As previously stated, we would like to separate the two main activities (and challenges) associated with **deductive verification**:

- The **invention** of auxiliary constructs.

- Establishing the **validity** of the premises of the relevant rule.

In this lecture, we will present one of the simplest methods for deciding validity of the premises – the **small model** theory.
Parameterized Systems

In its simplest form, a parameterized system is a parallel composition, such as

$$ S(N) = P[1] \parallel \cdots \parallel P[N] \quad \text{or} \quad S(N) = C \parallel P[1] \parallel \cdots \parallel P[N] $$

where $N > 1$ is a parameter and $P[1], \ldots, P[N]$ are finite-state processes.

**Uniform Verification:** Establish in one verification effort that all instances of the system satisfy a property $\psi$. That is, $S(N) \models \psi$, for every $N > 1$. 
Non-Mathematical Induction

Of course, we can check separately $S(2) \models \psi$, $S(3) \models \psi$, $S(4) \models \psi$, etc. Suppose they all come out valid. What can we conclude?

To the extent that formal verification is viewed as a debugging tool, then this would usually uncover most of the existing bugs. However, if we wish to establish the absolute correctness of the system, this is insufficient.

Much research has been expended in trying to identify a cutoff value $N_0$ such that

$$\forall N : S(N) \models \psi \quad \text{iff} \quad S(N_0) \models \psi.$$ 

So far, only partial results have been obtained by Emerson, Ip, and Namjoshi.
Example: MUTEX

\[
\text{MUTEX}[N] ::= \begin{cases}
\text{in } N : \text{natural where } N > 1 \\
\text{local } x : \text{boolean where } x = 1 \\
\text{loop forever do}
\end{cases}
\]

\[
N \parallel \begin{cases}
P[h] ::= \\
\text{\(h=1\) when } x = 1 \text{ do } x := 0
\end{cases}
\]

\[
\begin{cases}
\ell_0 : \text{Non-Critical} \\
\ell_1 : \text{request } x \\
\ell_2 : \text{Critical} \\
\ell_3 : \text{release } x
\end{cases}
\]

The semaphore instructions “request \(x\)” and “release \(x\)” appearing in the program stand, respectively, for \(\langle \text{when } x = 1 \text{ do } x := 0 \rangle\) and \(x := 1\).

Suppose we wish to establish the invariance of the assertion \(p : \text{at}\_\ell_2[1] \rightarrow (x = 0)\). This assertion is not inductive by itself. We need to strengthen it into the inductive assertion:

\[
\varphi : \forall i \neq j : (\text{at}\_\ell_{2,3}[i] + \text{at}\_\ell_{2,3}[j] + x) \leq 1
\]

It only remains to check the validity of the premises of rule \(\text{INV}\).
Checking the Premises for MUX-SEM[5]

We can check the validity of the premises by using BDD techniques, e.g. using TLV. In file `mux5.pf` we write:

To calc-phi;
    Let phi := 1;
    For (i in 1...N)
        For (j in i+1...N)
            Let phi := phi &
            (((P[i].loc in 2..3) + (P[j].loc in 2..3) + x) <= 1);
        End -- For (j in i+1...N)
    End -- For (i in 1...N)
End -- To calc-phi;
Let $p := ((P[1].loc=2) \rightarrow (x=0))$;
calc-phi;

Let counter := Theta & !phi; -- Check Premise 1
  If (counter)
    Print ``\n Premise 1 invalid: ",counter,"
''; Quit;
End -- If (counter)

Let counter := phi & rho & !next(phi); -- Check Premise 2
  If (counter)
    Print ``\n Premise 2 invalid: ",counter,"
''; Quit;
End -- If (counter)

Let counter := phi & !p; -- Check Premise 3
  If (counter)
    Print ``\n Premise 3 invalid: ",counter,"
''; Quit;
End -- If (counter)
Consider the following signature:

- **Boolean variables** — $x_1, \ldots, x_a : \text{boolean}$.
- **Index variables** — $i_1, \ldots, i_b : [1..N]$. $1$ and $N$ are considered to be constants of type index.
- **Boolean arrays** — $y_1, \ldots, y_c : \text{array}[1..N]$ of boolean.

We define an index term to be an index variable $i_j$ or one of the constants $1, N$. An atomic formula is a boolean variable $x_k$, a boolean term $a[t]$, where $a$ is an array and $t$ is an index term, or a comparison $t_1 < t_2$, where $t_1$ and $t_2$ are index terms.

A boolean $R$-assertion is a boolean combination of atomic formulas. An $R$-assertion is built out of boolean $R$ assertions to which we apply further boolean operations and quantification over index variables. For example:

- $\forall i : \exists j \geq i : y[j] \neq y[i]$
- $\forall i : (1 \leq i \leq N) \rightarrow \exists j : i \leq j \land j \leq N \land y[j] \neq y[i]$

Note that the first formula can be viewed as an abbreviation of the second formula.

An $R$-assertion of the form $\forall i_1, \ldots, i_k : \exists j_1, \ldots, j_m : \varphi$, where $\varphi$ is a boolean $R$-assertions is called an $AER$-assertion.
$n$-Models

Let $\psi$ be an $R$-assertion. An $n$-model $M(n)$ over the vocabulary of $\psi$ provides the following interpretation to the elements appearing in $\psi$:

- Each boolean variable $x_j$ is assigned a value from $\{0, 1\}$.
- Each index variable $i_j$ is assigned a value from $[1..n]$. The constants 1 and $N$ are assigned the values 1 and $n$.
- Each array variable $a_j$ is assigned a boolean array of size $n$.

In general, it is not necessary to assign values to index variables which appear bounded in $\psi$. Thus, an adequate model over the vocabulary of the $AER$-assertion $\forall i : \exists j \geq i : y[j] \neq y[i]$ should only interpret the constant $N$ (as $n$) and the array $y$.

For example a 3-model for this formula can be specified by the assignment:

$$N = 3, \quad y = (0, 1, 1)$$

Given an $R$-assertion $\psi$ and a model $M(n)$, we can evaluate $\psi$ over $M(n)$, and find out whether $M(n)$ satisfies $\psi$. $R$-assertion $\psi$ is said to be valid if it is satisfied by every model.
A Small Model Theorem

Let \( \psi : \forall i_1, \ldots, i_k : \exists j_1, \ldots, j_m : \varphi \) be a closed AER-assertion.

**Claim 15. [Small Model Theorem]**

Assertion \( \psi \) is valid iff it is satisfied by all models \( M(n) \) for \( n \leq k + 2 \).

**Proof Sketch:**

In order to prove the claim, it is sufficient to show that if the negation

\( \chi = \neg \psi = \exists i_1, \ldots, i_k : \forall j_1, \ldots, j_m : \neg \varphi \)

has a satisfying model \( M(n) \) of size \( n > k + 2 \) then it also has a satisfying model of size \( \leq k + 2 \). We establish this by showing that every satisfying model of size \( n > k + 2 \) can be reduced to a satisfying model of size \( n - 1 \).

Let \( M \) be a satisfying model of size \( n > k + 2 \). Model \( M \) assigns to \( 1, i_1, \ldots, i_k, N \) values ranging in the set \( 1..n \). This set of values can contain at most \( k + 2 < n \) distinct values. Thus, there must exists some value \( r \in 1..n \) which is not the interpretation of any index term. Note that \( 1 < r < n \). We construct a new model \( M' \) of size \( n - 1 \) as follows:

- \( M'(N) = n - 1 \).
- For each index term \( t \), we let

\[
M'(t) = \text{if } M(t) < r \text{ then } M(t) \text{ else } M(t) - 1
\]
For each array \( a \), we let

\[
M'(a) = \lambda u : \text{if } u < r \text{ then } M(a)[u] \text{ else } M(a)[u - 1]
\]

It is not too difficult to show that \( M' \) is also a satisfying model.
Example

Consider the formula $\psi = \forall i : \exists j \geq i : y[j] \neq y[i]$. In this case, $k = 1$ so the cutoff value is $N_0 = 3$. The negation is given by $\chi = \exists i : \forall j \geq i : y[j] = y[i]$. Assertion $\chi$ has a satisfying model of size 5 given by

$$M(5) : \langle N : 5, \ i : 3, \ y : (0,0,1,1,1) \rangle$$

Noting that the range of interpretation consists of $\{1, 3, 5\}$, we can remove from the model the index values 2 and 4. This yields the following smaller satisfying model:

$$M(3) : \langle N : 3, \ i : 2, \ y : (0,1,1) \rangle$$
The Systems we Consider: BDS

A bounded-data discrete system (BDS) is an FDS whose system variables have the form:

- \( N : \text{integer where } N > 0 \) — aka the system’s parameter.
- \( x_1, \ldots, x_a : \text{boolean} \) — Any finite-domain type can be encoded by booleans.
- \( i_1, \ldots, i_b : [1..N] \) — The index variables.
- \( y_1, \ldots, y_c : \text{array } [1..N] \text{ of boolean} \)

The initial condition is a boolean combination of formulas of the form \( \forall \vec{i} : \Theta_k \), where \( \Theta_k \) are boolean \( R \)-assertions.

The transition relation is required to have the form:

\[
\exists \vec{h} \forall \vec{t} : R(\vec{h}, \vec{t}),
\]

where \( \vec{h}, \vec{t} \) are index variables, and \( R(\vec{h}, \vec{t}) \) is a boolean \( R \)-assertion. We denote by \( H = |\vec{h}| \) the number of existentially quantified variables in \( \rho \).

Typically, a bounded-data parameterized system is a parallel composition \( P[1] \parallel \cdots \parallel P[N] \).
BDS Example: MUTEX

\[
\text{MUTEX}[N] ::=
\begin{cases}
\text{in } N : \text{ natural where } N > 1 \\
\text{local } x : \text{ boolean where } x = 1 \\
\text{loop forever do}
\end{cases}
\]

\[
\begin{cases}
P[h] ::=
\end{cases}
\]

\[
\begin{align*}
\ell_0 & : \text{ Non-Critical} \\
\ell_1 & : \text{ request } x \\
\ell_2 & : \text{ Critical} \\
\ell_3 & : \text{ release } x
\end{align*}
\]
MUTEX as a BDS

\[ V : \begin{cases} N : \text{natural where } N > 1 \\ x : \text{boolean where } x = 1 \\ \pi : \text{array } [1..N] \text{ of } 0..3 \end{cases} \]

\[ \Theta : x \land \forall h : \pi[h] = 0 \]

\[ \rho : \exists h : \begin{cases} \pi'[h] = \pi[h] \land x' = x \\ \lor \pi[h] = 0 \land \pi'[h] = 1 \land x' = x \\ \lor \pi[h] = 1 \land x \land \pi'[h] = 2 \land x' = 0 \land \\ \lor \pi[h] = 3 \land \pi'[h] = 3 \land x' = x \\ \lor \pi[h] = 3 \land \pi'[h] = 0 \land x' = 1 \end{cases} \]

\[ \forall t \neq h : \pi'[t] = \pi[t] \]

BDS's usually contain the system array variable \( \pi[1..N] \) which represents the program counter in each of the processes.
Deciding the Verification Conditions

Consider the case that property \( p \) and the auxiliary assertion \( \varphi \) both have the form

\[
\varphi = \forall i, j : \psi(i, j)
\]

where \( \psi(i, j) \) is a boolean \( R \)-assertion.

The most complex verification condition, \( I_2 \), is then:

\[
(\forall u, v : \psi(u, v)) \land (\exists h \forall t : R(h, t)) \rightarrow \forall i, j : \psi'(i, j)
\]

Moving quantifiers to the front, this can be rewritten as

\[
\forall i, j, h : \exists u, v, t : (\psi(u, v) \land R(h, t)) \rightarrow \psi'(i, j)
\]

Thus, this premise has 3 universally quantified variables. According to Claim 15, it is sufficient to check the validity of all premises on models of size \( n \leq 5 \).

For BDS’s, this solves the 2nd task associated with deductive verification — Establishing the validity of the premises.
Deductive Verification of MUTEX

To verify mutual exclusion of program MUTEX, we take

\[ p : \forall i \neq j : \neg (at_{-l2}[i] \land at_{-l2}[j]) \]
\[ \varphi : \forall i \neq j : (at_{-l2,3}[i] + at_{-l2,3}[j] + x \leq 1) \]

Therefore, we write in file `mutex.pf`

To calc-phi;

Let phi := 1; Let exc := 1;
For (i in 1...N)
    For (j in i+1...N)
        Let phi := phi &
            (((P[i].loc in 3..4) + (P[j].loc in 3..4) + y) <= 1);
        Let exc := exc & !(P[i].loc = 3 & P[j].loc = 3);
    End -- For (j in i+1...N)
End -- For (i in 1...N)
End -- To calc-phi;
calc-phi;
Print "\nCheck mutual exclusion\n";
Call inv(exc,phi,1);
Example: Program Arbiter

Consider the following program \texttt{ARBITER}:

\[
\begin{align*}
\text{r, g} & : \text{array}[1..N] \text{ of boolean where } \forall i : [1..N] : r[i] = g[i] = 0 \\
A & : [1..N] \text{ where } k = 1 \\
\text{loop forever do} \\
\text{m0} : \text{ if } r[k] \text{ then} \\
\quad m1 : g[k] := 1 \\
\quad m2 : \text{ await } \neg r[k] \\
\quad m3 : g[k] := 0 \\
\quad m4 : k := k \oplus N 1
\end{align*}
\]

for which we wish to prove

\[
p : \forall i \neq j : \neg (\text{at}_- \ell_3[i] \land \text{at}_- \ell_3[j])
\]
Auxiliary Inductive Assertions

$r, g : \text{array}[1..N] \text{ of boolean where } \forall i : [1..N] : r[i] = g[i] = 0$

\[
A ::
\begin{align*}
  k : [1..N] \text{ where } k = 1 \\
  \text{loop forever do} \\
  m_0 : \text{ if } r[k] \text{ then} \\
  \quad m_1 : g[k] := 1 \\
  \quad m_2 : \text{ await } \neg r[k] \\
  \quad m_3 : g[k] := 0 \\
  m_4 : k := k \oplus_N 1
\end{align*}
\]

\[
\begin{align*}
  \text{loop forever do} \\
  \ell_0 : \text{ Non-Critical} \\
  \ell_1 : r[i] := 1 \\
  \ell_2 : \text{ await } g[i] \\
  \ell_3 : \text{ Critical} \\
  \ell_4 : r[i] := 0 \\
  \ell_5 : \text{ await } \neg g[i]
\end{align*}
\]

An appropriate auxiliary inductive assertion is given by:

\[
\forall i : \\
  (g[i] \leftrightarrow (k = i \land \text{at}_{m_2,3})) \land (r[i] \leftrightarrow \text{at}_{\ell_2..4}[i]) \\
  \land (k \neq i) \land \text{at}_{\ell_0,1,2,5}[i] \lor \\
  (k = i) \land \left( \begin{array}{l}
  \text{at}_{m_0,4} \land \text{at}_{\ell_0,1,2,5}[i] \lor \text{at}_{m_1,2} \lor \text{at}_{\ell_2}[i] \lor \\
  \text{at}_{m_2} \land \text{at}_{\ell_3..5}[i] \lor \text{at}_{m_3} \land \text{at}_{\ell_5}[i]
\end{array} \right)
\]
Progressing to Richer Signatures

Up to now, we only considered systems with array signatures $[1..N] \mapsto \text{bool}$. The method can be extended to signatures $\langle [1..N_1] \mapsto \text{bool}, [1..N_1] \mapsto [1..N_2] \rangle$.

Declarations for such systems will have the form:

\[
\begin{align*}
\text{type}_1 &= [1..N_1] \\
\text{type}_2 &= [1..N_2] \\
x_1, \ldots, x_a &: \text{boolean} \\
y_1, \ldots, y_b &: [1..N_1] \\
z_1, \ldots, z_c &: \text{array}[1..N_1] \text{ of boolean} \\
u_1, \ldots, u_d &: [1..N_2] \\
w_1, \ldots, w_e &: \text{array}[1..N_1] \text{ of } [1..N_2]
\end{align*}
\]

In the assertions, we allow comparisons ($<, =$) between $\text{type}_1$ values and between $\text{type}_2$ values, but never between a $\text{type}_1$ and a $\text{type}_2$ values.

We consider invariants of the form $\forall \vec{i}_1, \vec{i}_2 : \psi(\vec{i}_1, \vec{i}_2)$ and transition relations of the form $\exists \vec{h}_1, \vec{h}_2 \forall \vec{t}_1, \vec{t}_2 : R(\vec{h}_1, \vec{h}_2, \vec{t}_1, \vec{t}_2)$. 

Analysis of Reactive Systems, NYU, Spring, 2006
Example: A Finitary Version of the BAKERY Algorithm

\[
\text{in } N_1, N_2 : \text{ integer where } N_1 > 1, N_2 > 1 \\
\text{local } y : \text{ array } [1..N_1] \text{ of } [0..N_2] \text{ where } y = 0 \\
\text{loop forever do}
\]

\[
\begin{align*}
\ell_0 & : \text{ NonCritical} \\
\ell_1 & : \bigvee_{u=1}^{N_2} \text{ when } \forall j : (u > y[j]) \text{ do } y[i] := u \\
\ell_2 & : \text{ await } \forall j : (y[j] = 0 \lor y[i] < y[j]) \\
\ell_3 & : \text{ Critical} \\
\ell_4 & : y[i] := 0
\end{align*}
\]

\[
\text{end loop}
\]

\[
\text{loop forever do}
\]

\[
\begin{align*}
m_0 & : \bigvee_{u=1}^{N_2} \text{ when } u < y[i] \land \forall j : \left( \left( y[j] < u \lor y[j] \geq y[i] \right) \right) \text{ do } y[i] := u
\end{align*}
\]

\[
\text{end loop}
\]
The SMV Representation of BAKERY

In file bakery.smv we place:

MODULE main
DEFINE
    N_1 := 4;
    N_2 := N_1+1;
VAR
    u : 1..N_2;
    for (i=1; i<=N_1; i=i+1) {
        y[i]: 0..N_2;
        C[i]: process CM(i,y,u,N_1);
        R[i]: process CR(i,y,u,N_1);
    }
    Idle : process Id;
MODULE Id
MODULE CM(i,y,u,N)
DEFINE
    cond1 := & for (j=1; j<=N; j=j+1) {y[j]<u};
    cond2 := & for (j=1; j<=N; j=j+1) {j=i | y[j]=0 | y[i]<y[j]};
VAR
loc : 0..4;

ASSIGN
init(loc) := 0;
init(y[i]) := 0;
next(loc) := case
    loc=0             : \{0,1\};
    loc=1 & cond1     : 2;
    loc=2 & cond2     : 3;
    loc in \{3,4\}    : (loc+1) mod 5;
    1                 : loc;
esac;
for (j=1; j<=N; j=j+1) {
    next(y[j]) := case
        j != i           : y[j];
        loc=1 & cond1    : u;
        loc=4            : 0;
        1                : y[i];
esac;
}

JUSTICE
loc != 1, !(loc=2 & cond2), loc != 3, loc != 4
MODULE CR(i, y, u, N)
DEFINE
  cond := (u<y[i]) & (& for (j=1; j<=N; j=j+1) {
            y[j]<u | y[i] <= y[j]});

ASSIGN
  for (j=1; j<=N; j=j+1) {
    next(y[j]) := case
      j=i & cond : u;
      1                 : y[j];
    esac;
  }
Adjusting the Bounds for the Signature
\[ \langle \text{type}_1 \mapsto \text{bool}, \text{type}_1 \mapsto \text{type}_2 \rangle \]

Claim 16. The premises of rule \texttt{INV} are valid over \( S(N_1, N_2) \) for all \((N_1, N_2) \geq (2, 2)\) iff they are valid over \( S(N_1, N_2) \) for all \((N_1, N_2) \leq (N_1^0, N_2^0)\) where \( N_1^0 = b + I_1 + H_1 \) and \( N_2^0 = d + I_2 + H_2 + e(b + I_1 + H_1) \).

Additional extensions are introduced by allowing

- \( h_2 = h_1 + 1 \)  
  
  Can be rewritten as

\[ \forall h : (h \leq h_1) \lor (h \geq h_2) \]

- \( h_0 = 0 \)
  
  Can be rewritten as

\[ \forall h : h \geq h_0 \]

Similarly, for \( h = N \).
Deductive Verification of BAKERY Using Small Model Theorem

Applying bottom-up techniques, we propose the following assertion:

$$\forall i : (at_{\ell,0,1}[i] \leftrightarrow y = 0)$$

Using the affirmed invariant heuristic and propagation techniques we derive the following assertion:

$$\forall i \neq j : (at_{\ell,3,4}[i] \rightarrow (y[j] = 0 \lor y[i] < y[j]))$$
Consequently, we place in file `bakery_ded.pf`:

To `calc_phi`;

Let `phi := 1;` Let `exc := 1;`

For (i in 1...N_1)
  Let `phi := phi & (C[i].loc in 0..1 <-> y[i]=0);`
  For (j in 1...N_1)
    If (j != i)
      Let `phi := phi & (C[i].loc>2 -> (y[j]=0 | y[i]<y[j]));`
      Let `exc := exc & ((C[i].loc notin 3) | (C[j].loc notin 3));`
    End -- If (j != i)
  End -- For (j in 1...N_1)
End -- For (i in 1...N_1)
End -- To `calc_phi`;

calc_phi;

Print "\n Check mutual exclusion\n";

Call `inv(exc, phi, 1);"
Further Extensions

It is possible to extend the approach to any stratified signature, allowing \( \text{type}_1, \ldots, \text{type}_k \) and array types \( \text{type}_i \mapsto \text{type}_j \) as long as \( i < j \). This leads to:

**Claim 17.** Let \( S \) be a \( k \)-parameter BDS with \( k \geq 1 \) stratified types to which we wish to apply proof rule INV with the assertions \( \varphi \) and \( p \) having each the form
\[
\forall \vec{i}_1, \ldots, \vec{i}_k : \psi(\vec{i}_i).
\]
Then, the premises of rule INV are valid over \( S(N_1, \ldots, N_k) \) for all \( N_1, \ldots, N_k > 1 \) iff they are valid over \( S(N_1^0, \ldots, N_k^0) \) where \( N_1^0 = b_1 + H_1 + I_1 \), and for every \( i = 2, \ldots, k \),
\[
N_i^0 = (b_i + H_i + I_i) + \sum_{j=1}^{i-1} (e_{ji} \cdot N_j^0).
\]
Unstratified Systems

In some cases, we have to move to unstratified systems. This is the case of program Peterson($N$).

```plaintext
in N : integer where N > 1

type Pr_id : [1..N]
Level : [0..N]

local y : array Pr_id of Level where y = 0
s : array Level of Pr_id

loop forever do

  \[\begin{align*}
  &\ell_0 : \text{NonCritical} \\
  &\ell_1 : (y[i], s[1]) := (1, i) \\
  &\ell_2 : \text{while } y[i] < N \text{ do} \\
  &\quad \ell_3 : \text{await } s[y[i]] \neq i \lor \forall j \neq i : y[j] < y[i] \\
  &\quad \ell_4 : (y[i], s[y[i] + 1]) := (y[i] + 1, i) \\
  &\ell_5 : \text{Critical} \\
  &\ell_6 : y[i] := 0
  \end{align*}\]

\[\bigcap_{i=1}^{N} P[i] ::\]
```
## Summary of Results

<table>
<thead>
<tr>
<th>System</th>
<th>$N_0$</th>
<th>$\tau_1$</th>
<th>$\tau_2$</th>
<th>$\tau_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>MUX-SEM</td>
<td>5</td>
<td>.01</td>
<td>.01</td>
<td>.01</td>
</tr>
<tr>
<td>S. German’s Cache</td>
<td>4</td>
<td>10.21</td>
<td>10.72</td>
<td>133.04</td>
</tr>
<tr>
<td>Illinois’s Cache</td>
<td>4</td>
<td>1.47</td>
<td>.04</td>
<td>.58</td>
</tr>
<tr>
<td>3-stages Pipeline</td>
<td>6</td>
<td>20.66</td>
<td>.27</td>
<td>29.59</td>
</tr>
<tr>
<td>SZYMANSKI</td>
<td>4</td>
<td>&lt; .01</td>
<td>.06</td>
<td>.06</td>
</tr>
<tr>
<td>TOKEN-RING</td>
<td>5</td>
<td>&lt; .01</td>
<td>&lt; .01</td>
<td>&lt; .01</td>
</tr>
<tr>
<td>BAKERY</td>
<td>5</td>
<td>.41</td>
<td>.16</td>
<td>.25</td>
</tr>
<tr>
<td>PETERSON(N)</td>
<td>(6,7)</td>
<td>79</td>
<td>1211</td>
<td>240</td>
</tr>
</tbody>
</table>

Here

$\tau_1$ : Time to compute $reach$

$\tau_2$ : Time to compute candidate assertion

$\tau_3$ : Time to check premises over $S(N_0)$