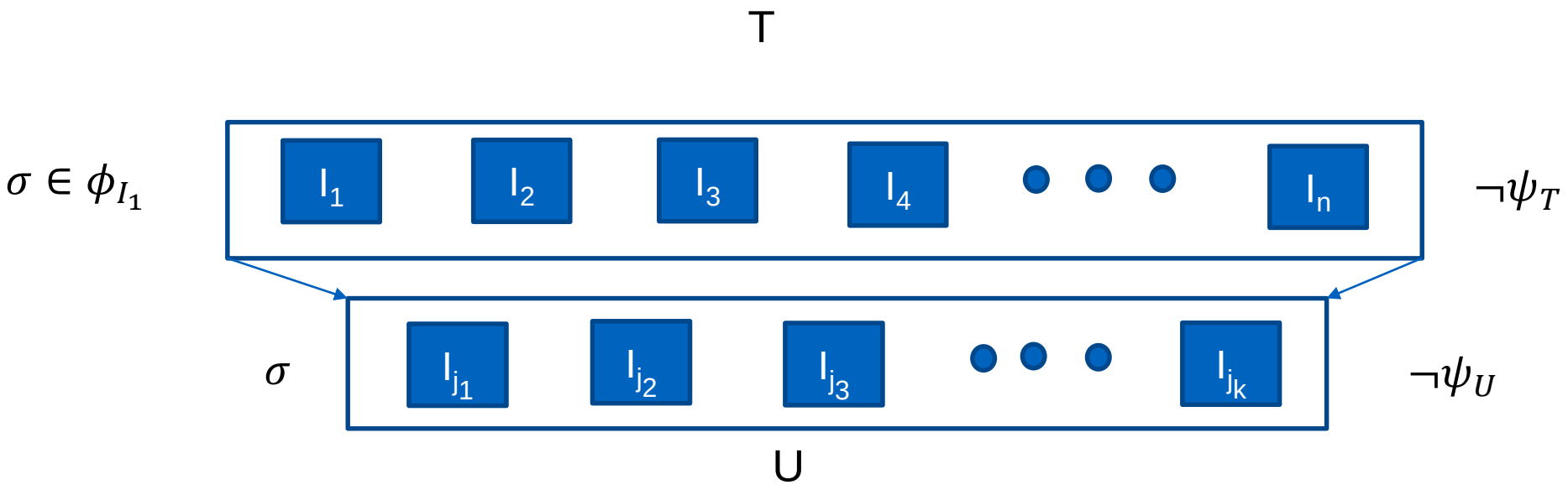
How to know if a loop is k-shrinkable

- Aim is to decide it by examining only smaller sequences
- Idea of sequence k-shrinkability
- So what is set of states at beginning of a residual loop ?
- We approximate it for iteration i as follows:

$$\phi_1 = \phi, \phi_i = sp(S_{i-1}, \phi_{i-1}) + \phi_{i-1}$$

  - The set of states at beginning of iteration i includes set of states at beginning of iteration i-1

Sequence k-Shrinkability



Sequence k-shrinkability

- Sequence T=j:T' is k-shrinkable wrt property ψ , iff starting from any state $\sigma \in \phi_j$, L_T satisfies ψ_T

whenever all residuals of k-sized sub-sequences of T satisfy their corresponding residual property

$$\forall \sigma \in \phi_j : ((\forall U \in P_k(T) : \{\sigma\} L_U \{\psi_U\}) \Longrightarrow \{\sigma\} L_T \{\psi_T\})$$

- Every iteration sequence of length 3 is 2-shrinkable for the given example

```
int i, m
int a[N]={2,4,6,8,11,2,2}
 m = a[0]   i=0
 while(i < N)
   if  (m >= a[i]-1)
       m = a[i]
   i++

 assert  ∀ j ∈ [0..N-1] .
                        (m <= a[j])
```

Shrinkable sequences to shrinkable loop

- k-shrinkability of complete sequence means loop is k-shrinkable
- Goal
  - k-shrinkablility of shorter sequences assures k-shrinkability of larger ones
- sequences of length k+1 are k-shrinkable → larger sequences are k-shrinkable ?
  - Unfortuntaley no
  - 3-size sequences are 2 shrinkable
  - 4-size sequences are not 3 shrinkable

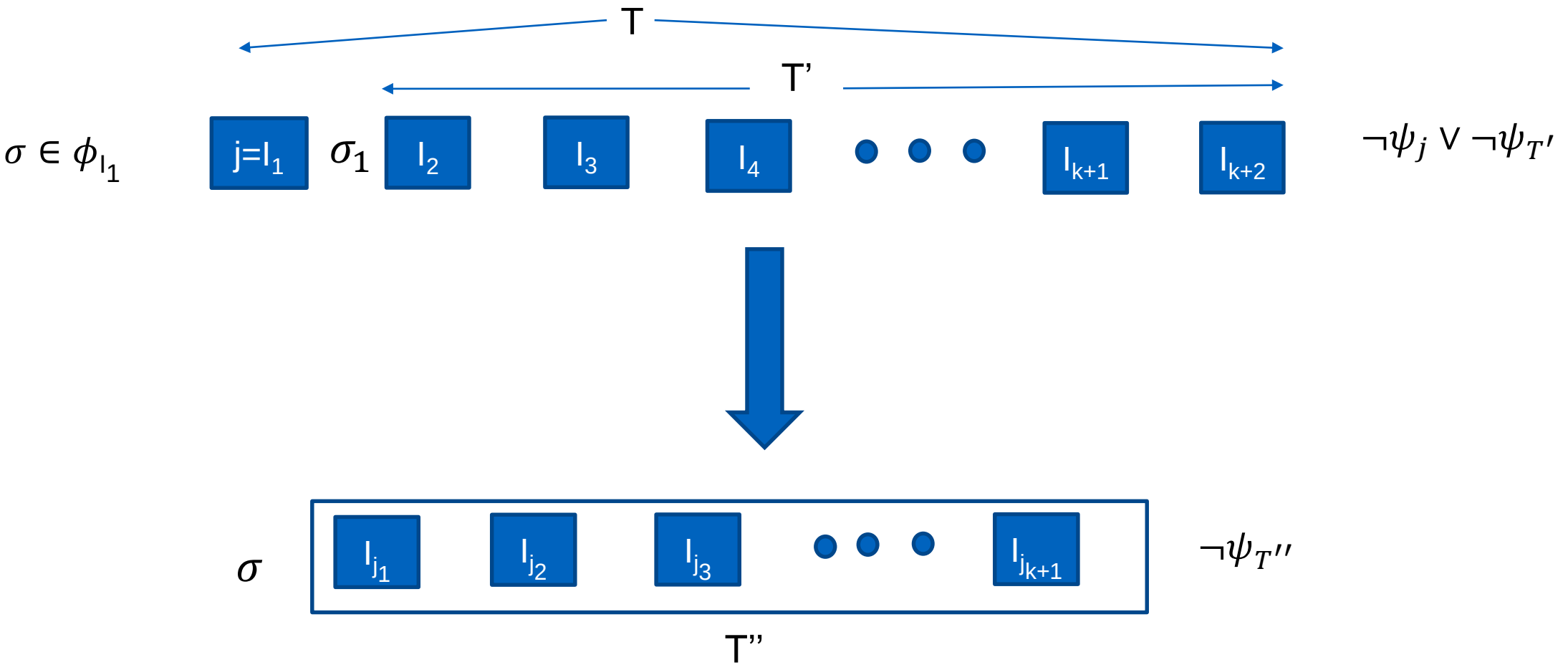
```
int i, m, a[S]
m = a[0] i=0
while(i < S)
  if(m >= a[i]-1) m = a[i]
  i++
```

assert ∀ j ∈ [0..S-1] .
(m <= a[j]+2)

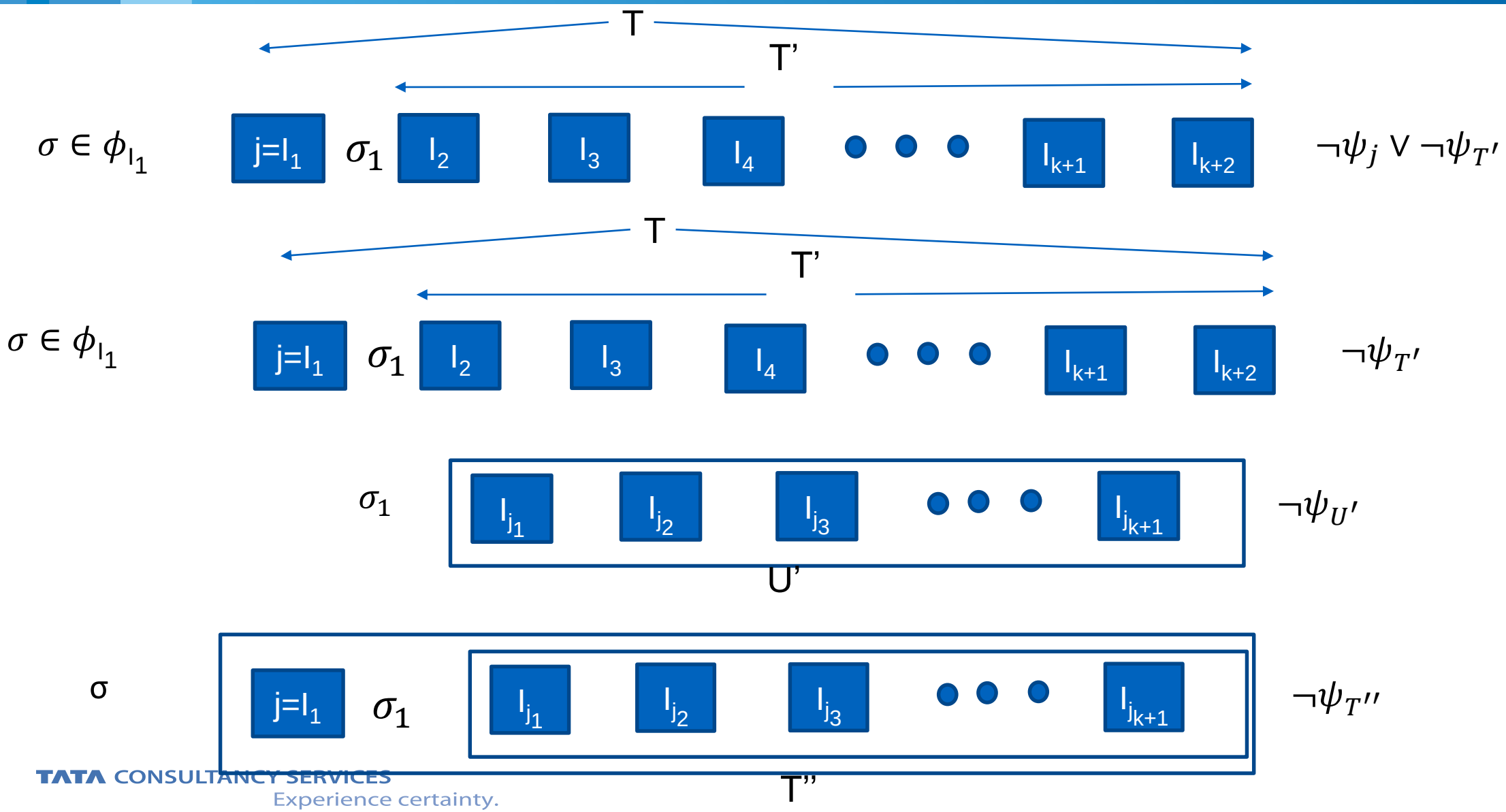
What is missing?

- Suppose all k+1 sized sequences are k-shrinkable
- Let T= j:T' be of size k+2
- For T to be k-shrinkable, if $\{\sigma\}\ L_T\ \{\neg\psi_T\}$ then there should be a U $\in P_k(T)$ such that $\{\sigma\}L_U\{\neg\psi_U\}$
- It is sufficient if we find T'' of size k+1 such that $\{\sigma\}\ \mathrm{L_{T''}}\{\neg\psi_{T''}\}$
- Assume we are dealing with conjunctive property

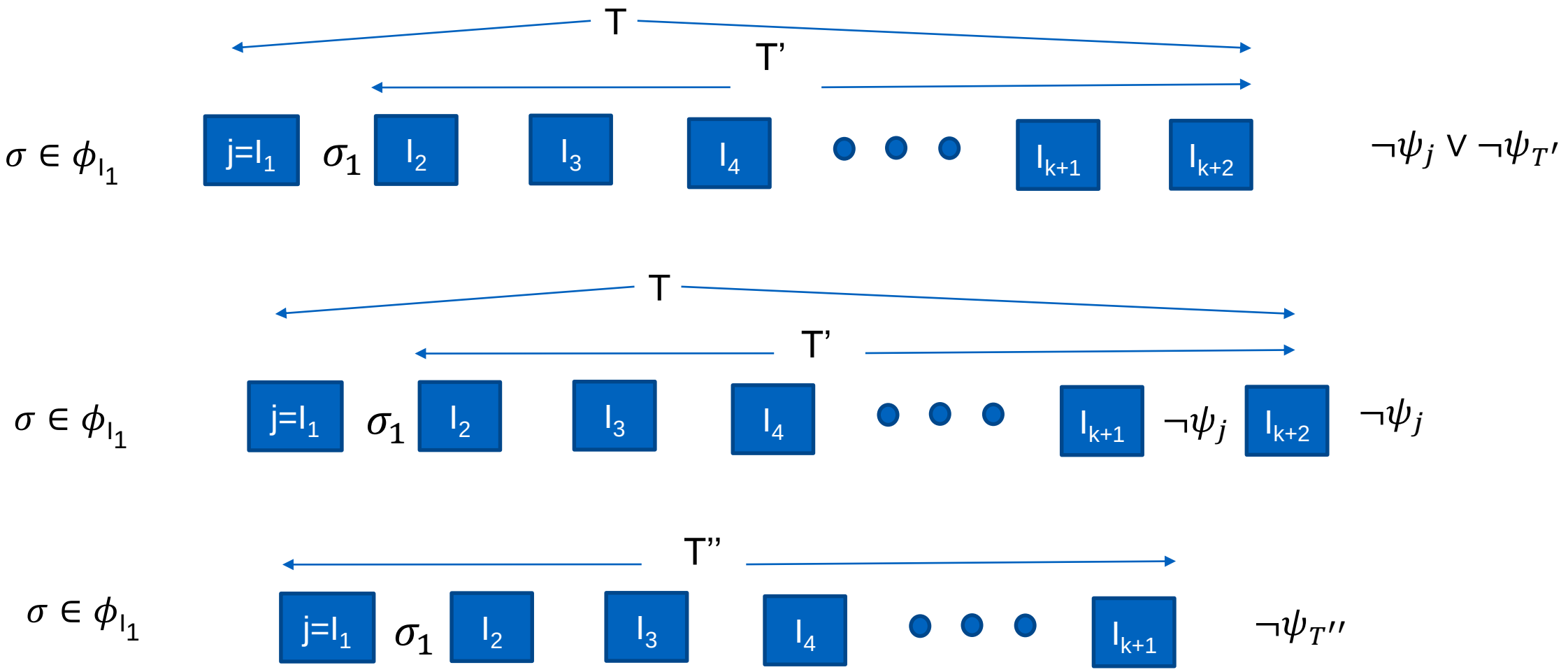
Sequence k-Shrinkability



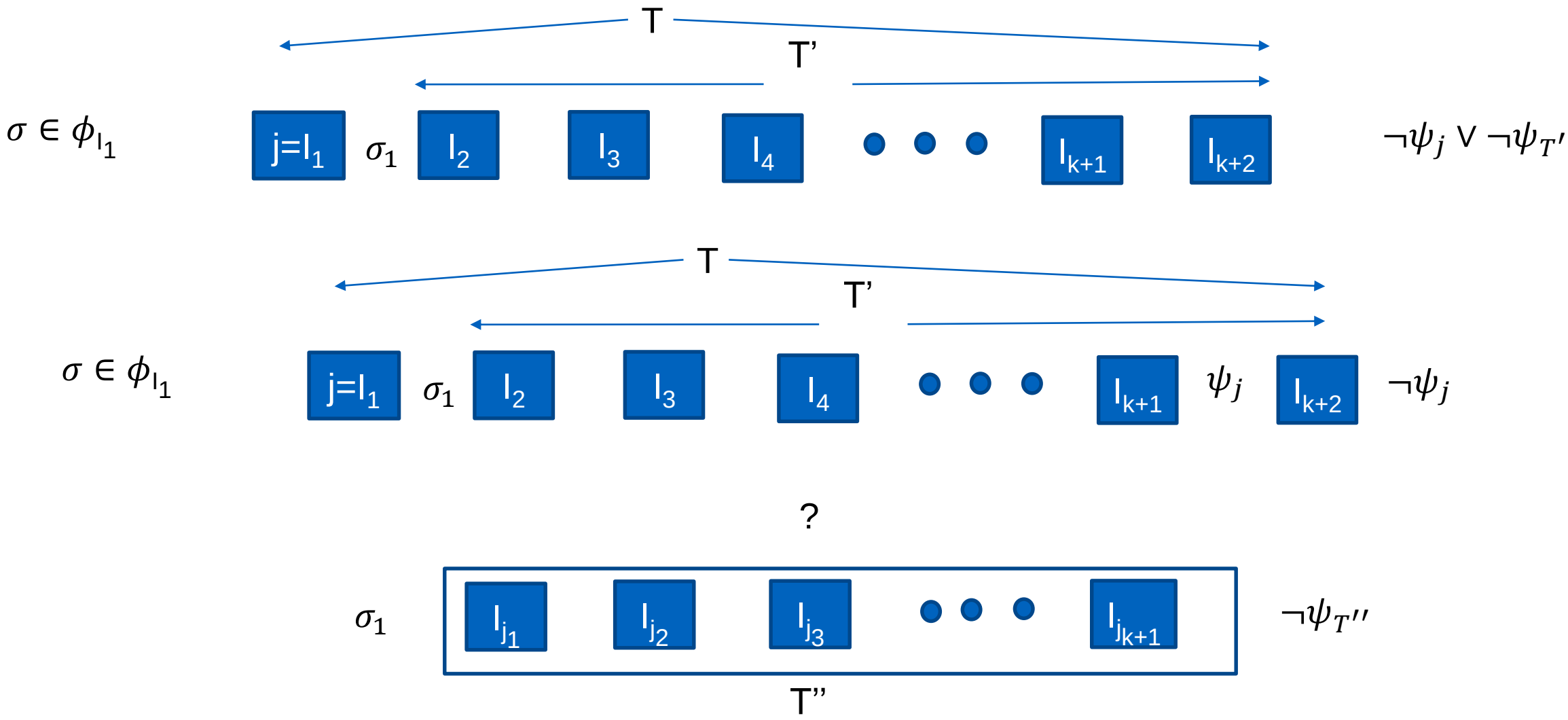
Case (1) : $\psi_{\{T'\}}$ is violated



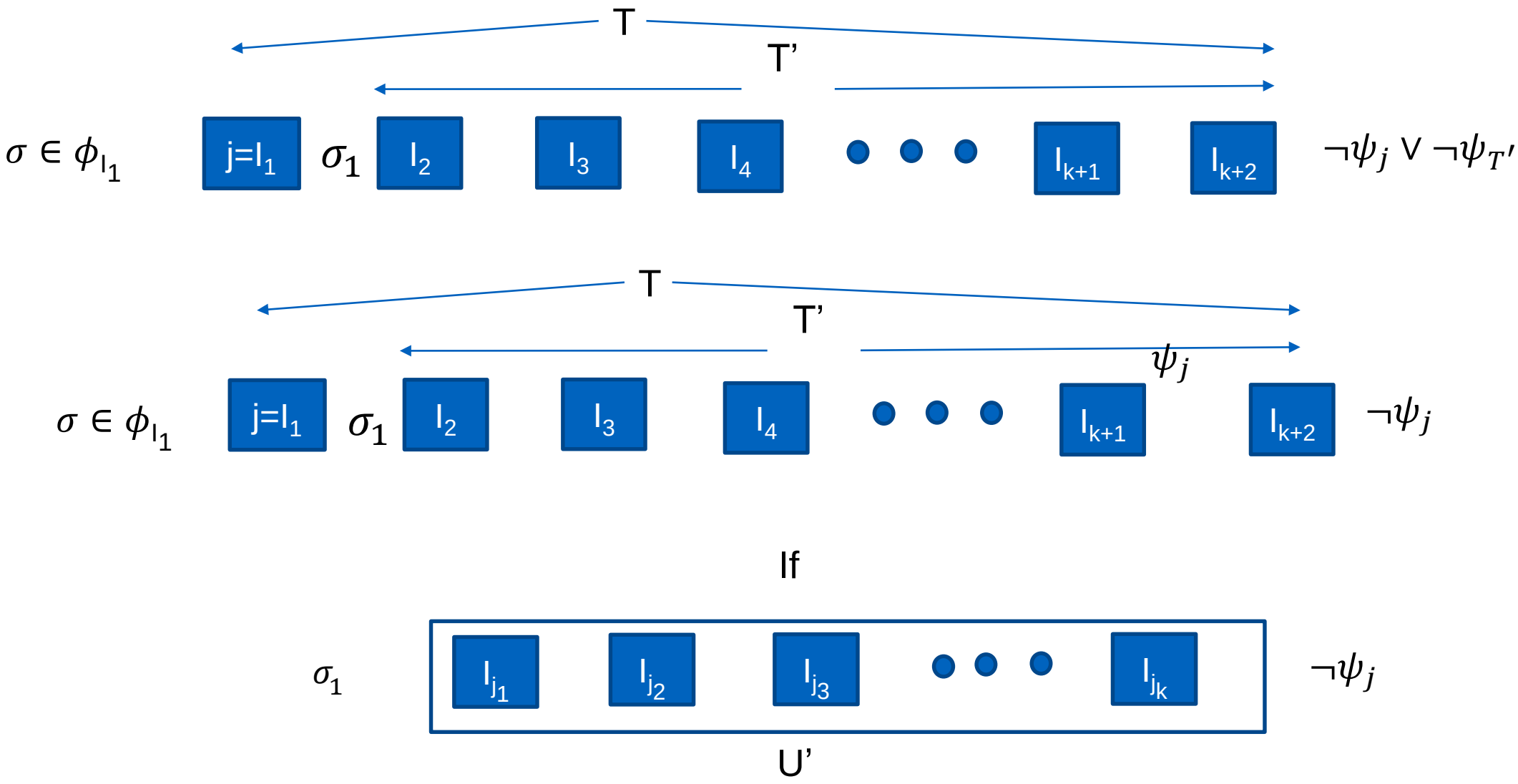
Case (2): ψ_j violates after last but one



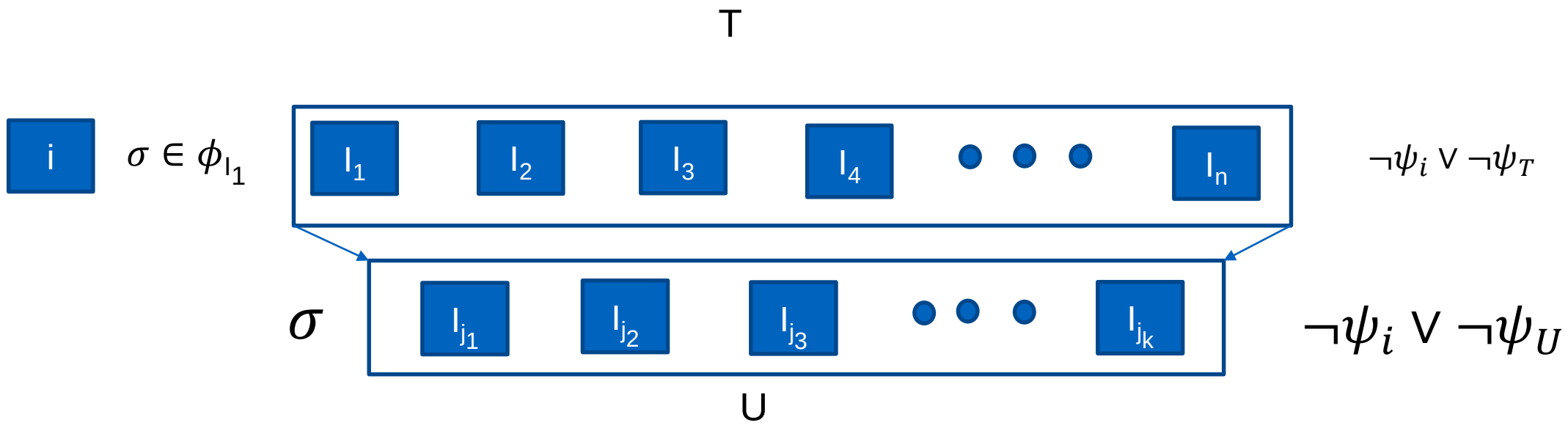
Case (3) : Only ψ_j violates and only at the end



Case (3) : Only ψ_j violates and only at the end



Refined Sequence k-Shrinkability



Sequence k-shrinkability (refined)

- T=j:T' is k-shrinkable wrt property ψ,
  iff
  given a past iteration i ($0 \leq i < j$)
  starting from any state $\sigma \in \phi_j$,
  L_T satisfies $\psi_i \wedge \psi_T$
  whenever all residuals of k-sized sub-sequences of T satisfy their corresponding residual property and ψ_i

$$\forall\, i, \forall\, \sigma \in \phi_j : \big((\forall\, U \in P_k(T) : \{\sigma\}\, L_U \{\psi_U \wedge \psi_i\}\big)$$

```
int i, m, a[S];
m = a[0]; i=0;
while(i < S){
  if(m >= a[i]-1)
       m = a[i];
  i++;
}
assert  ∀j ∈ [0..S-1] .
                    (m <= a[j]+2);
```

Important result

If all sequences of k+1 are k-shrinkable then the loop is k-shrinkable

Proof

- Show a stronger condition that all sequences longer than k+1 are k-shrinkable and hence loop will be k-shrinkable
- Let T=j:T' be a sequence of size n (>k+1) and assume all sequences of size up to n-1 are k shrinkable

Suppose $\exists i \,.\, 0 \leq i < j, \exists \sigma \in \phi_j \; : \{\sigma\} L_T \{\neg\psi_i \vee \neg\psi_T\}$

show that $\exists U \in P_k(T) : \{\sigma\}\; L_U \; \{\neg\psi_i \vee \neg\psi_U\}$

Let $\{\sigma\} L_j \{\sigma_1\}$

$\{\sigma_1\}\; L_{T'} \{\neg\psi_i \vee \neg\psi_j \vee \neg\psi_{T'}\}$

$\{\sigma_1\} L_{T'} \{\neg\psi_i \vee \neg\psi_{T'}\}$ OR $\{\sigma_1\} L_{T'} \{\neg\psi_j \vee \neg\psi_{T'}\}$

Here i and j both are past iterations for T'. So

$\exists U' \in P_k(T') : \{\sigma_1\} L_{U'} \{\neg\psi_i \vee \neg\psi_{U'}\}$ OR $\{\sigma_1\}\; L_{U'} \{\neg\psi_j \vee \neg\psi_{U'}\}$

$\{\sigma\} L_j ; L_{U'} \{\neg\psi_i \vee \neg\psi_j \vee \neg\psi_{U'}\}$

Let T'' = j:U'. Obviously T'' is of size k+1 and therefore shrinkable

$\exists U \in P_k(T'') : \{\sigma\} L_U \{\neg\psi_i \vee \neg\psi_U\}$

But $T''' \subset T$ and therefore $U \in P_k(T)$

Finding k-shrinkability

```
check_loop(k)
{
  choose an arbitrary
    iteration-sequence
    T of size k+1
  check_iter_seq(T);
}
```

Check property for check_loop for k=1,2,.. till property is satisfied

```
check_iter_seq(T)
{
  j = head(T); i = nondet();
  assume(0 <= i < j);
  X_initial= nondet(); c = true;
  for each k sized  U  ⊂  T {
    X = X_initial;   L_U;   c =  ψ_i ∧ ψ_U ;
    if (!c) break;
  }
  X = X_initial ; L_T; r = ψ_i ∧ ψ_T  ;
  assert(c → r);
}
```

$$\forall\, i, \forall\, \sigma \in \phi_j : \big((\forall\, U \in P_k(T) : \{\sigma\}\, L_U\{\psi_U \wedge \psi_i\}) \implies \{\sigma\}\, L_T\{\psi_T \wedge \psi_i\}\big)$$

Example of residual loop

```
int a[p];
int i=0,t=0;
while(i < n)
{
  a[t] = i;
  i+=2; t++;
}
```

```
int a[p], T[k], l, i=0, t=0;
init(T) ;  // ∀l ∈ 1..k − 1 : 1 ≤ T[l − 1] < T[l]
for (l=0;l<k;l++) {
  j = T[l]; i = (j-1)*2;
  t = j-1;
  if (!(i < n)) break;
  assume(0<=t<p),
  a[t] = i;
  i+=2; t++;
}
assume(l==k);
```

Handling Multiple loops

- It can handle computation consisting of a cascaded series of multiple loops
- If such cascaded loops can be coalesced into one equivalent loop

```
for (i=0; i< N; i++)
    sum1 = sum1+a[i];
for(i=0; i< N; i++)
    sum2 = sum2+a[i];
assert (sum1==sum2);
```

```
for (i=0; i< N; i++)
{
    sum1 = sum1 + a[i];
    sum2 = sum2 + a[i];
}
assert(sum1==sum2)
```

Results on SVCOMP 2018

Programs	True	False	Total
Total	123	44	167
Correct result by VeriAbs	109	33	142
Could not decide	14	11	25

VeriAbs came FIRST in Arrays sub-category