Meaning of Lexical Predicates and Tenses using Combinatory Logic and Topology

Jean-Pierre Desclés
Sorbonne University
LaLICC
« Language, Logics, Informatics, Cognition and Communication » Laboratory

0. Formal and Linguistic Framework
Cognitive and Applicative Grammar (CAG)

Semantico-cognitive representations

Lexical Analysis  
Lexical Synthesis

Logico-grammatical Representations
By means of applicative expressions

Categorial Analysis  
Categorial Synthesis

Morpho-syntactical Configurations of a Natural Language

Three Representation Levels

**Level 3**: Semantico-cognitive representations generated from schemes:

```
(λy. λx. { CONTR (CHANG (no (is-alive (y))) (is-alive (y))) z} x)
```

(λy.λx.l) (the-deer) (the-hunter)

**Level 2**: Logico-grammatical Order (operator–operand structure):

Logico-grammatical operations (aspects: tenses …) and grammatical roles (Agent, patient, instrument, location …):

PROC J0 (SAYS (EVENT F (killing (the-deer) (the-hunter)))) S0 & ( \[ d(F) < d(J0) \])

((killed (the-deer)) (the-hunter))

**Level 1**: Syntagmatic Order (concatenation structure):

Morpho-syntactical configurations: *The hunter killed the deer*
Semantico-cognitive Primitives

• \(< x \text{ REP } \text{Loc}(z) >\) (in French “repérage”) means "locating x in relation to the place \(\text{Loc}(z)\)" ;

• \(\text{INT } (\text{Loc}(z)\) means the topological interior of the place \(\text{Loc}(z)\);

• \(\text{EXT } (\text{Loc}(z)\) means the topological exterior of the place \(\text{Loc}(z)\);

• \(<\text{SIT1 MOV T SIT2}>\) means "a binary relation expressing a spatial movement, from the static situation \text{SIT1} to another static situation \text{SIT2} with:

\[
\begin{align*}
\text{SIT1} & = < x \text{ REP } \text{EXT } (\text{Loc}(z)) > \\
\text{SIT2} & = < x \text{ REP } \text{INT } (\text{Loc}(z)) >
\end{align*}
\]
1. Meaning of Lexical Predicates

An example
(1)  The ball enters the room

(1')  MOVT

( REP EXT (Loc (the-room)) the-ball )
( REP INT (Loc (the-room)) the ball )

Meaning of ENTERS’

•  [ enters' =def X MOV'T REP EXT INT ]

•  The meaning of the lexical predicate is given as a combination of the primitives

   MOV'T, REP, EXT and INT

   with an applicative program expressed by the Combinator ‘X’ of Curry’s Combinatory Logic
The ball enters the room

\[ \lambda y. \lambda x. \left[ \text{MOVT} \left( \text{REP} \left( \text{ext} \left( \text{Loc}(y) \right) \right) x \right) \left( \text{REP} \left( \text{int} \left( \text{Loc}(y) \right) \right) x \right) \right] \]

\[ \text{Enters'} \]

The lexical predicate enters' is considered as a "complex predicate". Its meaning is expressed by the "combination" of preceding cognitive primitives MOVT, REP, ext, int and Loc by means of a particular combinator \( X \)

\[ \text{[ enters'} =_{\text{def}} X \text{ MOVT REP ext int Loc } \]
Scheme of ENTERS’

(5) \( \lambda z. \lambda x. \{ \text{MOVT} \)
    ( \text{REP EXT} (\text{Loc}(z)) x) 
    (\text{REP INT} (\text{Loc}(z)) x \} 

where 'LOC(z)' is a variable whose the semantic type is: "place"
and 'x' a variable whose the semantic type is: "individual entity".

The meaning of (5) is expressed in English by:

(5’) "A Movement from outside of any place to inside of the same place".

\[ \beta \] -reductions

(5'') (\( \lambda z. \lambda x. \{ \text{MOVT} \)
    ( \text{REP EXT} (\text{Loc}(z)) x) 
    (\text{REP INT} (\text{Loc}(z)) x \} 

\[ \beta \] (\( \lambda x. \{ \text{MOVT} \)
    ( \text{REP EXT} (\text{Loc (the-room'}) x) 
    (\text{REP INT} (\text{Loc (the-room'}) x \} 

\[ \beta \]

\text{MOVT}
    ( \text{REP EXT} (\text{Loc (the-room'}) the-ball') 
    (\text{REP INT} (\text{Loc (the-room'}) the-ball')
The ball enters the room

Principal steps of deduction from applicative expression to its normal form

1. enters’ the-room’ the-ball’
2. [ enters’ =def X MOVT REP EXT INT ]
3. X MOVT REP EXT INT
4. [ X =def C (B B (B Ψ Φ3) ) Y ]
5. [ Y =def C ( B B2 B ) B ]
6. MOVT
   ( REP EXT (Loc (the-room)) the-ball )
   (REP INT (Loc (the-room)) the ball )
Applicative Program

Applicative expression of the combinator ‘X’

\[ X = \text{def } C (B B (B \Psi \Phi 3 ) ) Y \]

with : \[ Y = \text{def } C ( B B2 B) B \]

in terms of more elementary Combinators of Combinatory Logic

Integration Process

1. \( \text{MOV} ( \text{REP} (\text{ext}(\text{Loc}(y)))) x) (\text{REP} (\text{int}(\text{Loc}(y))) x) \)

2. 1. \( \text{REP} (\text{ext}(\text{Loc} y )) \)
   2. \( \text{REP} (B \text{ext Loc } y ) \)
   3. \( B \text{ REP} (B \text{ext Loc} ) y \)
   4. \( B2 (B \text{ REP}) B \text{ ext Loc } y \)
   5. \( B B2 B \text{ REP} B \text{ ext Loc } y \)
   6. \( C ( B B2 B) B \text{ REP ext Loc } y \)
   7. \[ Y = \text{def } C ( B B2 B) B \]
   8. \( Y \text{ REP ext Loc } y \)

\( Y \) is a combinator; it is an applicative program for combining the cognitive primitive
### Integration of enters1’

3. \( \text{MOVT} (Y \text{ REP ext Loc } y \ x) \ (Y \text{ REP int Loc } y \ x) \)

4. \( \Phi_3 \text{ MOVT } (Y \text{ REP ext}) (Y \text{ REP int}) \text{ Loc } y \ x \)

5. \( \Psi (\Phi_3 \text{ MOVT}) (Y \text{ REP ext int Loc } y \ x) \)

6. \( B (\Psi (\Phi_3 \text{ MOVT})) Y \text{ REP ext int Loc } y \ x \)

7. \( B (B \Psi \Phi_3 \text{ MOVT}) Y \text{ REP ext int Loc } y \ x \)

8. \( B \ B (B \Psi \Phi_3 \text{ MOVT}) Y \text{ REP ext int Loc } y \ x \)

9. \( C (B \ B (B \Psi \Phi_3)) \text{ Y MOV T REP ext int Loc } y \ x \)

10. \( [X = \text{ def } C (B \ B (B \Psi \Phi_3)) Y ] \)

11. \( X \text{ MOVT REP ext int Loc } y \ x \)

12. \( [\text{ enters1’ } = \text{ def } X \text{ MOVT REP ext int Loc } ] \)

13. \( \text{ enters1’ } y \ x \)

---

### Sceme of enters’ in John enters the room

\( (6) \quad \text{John enters the room} \)

\( (7) \quad "\text{An Agent has control over the process, that is, he has the ability to initiate or to stop it, or to maintain it".} \)

\( (6’) \quad \text{CONTR} \)

\( \text{MOVT} \)

\( (\text{REP (EXT (Loc (the-room’))) John’}) \)

\( (\text{REP (INT (Loc (the-room’))) John’}) \)

\( \text{John’} \)
Let the control scheme

(9) \( \lambda x. \lambda \Lambda. \{ \text{CONTR} (\Lambda) x \} \)

where ‘\( \Lambda \)’ designates a lexical scheme called “lexis” (from the old notion “\( \lambda e k t o n \)” of Stoicians).

This control scheme, as operator, acts onto the operand John’ and we obtain the \( \beta \)-relation:

(10) \( \lambda x. \lambda \Lambda. \{ \text{CONTR} (\Lambda) x \} ) ) \text{ John’} \\
\rightarrow \beta \lambda \Lambda. \{ \text{CONTR} (\Lambda) \} \text{ John’}

Scheme of ENTERS’ with a control

(10) \( \lambda z.\lambda x.\{ \text{CONTR} (\text{MOVT} \text{REP (EXT (Loc (z))) x } ) \text{REP (INT (Loc (z))) x } ) x \} \)

The meaning of this scheme is given by (10’):  

(10’) “Any Agent controls his movement from outside of any place to inside of the same”
John enters the room

Meaning of ENTERS

(11) \[ \text{enters'} = \text{def } Z \text{ CONTR MOVT REP EXT INT } \]

Where 'Z' is an applicative program expressed by a Combinator of Combinatory Logic
Polysemy of *to enter*

- These two examples of occurrences of the verb *enters* in the two different sentences

  *The ball enters the room*
  
  *John enters the room*

  show that the lexical item *enters* is polysemic, its meaning is described by two different cognitive schemes but, the first scheme (5) is embedded into the another (10).

---

**John enters the car into the garage**

\[
\text{SIT 1}[x,y] \xrightarrow{\text{MOVT}} \text{SIT 2}[x,y]
\]

\[
< y \ \text{REP} (\text{ext\,(Loc\,(z)))} > \xrightarrow{\text{CONTR}} < y \ \text{REP} (\text{int\,(Loc\,(z)))} >
\]

\[
\text{Enters 2}'(z, y, x)
\]

\[
[\text{enters 3}' \equiv \text{U CONTR MOVT REP ext int Loc}]
\]

\[
x := \text{An Agent}
\]

\[
y := \text{a Patient}
\]

\[
z := \text{a Localizer}
\]
INTEGRATION OF Enters’3

CONTR ( MOVT ( REP (EXT (Loc(z))) y) (REP (INT (Loc(z))) y) x

\[ \text{enters}\prime = \text{def } X \text{ MOVT REP EXT INT Loc } \]

CONTR ( enters\prime z y ) x

B_2 \text{ CONTR enters}\prime z y x

\[ \text{enters}'3 = \text{def } B_2 \text{ CONTR enters}'\prime \]

enters’3 z y x

2. Meaning of tenses

Some examples
Utterance Scheme

(12) \( \lambda \Lambda. \{ \text{PROCESS } J_0 \)

( \text{SAYS}

( \text{ASP } J_1 (\Lambda) )

S_0

& [J_1 \text{ REL } J_0] \}

(12') “The speaker S_0 SAYS (enunciates), during the interval [J_0[, with an open bound at the left, (that) any aspectual predicative relation L is realized on the interval J_1, whose the topological properties are determined by aspectual choices. Furthermore, a relation REL (‘=’, ‘<’ ou ‘>’) holds between the two intervals J_1 and J_0.”

Different sentences with the same predicative relation

These sentences

the hunter killed a rabbit,
the hunter has killed a rabbit,
the hunter is killing a rabbit,
the hunter is going to kill a rabbit,
the hunter will kill a rabbit …,

use different tense markers.
Basic Aspects

- Event
- State
- Left close boundary
- Right open boundary
- (Unaccomplished) Process

Past event: *The hunter killed the rabbit*

- Past event:
  \[ \lambda \Lambda. \{ \text{PROCESS J0 (SAYS (EVENT F1 (\Lambda ) S0) & } [F1] < [JO[ ] } \]

- Means: “the past event is located before the speaking process”

\[ \framebox{\begin{array}{c}
F1 \\

\end{array}} \leq \framebox{\begin{array}{c}
T0 \\

\end{array}} \]
Progressive Present:

*The hunter is killing the rabbit*

\[ \lambda \Lambda. \{ \text{PROCESS } J_0 \]
\[(\text{SAYS (PROCESS } J_1, \Lambda ) \text{ S0 )}
\& [ \delta(J_1) = \delta(J_0) ] \}

Means: “the (unaccomplished) process is concomitant with the speaking process (hence the two open right bounds of the two processes are identical).

Meaning of Present Perfect

- Means: “the resulting-state is concomitant with the speaking process (hence, the right boundaries are open and identical) AND there is an event before this resulting state such that the event and the resulting state are contiguous (hence the right close boundary of event is identical with the open left boundary of the resulting state).”
Present Perfect:
The hunter has killed the rabbit

\[ \lambda \Delta \{ \text{PROCESS } J_0 \ (\text{SAYS}) \]

(RESULTING-STATE \( O_1 \) (\( \Lambda \)) \( S_0 \))

& \[ \delta(O_1) = \delta(JO) \]

& EXISTS (EVENT \( F_1 \) (\( \Lambda \)))

& \[ F_1 < J_0 \] & \[ \delta(F_1) = \gamma(O_1) \]

\[ \delta(F_1) = \gamma(O_1) \]

\[ \delta(O_1) = \delta(J_0) \]

EVENT and contiguous STATE

\[ \gamma(F) \]

\[ \delta(F) = \gamma(O) \]

\[ \delta(O) \]
Prospective Present:
*The hunter is going to kill the rabbit*

- Approximate representation:
  \[\lambda\lambda. \{ \text{PROCESS } J_0 \]
  \[(\text{SAYS (QUASI-CERTAIN (EVENT } F_1 (\Lambda)) S_0) \]
  \& \[\gamma(F_1) > \delta(J_0) \}\]

- Means: “the event is located after the speaking process (hence the close left bound of the event comes after the open right bound of the speaking process) AND the event is construed as being quasi-certain.”

Future tense:
*The hunter will kill the rabbit*

\[\lambda\lambda. \{ \text{PROCESS } J_0 (\text{SAYS} \]
\[(\text{PROBABLE (EVENT } F_1 (\Lambda)) S_0) \]
\& \[\gamma(F_1) > \delta(J_0) \}\]

Means: “the event is located after the speaking process (hence the close left bound of the event comes after the open right bound of the speaking process) AND the event is construed as being probable but not certain.”
Past Event:

\[ \text{Past Event:} \]

Progressive Present:

\[ \delta(O1) = \delta(J0) \]

Present Perfect (present result of a past event):

\[ \delta(F1) = \gamma(O1) \]

\[ \delta(J0) = \delta(J0) \]

Prospective Present:

\[ \delta(J0) \]

\[ \gamma(F1) \]

Quasi-certain Event

Future Tense:

\[ \delta(J0) = T0 \]

Probable Event

Possible negative Event
3. Integration of Lexical Predicates inside of Aspectual and Temporal Scheme

An example

Scheme of \((TO \ KILL \ (y)) \ (x)\)

\[
\lambda y. \lambda x. \{ \text{CONTR} \\
\quad \text{(CHANG}_F \\
\quad \quad \text{(is-alive } y \text{)} \circ_1 \\
\quad \quad \text{(NO (is-alive } y\text{)) } \circ_2 \) \\
\quad x \}
\]

\(\delta(\circ_1) = \gamma(F)\)
\(\delta(F) = \gamma(\circ_2)\)

y is alive

y is not alive
Past event
\[ \lambda y \lambda x. \{ \text{PROCESS J0} \ ( \text{SAYS} \ (\text{EVENT F1} \ (\text{CONTR} \ (\text{CHANG F1} \ (\text{is-alive} y) \ (\text{NO} \ (\text{is-alive} y))) \ x)) \ ) \} \]

\& \ [ d(F1) < d(J0) ] \}

with its temporal diagram:

```
<table>
<thead>
<tr>
<th>is-alive</th>
<th>EVENT (TO KILL)</th>
<th>NO(is-alive)</th>
</tr>
</thead>
<tbody>
<tr>
<td>O1</td>
<td>F1</td>
<td>O2</td>
</tr>
</tbody>
</table>
```

\[ \delta(F1) < \gamma(J0) \]

Progressive present:
\[ \lambda y \lambda x. \{ \text{PROCESS J0} \ ( \text{SAYS} \ (\text{PROCESS J1} \ (\text{CONTR} \ (\text{CHANG F1} \ (\text{is-alive} y) \ (\text{NO} \ (\text{is-alive} y))) \ x)) \ ) \} \]

\& \ [ [ \delta(J1) = \delta(J0) ] \ & \ [ J1 \subset F1 ] \ & \ [ \gamma(J1) = \gamma(F1) ] ] \}

with its temporal diagram:

```
<table>
<thead>
<tr>
<th>is-alive</th>
<th>EVENT (TO KILL)</th>
<th>NO (is-alive)</th>
</tr>
</thead>
<tbody>
<tr>
<td>O1</td>
<td>Is killing J1</td>
<td>O2</td>
</tr>
</tbody>
</table>
```

\[ \gamma(J1) = \gamma(F1) \]

\[ \delta(J1) = \delta(J0) \]
Present Perfect (= state of an acquired situation by subject)
\[ \lambda y. \lambda x. (\text{PROCESS } J0) \]
\[ (\text{SAYS}) \]
\[ (\text{STATE } O) \]
\[ (\text{PROPERTY-OF }) \]
\[ (\text{EVENT } F1) \]
\[ (\text{CONTR}) \]
\[ (\text{CHANG } F1) \]
\[ (\text{is-alive } y) \]
\[ (\text{NO (is-alive } y)) \]
\[ (x)) \]
\[ ) \]

with its temporal diagram:

Prospective Present (process of a movement of the Agent towards a quasi-certain event):
\[ \lambda y. \lambda x. (\text{PROCESS } J0) \]
\[ (\text{SAYS}) \]
\[ (\text{PROCESS } J0) \]
\[ (\text{GOING-TOWARDS}) \]
\[ (\text{QUASI-CERTAIN}) \]
\[ (\text{EVENT } F1) \]
\[ (\text{CHANG } F1) \]
\[ (\text{is-alive } y) \]
\[ (\text{NO (is-alive } y)) \]
\[ (x)) \]
\[ ) \]

\[ & [ \delta (O) = \delta (J0) ] \land [ \gamma (O) = \delta (F1) ] \land [ O \subseteq O2 ] \land [ \delta (F1) < \delta (J0) ] \]
Future Tense:

\[ (23) \lambda x. \{ \text{PROCESS } J_0 (\text{SAYS} \text{ (PROBABLE} (\text{EVENT} F_1 \text{ (CONTR} (\text{CHANG} F_1 \text{ (to-be-alive } x) O_1 \text{ (NO} (\text{to-be-alive } x) O_2 x))))) S_0 ) \} \]

\& [ γ(F_1) > δ(J_0) ]

with its temporal diagram:

**Temporal diagrams: Past Event and Progressive Present**

**Past Event:**

- is-alive
- To KILL
- NO (is-alive)

**Progressive Present:**

- is-alive
- EVENT (TO KILL)
- NO (is-alive)

\[ γ(J_1) = γ(F_1) \]

\[ δ(J_1) = δ(J_0) \]
**Present Perfect**

Present Perfect:

\[ \&(F1) = \gamma(O) \]

**Prospective Present and Future Tense**

Prospective Present:

\[ \gamma(F1) > \&(J0) \]

Future Tense:

\[ \gamma(F1) > \&(J0) \]
4. Formal Inference from a sentence

Formal Inferential Deduction

1. This morning, the hunter killed the rabbit
2. (this morning) ((killed the-rabbit) (the-hunter))
3. (this morning) (EVENT F1 (TO KILL the-rabbit) (the-hunter))
   & [EVENT F1 ((TO KILL the-rabbit) (the-hunter)) < PROCESS J0 (SAYING S0)]
4. [STATE (now) IS-INCULDED-IN (STATE (this morning)]
5. [EVENT F1 ((TO KILL the-rabbit) (the-hunter)) IS-INCULDED-IN STATE (this morning)]
Formal Inferential Deduction (2)

6. $(\text{STATE O2 (NO (is-alive the-rabbit)))}
\& [(\text{STATE O (now)}) \text{-IS-INCLUDED-IN} \ (\text{STATE O2 (NO (is-alive the-rabbit)))}]
\& [(\text{PROCESS J0 (SAYING S0)}) \text{-IS-INCLUDED-IN} \ (\text{STATE O (now)})]
\& [d (\text{STATE O (now)}) = d (\text{PROCESS J0 (SAYING S0)})]

7. $[\text{is-not-alive (the-rabbit)} = \text{def NO (is-alive (the-rabbit))}]

8. $[\text{is-dead} = \text{def is-not-alive}]

9. $(\text{STATE O2 (is-dead the-rabbit)})
\& [(\text{STATE O (now)}) \text{-IS-INCLUDED-IN} \ (\text{STATE O2 (is-dead the-rabbit)})]
\& [(\text{PROCESS J0 (SAYING S0)}) \text{-IS-INCLUDED-IN} \ (\text{STATE O (now)})]
\& [d (\text{STATE O (now)}) = d (\text{PROCESS J0 (SAYING S0)})]

Formal Inferential Deduction (3)

10. $(\text{STATE O (is-dead the-rabbit)})
\& [\delta (\text{STATE O (now)})
= \delta (\text{PROCESS J0 (SAYING S0)})]

11. Now (is-dead the-rabbit)

12. Now, the rabbit is dead
Annexes

ANNEX 1: COMBINATORS

<table>
<thead>
<tr>
<th>λ-closed expressions</th>
<th>reduction schemes</th>
</tr>
</thead>
<tbody>
<tr>
<td>I = λx. x</td>
<td>IX -&gt; β X</td>
</tr>
<tr>
<td>C = λxyz.zxy</td>
<td>CXYZ -&gt; β XZY</td>
</tr>
<tr>
<td>C* = λxy.yx</td>
<td>C*XY -&gt; β XY</td>
</tr>
<tr>
<td>B = λxyz.yz(x)</td>
<td>BXYZ -&gt; β X(YZ)</td>
</tr>
<tr>
<td>W = λxy.xyy</td>
<td>WXY -&gt; β XYY</td>
</tr>
<tr>
<td>S = λxyz.xz(yz)</td>
<td>SXZY -&gt; β X(ZY)</td>
</tr>
<tr>
<td>K = λxy.x</td>
<td>KXY -&gt; β X</td>
</tr>
<tr>
<td>Φ = λxyz.u(x(y)(z))</td>
<td>Φ XYZU -&gt; β X(YU)(ZU)</td>
</tr>
<tr>
<td>Ψ = λxyuv.x(y)(v)</td>
<td>Ψ XYZU -&gt; β X(YZ)(YU)</td>
</tr>
</tbody>
</table>
ANNEX 2

Calculus of Combinators ‘X’ and ‘Z’ for integrating cognitive primitives into a whole

Calculus of ‘X’

• 1. enters’ (the-room’) (the-ball’).
   hyp.
• 2. [ enters’ =def X MOVT REP EXT INT ]
• 3. [ X =def C(BB(BΨ BΦ BΦ)B]
• 4. X MOVT REP EXT INT ((the-room’’) (the-ball’’)
• 5. C(BB(BΨ BΦ)B MOVT REP EXT INT (the-room’) (the-ball’)
• 6. BB(BΨ BΦ) MOVT B REP EXT INT (the-room’) (the-ball’)
• 7. B((BΨ BΦ) MOVT) B REP EXT INT (the-room’) (the-ball’)
• 8. (BΨ BΦ) MOVT (B REP ) EXT INT (the-room’) (the-ball’)
• 9. Ψ (BΦ MOVT) (B REP ) EXT INT (the-room’) (the-ball’)

03/02/2009
Calculus of the Combinator ‘Z’

1. CONTR(MOV'T(REP EXT (the-room')(John'))( REP INT (the-room')(John')))(John')
2. 1. MOV'T (REP EXT (the-room')(John')) (REP INT (the-room')(John'))
3. 2. \[ X \text{ def C(BB (BΨΨ ΨΨ BΦΦ ΦΦ ΦΦ ΦΦ)B)} \]
4. 3. \[ \text{enters'}1 = X \text{ MOV'T REP EXT INT} \]
5. 4. enters’1 (the-room’) (John’)
6. CONTR (enters’1 (the-room’) (John’)) (John’)
7. B CONTR (enters’1 (the-room’) (John’))
8. B (B CONTR) (enters’1) (the-room’) (John’)
9. BBB CONTR (enters’1) (the-room’) (John’)
10. \[ \text{enters’2 =def BBB CONTR (enters’1)} \]
11. enters’2 (the-room) (John’)
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>[ enters'2 = \texttt{BBB CONTR (enters'1)} ]</td>
</tr>
<tr>
<td>2.</td>
<td>\texttt{BBB CONTR (enters'1)}</td>
</tr>
<tr>
<td>3.</td>
<td>[ enters'1 = X MOVТ REP \texttt{EXT INT} ]</td>
</tr>
<tr>
<td>4.</td>
<td>\texttt{BBB CONTR (X MOVТ REP \texttt{EXT INT})}</td>
</tr>
<tr>
<td>5.</td>
<td>\texttt{B5 (BBB CONTR) X MOVТ REP \texttt{EXT INT}}</td>
</tr>
<tr>
<td>6.</td>
<td>\texttt{BB5 BBB CONTR X MOVТ REP \texttt{EXT INT}}</td>
</tr>
<tr>
<td>7.</td>
<td>\texttt{C(BB5 BBB) X CONTR MOVТ REP \texttt{EXT INT}}</td>
</tr>
<tr>
<td>8.</td>
<td>[ X = \texttt{C(BB (BΨ Bφφ))B} ]</td>
</tr>
<tr>
<td>9.</td>
<td>\texttt{C(BB5 BBB)(C(BB (BΨ Bφφ))B) CONTR MOVТ REP \texttt{EXT INT}}</td>
</tr>
<tr>
<td>10.</td>
<td>[ Z = \texttt{C(BB5 BBB)(C(BB (BΨ Bφφ))B)} ]</td>
</tr>
<tr>
<td>11.</td>
<td>Z CONTR MOVТ REP \texttt{EXT INT}</td>
</tr>
<tr>
<td>12.</td>
<td>[ enters'2 = Z CONTR MOVТ REP \texttt{EXT INT} ]</td>
</tr>
<tr>
<td>24.</td>
<td>(Z CONTR MOVТ REP \texttt{EXT INT})(the-room) (John')</td>
</tr>
</tbody>
</table>