CS513:	**Lect. 13: Undecidability of FO Logic and Gödel's Incompleteness Thm**

Recall that last time we defined recursive and r.e. and we proved that K and HALT are both r.e. and not recursive.

Definition 13.1 A **partial, recursive** function, $f : \mathbf{N} \to \mathbf{N} \cup \{\nearrow\}$ is a function computed by a Turing Machine. A **total, recursive** function is a partial, recursive function, f, that happens to be total, i.e, $f : \mathbf{N} \to \mathbf{N}$. □

Theorem 13.2 (Fundamental Theorem of r.e. Sets) *Let $S \subseteq \mathbf{N}$. T.F.A.E.*

1. *S is the domain of a partial, recursive function, i.e., for some $n \in \mathbf{N}$, $S = \{x \in \mathbf{N} \mid M_n(x)\downarrow\}$.*
2. *$S = \emptyset$ or S is the range of a total, recursive function, i.e., for some total, recursive $M_m(\cdot)$, $S = \emptyset$ or $S = M_m(\mathbf{N})$*
3. *S is the range of a partial, recursive function, i.e., for some $r \in \mathbf{N}$, $S = M_r(\mathbf{N})$*
4. *S is r.e., i.e., for some $t \in \mathbf{N}$, $S = W_t$*

13.1 Reductions = Translations

Definition 13.3 For $S, T \subseteq \{0,1\}^\star$, S is **reducible** to T ($S \leq T$) iff there exists a "very easy to compute" total function $f : \mathbf{N} \to \mathbf{N}$, s.t. $\forall w \in \mathbf{N}, \quad (w \in S) \quad \Leftrightarrow \quad (f(w) \in T)$.

Note: All our reductions, f, will be computable in LOGSPACE and in fact they will be in FO, i.e., first-order definable..

Note: f **translates** membership questions for S to membership questions for T. Thus, complexity$(S) \leq$ complexity(T). □

$$\forall w \in \mathbf{N}, \quad \chi_S(w) \quad = \quad \chi_T(f(w))$$
$$\forall w \in \mathbf{N}, \quad (w \in S \quad \Leftrightarrow \quad f(w) \in T)$$

Sometimes the "⇔" in the definition of reductions makes students think that reductions go both ways, but that is not true, they only go from S to T. The reason for the "⇔" is that one arrow tells us that if $f(w) \in T$ then $w \in S$, and the arrow in the other direction tells us that if $f(w) \notin T$ then $w \notin S$. Thus the answer to the question, "Is $f(w) \in T$?", is always the same as the answer to the question, "Is $w \in S$?".

Proposition 13.4 $K \leq$ HALT

Proof: We want to build an easy-to-compute program translator $f : \mathbf{N} \to \mathbf{N} \times \mathbf{N}$ such that,

Want: $$\forall z \in \mathbf{N} \quad (z \in K) \quad \Leftrightarrow \quad (f(z) \in \text{HALT})$$

Want: $$\forall z \in \mathbf{N} \quad (M_z(z) = 1) \quad \Leftrightarrow \quad (M_{L(f(z))}(R(f(z)))) \downarrow$$

Here for a pair, $z = (a, b)$, we use the component functions L, R, selecting the left part and the right part, i.e., $L(a, b) = a$ and $R(a, b) = b$.

Define $f(z) = (A(z), 0)$ where $A(z)$ is the following Turing Machine program, which on on input x,

1. Delete input, x.
2. run $M_z(z)$
3. **if** $(M_z(z) == 1)$: **return**(17)
4. run forever

Thus, if $M_z(z) = 1$, then $M_{A(z)}(0) \downarrow$; and otherwise, $M_{A(z)}(0) = \nearrow$.

Thus,

$$z \in K \quad \Leftrightarrow \quad M_z(z) = 1 \quad \Leftrightarrow \quad M_{A(z)}(0) \downarrow \quad \Leftrightarrow \quad f(z) \in \text{HALT}$$

[In the proof of the above series of equivalences, note that if $M_z(z) = \nearrow$, then $M_{A(z)}(0) = \nearrow$.] □

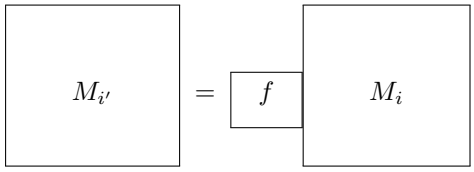
Theorem 13.5 (Fundamental Thm. of Reductions) *If $S \leq T$, then,*

1. *If* T *is* r.e., *then* S *is* r.e..
2. *If* T *is co-*r.e., *then* S *is co-*r.e..
3. *If* T *is* Recursive, *then* S *is* Recursive.

Proof: $S \leq T \ \wedge \ T \in$ r.e. $\Rightarrow$ $S \in$ r.e.

Let $f : S \leq T$, i.e., $\forall x(x \in S \ \Leftrightarrow \ f(x) \in T)$, $T = W_i$.

From M_i, compute the TM $M_{i'}$ which on input x does the following: (a). compute $f(x)$; (b) run $M_i(f(x))$



$$(x \in S) \quad \Leftrightarrow \quad (f(x) \in T) \quad \Leftrightarrow \quad (M_i(f(x)) = 1) \quad \Leftrightarrow \quad (M_{i'}(x) = 1)$$

Therefore, $S = W_{i'}$, and S is r.e. as desired.

In other words, $\text{P}_S = p_T \circ f$. We are given the Turing machines that compute the partial recursive function p_T and the total recursive function f. From these, we can easily construct the Turing machine, $M_{i'}$, which computes p_S.

Observation 13.6 $f : S \leq T \quad \Leftrightarrow \quad f : \overline{S} \leq \overline{T}$.

Thus, $T \in$ co-r.e. $\Rightarrow$ $\overline{T} \in$ r.e. $\Rightarrow$ $\overline{S} \in$ r.e. $\Rightarrow$ $S \in$ co-r.e.

$T \in$ Recursive $\Rightarrow$ $(T \in$ r.e. $\wedge \ T \in$ co-r.e.$)$ $\Rightarrow$

$(S \in$ r.e. $\wedge \ S \in$ co-r.e.$)$ $\Rightarrow$ $S \in$ Recursive □

Moral: Suppose $S \leq T$. Then,

- If T is easy, then so is S.
- If S is hard, then so is T.

Definition 13.7 Let $C \subseteq \mathbf{N}$. C is **r.e.-complete** iff

1. $C \in$ r.e., and
2. $\forall A \in$ r.e. $(A \leq C)$

□

Intuition: C is a "hardest" r.e. set.

Theorem 13.8 *K is r.e. complete.*

Proof: We already know that K is r.e.

Let A be an arbitrary r.e. set, i.e., $A = W_i$ for some i.

Wanted: total recursive f, s.t.: $\forall n(n \in A \Leftrightarrow f(n) \in K)$

Define total, recursive f which on input n computes:

$M_{f(n)} =$ | Erase input | Write n | M_i |

$M_{f(n)}$ ignores its input and instead runs $M_i(n)$.

$$n \in A \quad \Leftrightarrow \quad M_i(n) = 1 \quad \Leftrightarrow \quad \forall x(M_{f(n)}(x) = 1) \quad \Leftrightarrow \quad M_{f(n)}(f(n)) = 1 \quad \Leftrightarrow \quad f(n) \in K$$

□

Proposition 13.9 *Suppose C is r.e.-complete and:*

1. $S \in$ r.e., *and*
2. $C \leq S$

then S is r.e.-complete.

Proof: Show: $\forall A \in \text{r.e.}\,(A \leq S)$

Know: $\forall A \in \text{r.e.}\,(A \leq C)$

Follows by transitivity of $\leq$: $A \leq C \leq S$. □

Corollary 13.10 HALT *is r.e.-complete.*

13.2 Post's Correspondence Problem (PCP)

Definition 13.11 [Post's Correspondence Problem (PCP)] An instance of PCP, $p = ((x_1, y_1), \ldots, (x_r, y_r))$ is a finite sequence of pairs of binary strings. The instance p has a solution ($p \in$ PCP) iff there exists a finite sequence of indices: $i_1, \ldots, i_n$ from $\{1, \ldots, r\}$ such that $x_{i_1} x_{i_2} \cdots x_{i_n} = y_{i_1} y_{i_2} \cdots y_{i_n}$. The PCP Question is whether can we contruct a pair of equal strings by repeatedly appending pairs from p. □

Example 13.12 $p_0 = ((1, 101), (10, 00), (011, 11))$ is a positive instance of the Post Correspondence Problem, i.e., $p_0 \in$ PCP. A solution is 1323. Note that,

$$x_1 x_3 x_2 x_3 = 1\,011\,10\,011 = 101\,11\,00\,11 = y_1 y_3 y_2 y_3\,.$$ □

PCP is a very simplified version of the Halting Problem in which Turing machine computations are encoded as binary strings. PCP has the same complexity at the halting problem, HALT:

Fact: [Post] PCP is r.e. complete. In particular, PCP is r.e. but not recursive.

13.3 Undecidability of FO Logic

Recall that FO-VALID is the set of all first-order formulas that are true in all appropriate structures. In his Ph.D. thesis in 1930, Gödel proved,

Theorem 13.13 *[Gödel's Completeness Thm] There is a complete recursive axiom system for FO Logic. That is, in this system, for all $\varphi \in \mathcal{L}(\Sigma)$, φ is a theorem iff φ is valid. In symbols,*

$$\vdash \varphi \quad \Leftrightarrow \quad \varphi \in \text{FO-VALID} .$$

Corollary 13.14 (FO-VALID **is** r.e.**)** FO-VALID $\in$ r.e..

In Gödel's original proof, he showed that the axiomitization of FO Logic in Russel and Whitehead's *Principia Mathematica* is complete. In Lecture 10, we proved Thm. 13.13 by showing that Resolution is complete for FO logic.

A year later, in 1931, Gödel proved that there is no recursive and complete axiom system for all of mathermatics (or even for number theory). If there had been, then it would have meant that MATH and thus also FO-VALID and HALT would all be recursive.

Theorem 13.15 *[*Gödel's First Incompleteness Thm.*] There is no complete recursive axiom system for all of math, nor even for just* Theory($\mathbf{N}$) $= \{\varphi \in \mathcal{L}(\Sigma_{\#\text{thy}}) \mid \mathbf{N} \models \varphi\}$.

13.4 Unsolvability of FO Logic

We now prove that FO-VALID is not solvable:

Theorem 13.16 (FO Logic is not Recursive) FO-VALID *is* r.e. *complete.*

Proof: We follow the proof in the text which shows that PCP $\leq$ FO-VALID. In particular, we will show that there is a very easy-to-compute transformation, τ, which translates any PCP problem, p, to an FO formula, $\tau(p)$, such that

$$p \in \text{PCP} \quad \Leftrightarrow \quad \tau(p) \in \text{FO-VALID} \tag{13.16}$$

Let $\Sigma_{\text{PCP}} = (P^2; a, f_0^1, f_1^1)$. Let $p = ((x_1, y_1), \ldots, (x_r, y_r))$ be an arbitrary instance of PCP.

Notation 13.17 (Schöning)

$$f_{j_s j_{s-1} \cdots j_2 j_1}(x) \quad \hookrightarrow \quad f_{j_1}(f_{j_2}(\cdots(f_{j_{s-1}}(f_{j_s}(x)))\cdots)$$

For example, $f_{110}(x) = f_0(f_1(f_1(x)))$.

We now construct $\tau(p) = \alpha_p \wedge \beta_p \rightarrow \gamma_p$ so that Eqn 13.16 holds.

$$\begin{aligned}
\alpha_p &\stackrel{\text{def}}{=} \bigwedge_{i=1}^{r} P(f_{x_i}(a), f_{y_i}(a)) \\
\beta_p &\stackrel{\text{def}}{=} \varphi_2 = \forall uv\,(P(u,v) \rightarrow \bigwedge_{i=1}^{r} P(f_{x_i}(u), f_{y_i}(v))) \\
\gamma_p &\stackrel{\text{def}}{=} \exists z\, P(z,z)
\end{aligned}$$

The intuitive idea behind these formulas is that they mean, "If we start with some pair from p, (α_p), and we continue to add any pairs from p any number of times, (β_p), then we can eventually reach a situation where the two strings are equal, (γ_p)."

More explicitly, a is the starting point, e.g., the empty string, and we are **not** given that $P(a, a)$ holds.

The formula α_p says that we may start with any of the pairs (x_1, y_1), through (x_r, y_r). For the example of p_0 (Ex. 13.12),

$$\alpha_{p_0} \quad \stackrel{\text{def}}{=} \quad P(f_1(a), f_{101}(a)) \ \wedge \ P(f_{10}(a), f_{00}(a)) \ \wedge \ P(f_{011}(a), f_{11}(a))$$

The formula β_p says that for any position (u, v), that we have reached, we made add any with any of the pairs (x_1, y_1), through (x_r, y_r). Continuing the example of p_0,

$$\beta_{p_0} \quad \stackrel{\text{def}}{=} \quad \forall uv \, (P(u, v) \ \rightarrow \ (P(f_1(u), f_{101}(v)) \ \wedge \ P(f_{10}(u), f_{00}(v)) \ \wedge \ P(f_{011}(u), f_{11}(v))))$$

We now prove Eqn 13.16.

Assume $p \in$ PCP. Let $\mathcal{A} \in \text{STRUC}[\Sigma_{\text{PCP}}]$ be arbitrary. We will show that $\mathcal{A} \models \alpha_p \wedge \beta_p \rightarrow \gamma_p$.

If $\mathcal{A} \not\models \alpha_p \wedge \beta_p$, then $\mathcal{A} \models \tau(p)$, so we may assume that $\mathcal{A} \models \alpha_p \wedge \beta_p$.

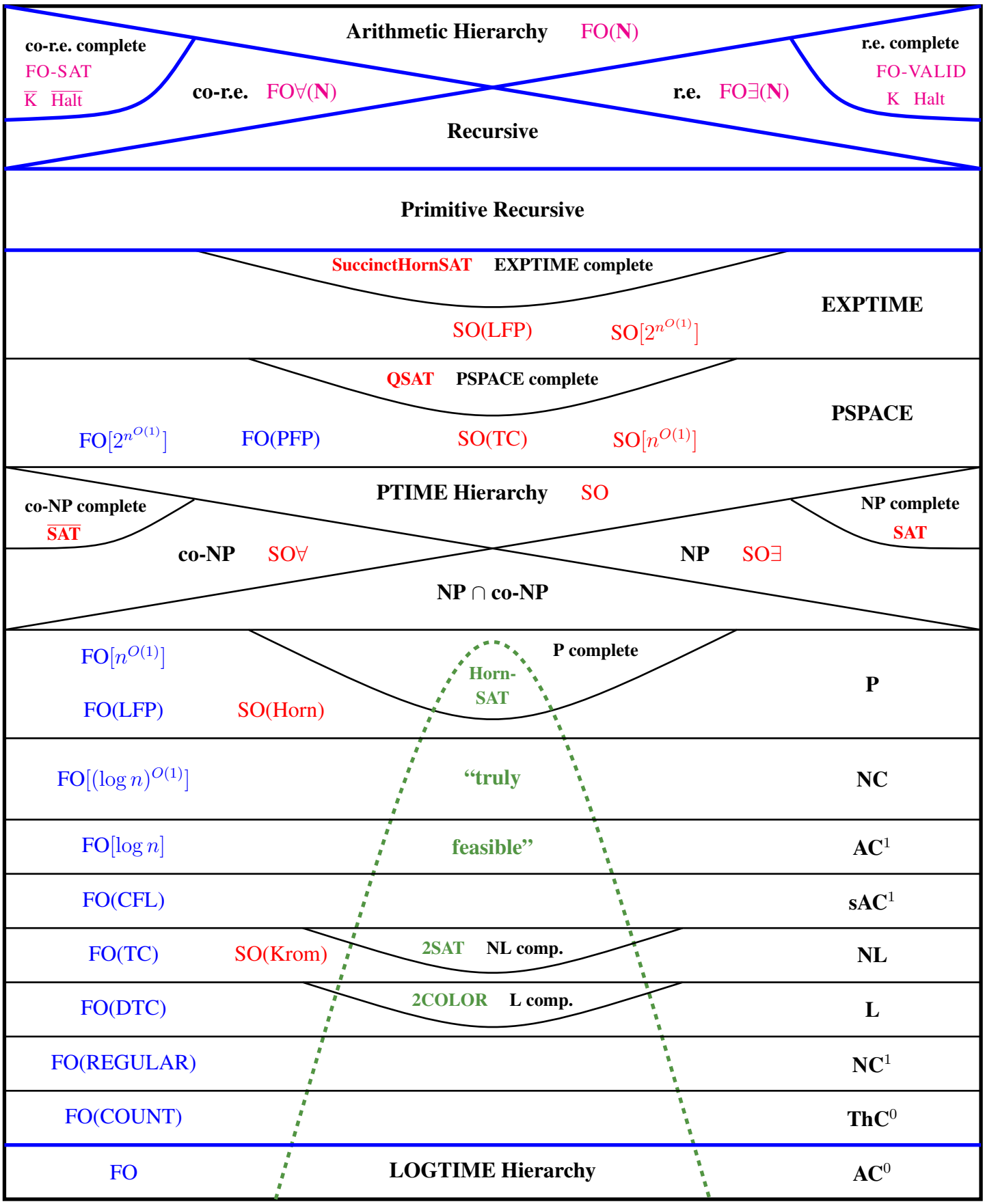
Consider the example of p_0 (Ex. 13.12) whose solution is 1323. Since $\mathcal{A} \models \alpha_{p_0}$, we have that $\mathcal{A} \models P(f_1(a), f_{101}(a))$. Since $\mathcal{A} \models \beta_{p_0}$, we can conclude that $\mathcal{A} \models P(f_{1011}(a), f_{10111}(a))$. Applying β_{p_0} again, we have that $\mathcal{A} \models P(f_{101110}(a), f_{1011100}(a))$. Applying β_{p_0} a third time we get, $\mathcal{A} \models P(f_{101110011}(a), f_{101110011}(a))$.

Now, we have won because $\mathcal{A} \models \gamma_{p_0}$. The witness is $z = f_{101110011}(a)$. Thus, since $p \in$ PCP, $\mathcal{A} \models \tau(p)$. Since $\mathcal{A}$ was arbitrary, it follows that $\tau(p) \in$ FO-VALID.

Conversely, assume $\tau(p) \in$ FO-VALID. We construct the standard model we have in mind, $\mathcal{S} \in \text{STRUC}[\Sigma_{\text{PCP}}]$:

$$\begin{array}{rcl}
|\mathcal{S}| & = & \{0,1\}^* \\
P^{\mathcal{S}} & = & \left\{(u, v) \in (\{0,1\}^*)^2 \ \middle| \ \exists i_1, \ldots, i_n \ (u = f_{x_{i_1} \cdots x_{i_n}}(a) \wedge v = f_{y_{i_1} \cdots y_{i_n}}(a))\right\} \\
a & = & \epsilon \\
f_0^{\mathcal{S}} & = & \left\{(w, w0) \ \middle| \ w \in \{0,1\}^\star\right\} \\
f_1^{\mathcal{S}} & = & \left\{(w, w1) \ \middle| \ w \in \{0,1\}^\star\right\}
\end{array}$$

Note that by construction, $\mathcal{S} \models \alpha_p \wedge \beta_p$. Thus, $\mathcal{S} \models \gamma_p$. But, again by the construction of $\mathcal{S}$, this means that the z asserted to exists in γ_p is a winning position for the PCP problem of p, i.e., $p \in$ PCP. $\square$


Arithmetic Hierarchy FO(N)
co-r.e. complete
FO-SAT
$\overline{\text{K}}$ $\overline{\text{Halt}}$
co-r.e. FO∀(N)
r.e. FO∃(N)
r.e. complete
FO-VALID
K Halt
Recursive
Primitive Recursive
SuccinctHornSAT EXPTIME complete
EXPTIME
SO(LFP)
SO[$2^{n^{O(1)}}$]
QSAT PSPACE complete
PSPACE
FO[$2^{n^{O(1)}}$]
FO(PFP)
SO(TC)
SO[$n^{O(1)}$]
PTIME Hierarchy SO
co-NP complete
$\overline{\text{SAT}}$
co-NP SO∀
NP SO∃
NP complete
SAT
NP ∩ co-NP
FO[$n^{O(1)}$]
FO(LFP)
SO(Horn)
P complete
Horn-SAT
P
FO[$(\log n)^{O(1)}$]
"truly
NC
FO[$\log n$]
feasible"
AC^1
FO(CFL)
sAC^1
FO(TC)
SO(Krom)
2SAT NL comp.
NL
FO(DTC)
2COLOR L comp.
L
FO(REGULAR)
NC^1
FO(COUNT)
ThC^0
FO
LOGTIME Hierarchy
AC^0