Classical Linguistic Inference II:

Computational Linguistics and Theorem Proving in a Computer Game

Seminar „Linguistic Inference and Textual Entailment“

Michaela Regneri

03.07.2007

Motivation

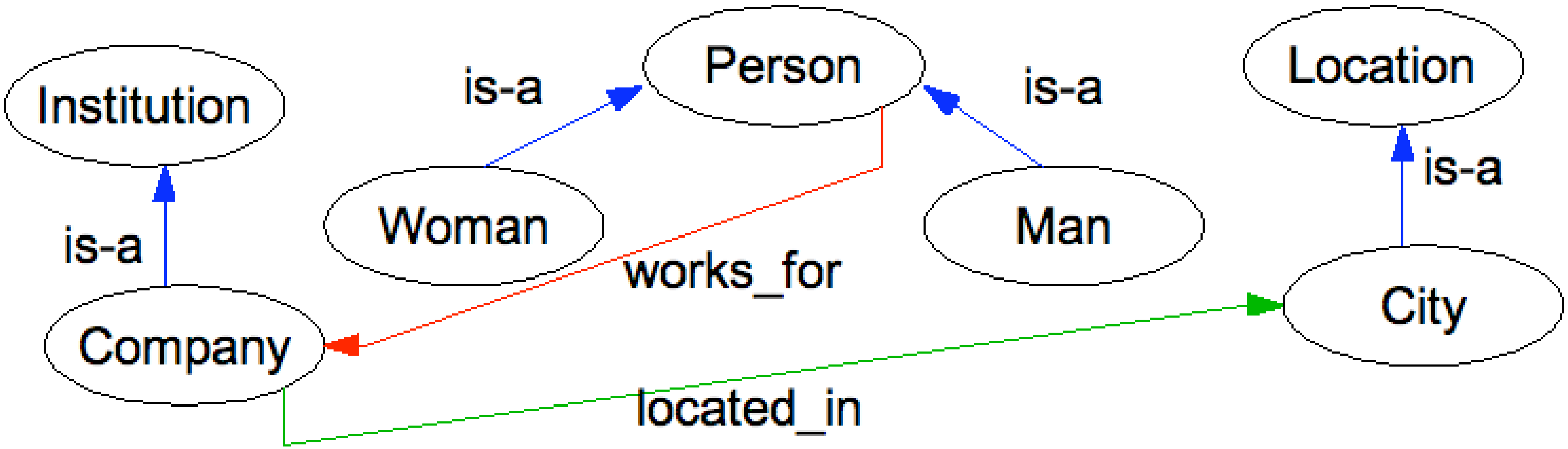
- First order logic is not decidable; running a prover may take forever
- user-oriented applications mostly require knowledge application within two seconds
- restricting the logic‘s expressive power can fasten reasoning enormously (and make it terminate for sure)
- show that a tradeoff between expressiveness and computational tractability is possible for some applications

Outline

- Description Logic in a nutshell
  - Basics and Terms
  - RACER
- Computational Linguistics & Theorem Proving in a Computer Game
  - Motivation and System Overview
  - The components in detail
- Summary & Conclusion

Description Logic - basics

- designed for knowledge representations



- allowing to encode general knowledge (as above) as well as world models (with individuals, s.a. „person(john)“)

Description Logic - basics (cont.)

- T-Box: The world's rules (as described in the knowledge base)

$$\begin{aligned} man &\sqsubseteq person \\ woman &\sqsubseteq person \\ city &\sqsubseteq location \\ &\forall located\_in.location \\ &\ldots \end{aligned}$$

- A-Box: Relations between and properties of individuals

person(mary)	works_for(mary, c1)
person(john)	located_in(NY, c1)
loves(mary, john)	woman(mary)
loves(john, mary)	man(john)

Description Logic - Terms

- (atomic) concepts C denoting sets of individuals (*person*)
  ≈ unary predicates in FOL
- (atomic) roles R: (*loves*) ≈ binary predicates in FOL
- complex concepts:
  - conjunction and disjunction of concepts: $C_1 \sqcap C_2$, $C_1 \sqcup C_2$
  - negation (the complementary concept): $\neg C$
  - existential restriction: $\exists R.C$ (set of all *a* having an *x* s.t. R(a,x) & C(x))
  - value restriction: $\forall R.C$ (set of all *a* s.t. for all *x s.t.* R(a,x), C(x) holds)

Description Logic - Terms (cont.)

- inverse roles R^{-1}: loves(john, mary) ≡ $loves^{-1}$(mary, john)
- the empty concept ⊥ and the universal concept ⊤
- concept equality: C1 $\doteq$ C2
  (abbreviates C1 ⊑ C2 ∧ C2 ⊑ C1)
- ‚at most' and ‚at least' number restrictions:
  $\exists_{\leq m}R$: Set of all a s.t. there are at most m (different) *x* for which
  R(a,x) holds

Description Logic - Example

A-BOX

man(john)	loves(john,mary)
woman(mary)	loves(mary,sam)
man(sam)	married(sam,sue)
woman(sue)	happy(sam)

Some assertions...

...and some rules:

T-BOX

bachelor $\doteq \neg \exists$married.$\top \sqcap$ man	*„bachelors are unmarried men“*
married $\doteq$ married^{-1}	(being married to so. is reflexive)
$\exists$married.$\top \sqsubseteq$ happy	*„all married people are happy“*
$\exists_{\geq 2}$ love $\sqsubseteq \bot$	*„you can love at most one person“*
$\exists$married.woman $\sqsubseteq \exists$love.woman	*„someone married to a woman also loves a woman“*

Description Logic - RACER

- a reasoner for description logic
- provides reasoning with T-Boxes and (multiple) A-Boxes
- performs consistency checks (of A-Boxes, T-Boxes or both)
- several retrieval tasks:
    - all individuals of a concept, all concepts of an individual
    - check for subsumption („*are cities locations?*“)

Description Logic - RACER (cont.)

- several retrieval tasks:
  - find the *parent concepts* parents of C are the most specific C‘ s.t. C $\sqsubseteq$ C‘ (*children* analogously)
  - find *predecessors* (*successors*): predecessors of C are all C‘ s.t. C $\sqsubseteq^*$ C‘ (*successors* analogously)
  - determine *domain* and *fillers* of a role:

    fillers of R are all f s.t. $\exists x.R(x,f)$ ($\doteq \exists R^{-1}.\top$)

    domain of R consists of all d s.t. $\exists x.R(d,x)$ ($\doteq \exists R.\top$)

Description Logic - RACER (cont.)

- Example queries:

  Is Sue happy?
  (Does ‚happy' contain Sue?)

  Can Mary love John?
  (loves(mary, john) -> consistent?)

  What properties does Mary have?
  (Concepts containing mary)

A-BOX

man(john)	loves(john,mary)
woman(mary)	loves(mary,sam)
man(sam)	married(sam,sue)
woman(sue)	happy(sam)

T-BOX

$\text{bachelor} \doteq \neg \exists \text{married}.\top \sqcap \text{man}$

$\text{married} \doteq \text{married}^{-1}$

$\exists \text{married}.\top \sqsubseteq \text{happy}$

$\exists_{\geqq 2} \text{love} \sqsubseteq \bot$

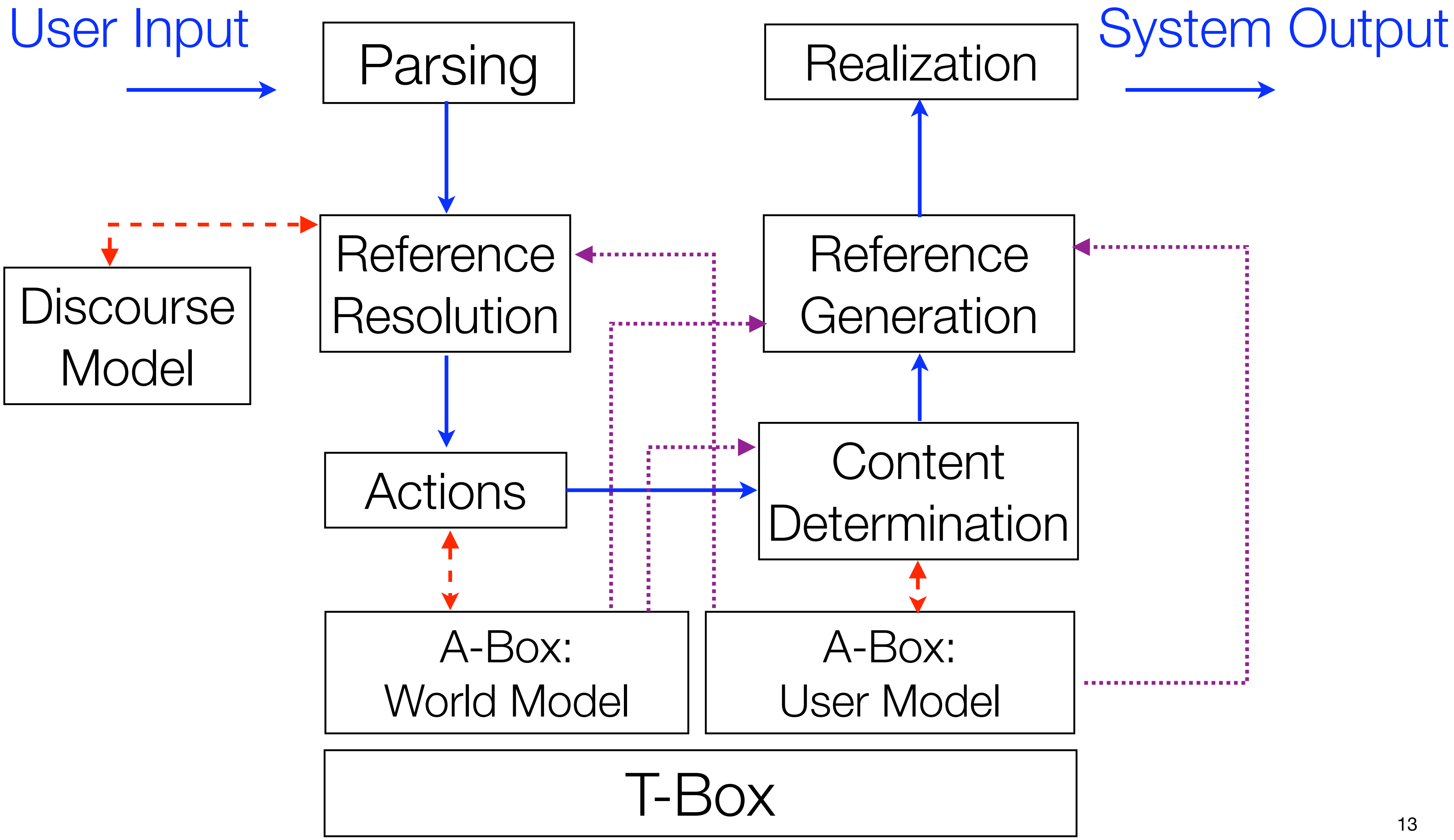
$\exists \text{married}.\text{woman} \sqsubseteq \exists \text{love}.\text{woman}$

Computational Linguistics & Theorem Proving in a Computer Game

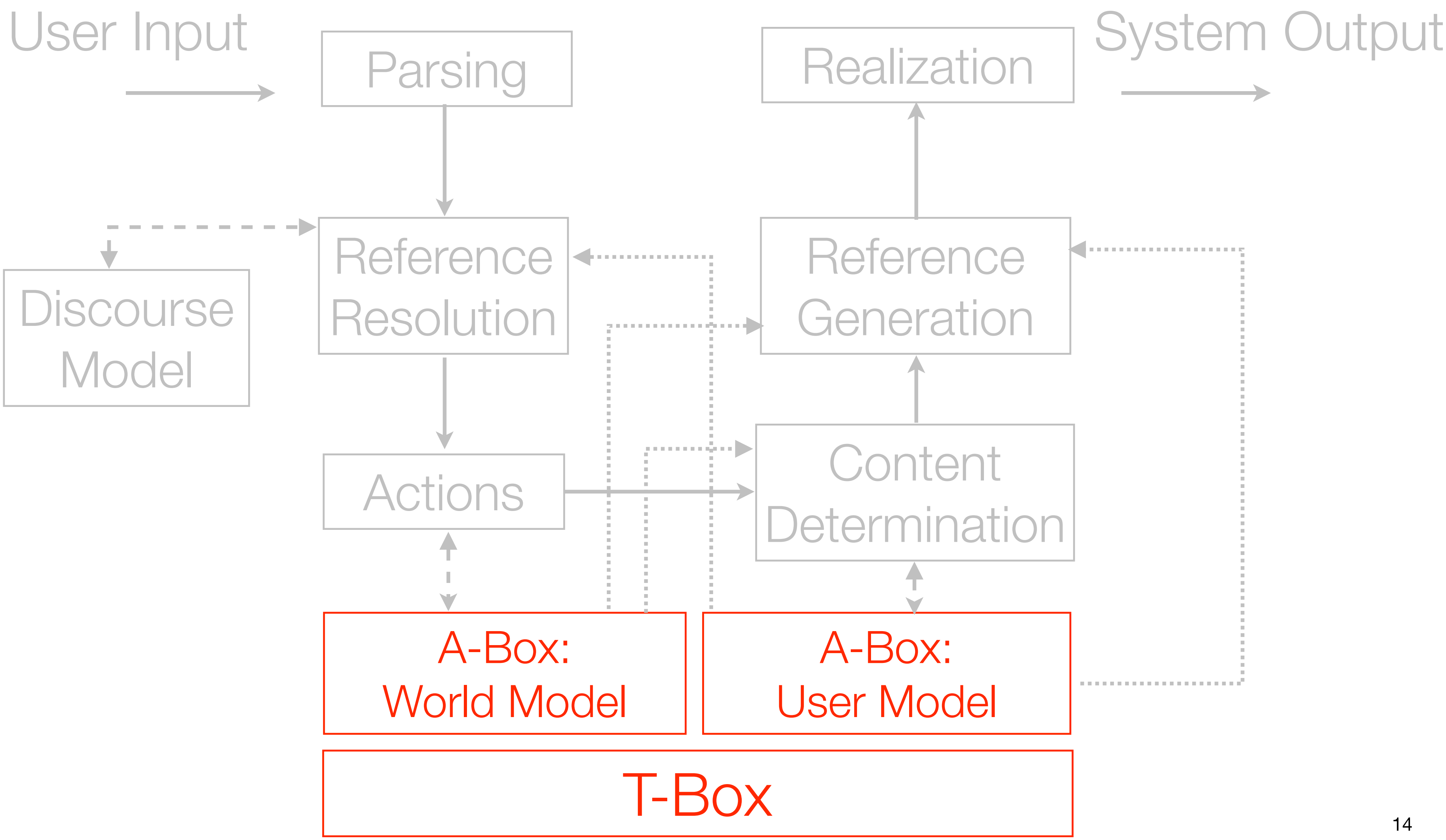
- see Koller et al. 2004
- task (of a student software project): Make use of syntactic and semantic processing to make the user input as convenient as possible, i.e. provide flexible possibilities to refer to things
- example for interaction in a text adventure („A Bear‘s Night Out“):

```
Your warm winter jacket is here,
which may be just as well, it's a
little chilly.

>look at the jacket
A smart green jacket with big
pockets, teddy bear sized.

>take the smart green jacket
You can't see any such thing.

>take the jacket with big pockets
I only understood you as far as
wanting to take the green jacket.

>take the teddy bear sized jacket
You can't see any such thing.

>take the jacket
Taken.
```

System Overview



The knowledge base



The knowledge base

- two A-Boxes: one for the user, one for the world model
- user knowledge and world model may be contradicting (but usually, the first is a subset of the latter)
- one unique individual ‚myself‘ representing the (only) player

room(kitchen)	red(a1)	has-location(a1, b2)
player(myself)	green(a2)	has-location(a2, kitchen)
table(t1)	bowl(b1)	has-detail(a2,w1)
apple(a1)	bowl(b2)	has-location(myself, kitchen)
apple(a2)	has-location(t1, kitchen)	has-location(b2, kitchen)
worm(w1)	has-location(b1, t1)	kitchen(kitchen)

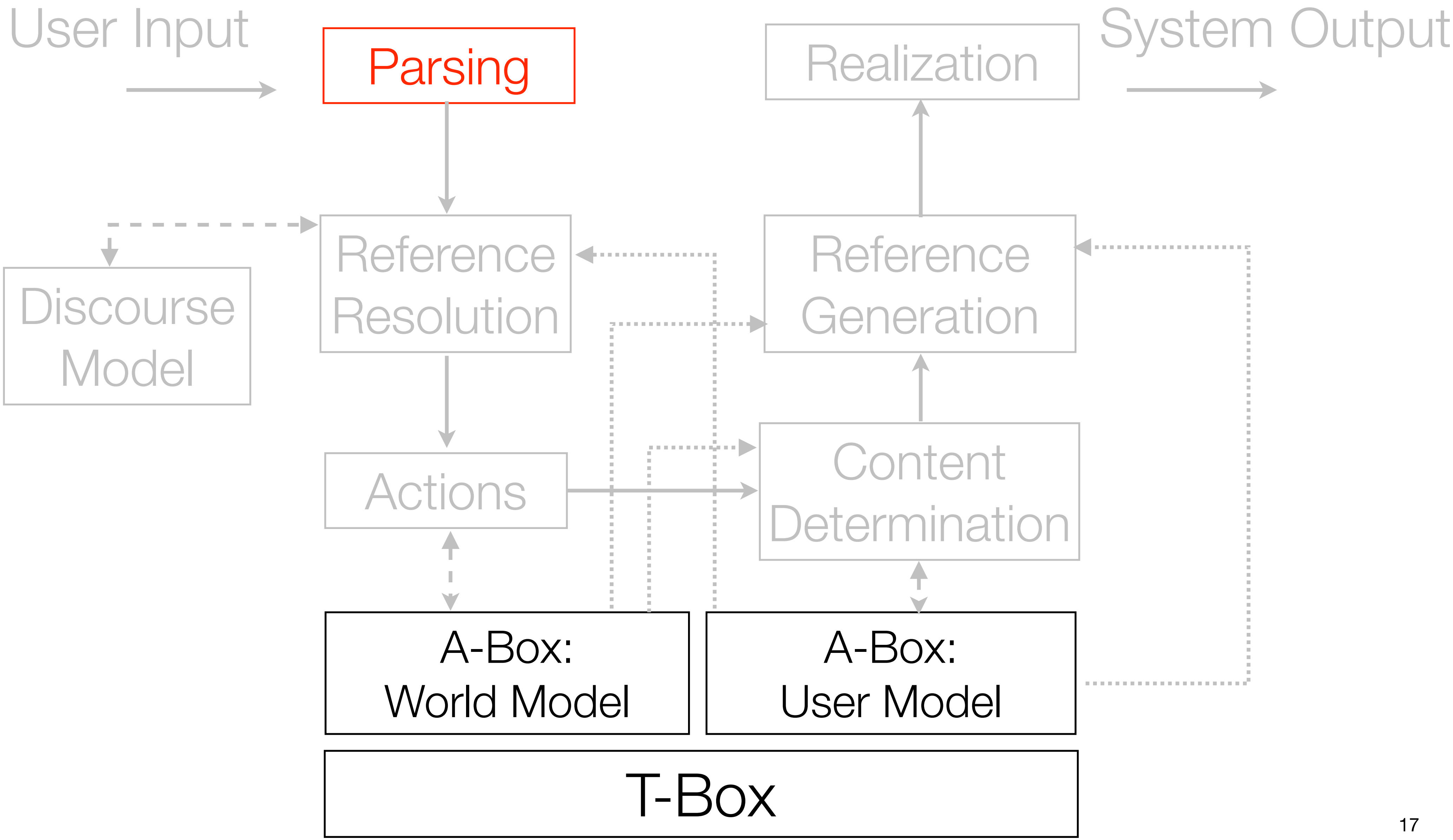
The knowledge base (cont.)

- T-Box: Axioms holding for the user and the world model
  - concept hierarchies: apple $\sqsubseteq$ object, red $\sqsubseteq$ color
  - complex concepts:

    $$\text{here} \doteq \exists \text{has-location}^{-1}.\text{player}$$

    $$\text{accessible} \doteq \forall \text{has-location.here} \sqcup \forall \text{has-location (accessible} \sqcap \text{open)}$$

Parsing & Syntax-Semantics Interface



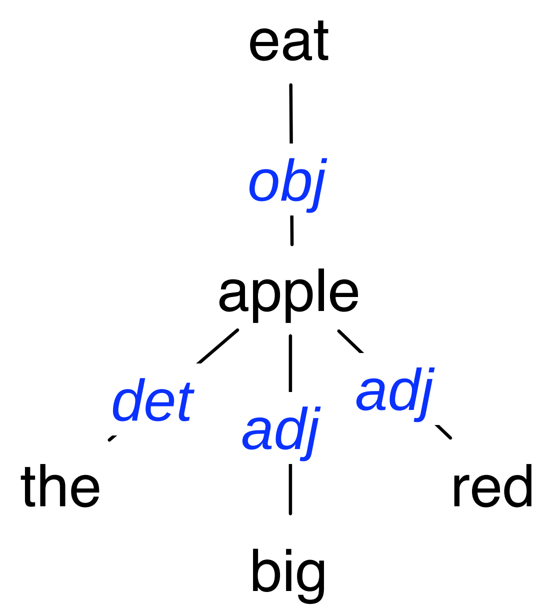
Parsing

- parse user input with a constraint-based TDG (Topological Dependency Grammar) parser; use the syntax output:

Eat the big red apple.

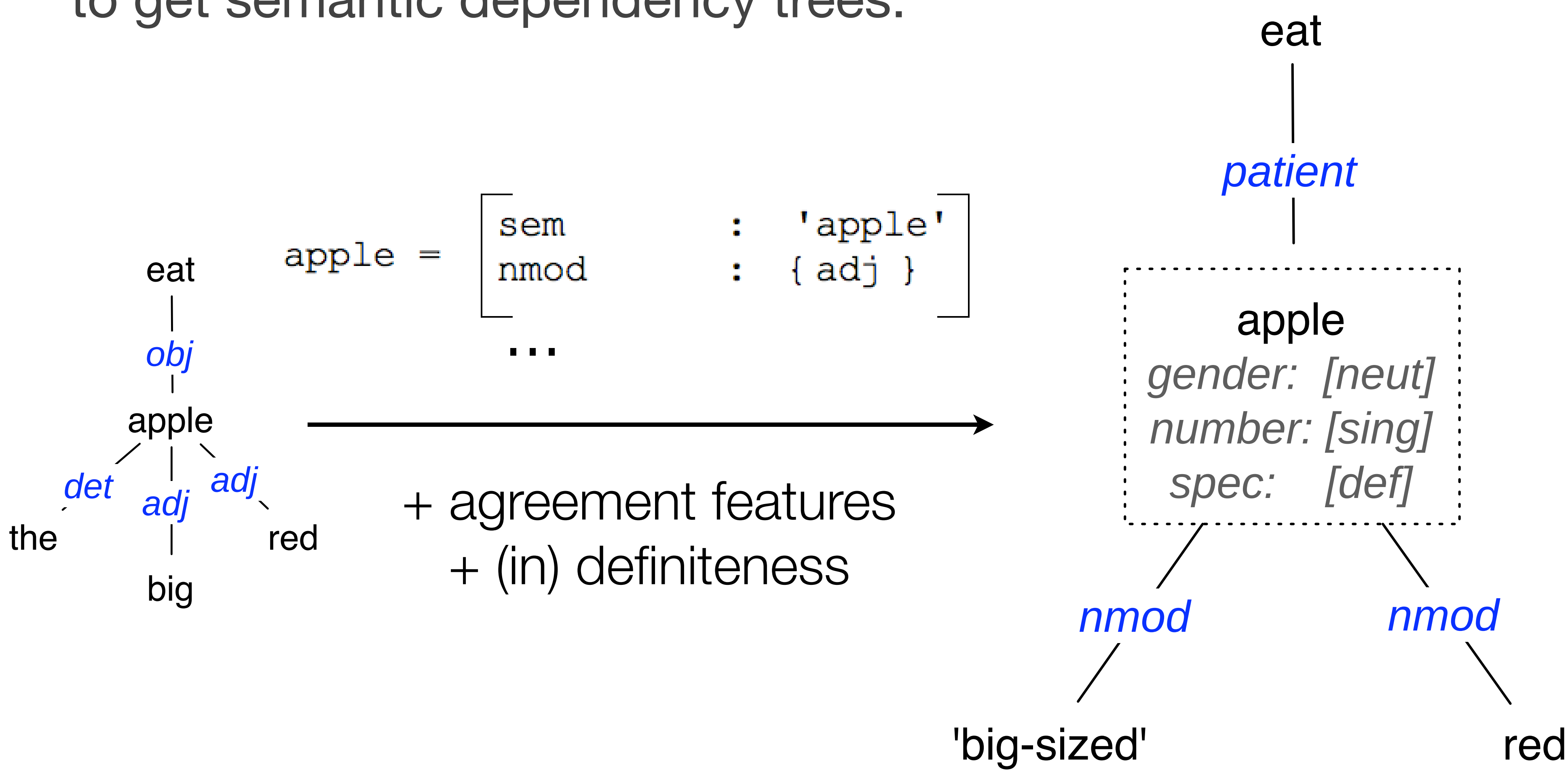
lexical entries

```
eat   = [ in   :  {   }
          out  :  { (subj?), (obj!)} ]

...

apple = [ in   :  { subj, obj }
          out  :  { (det?), (adj.*)} ]
```

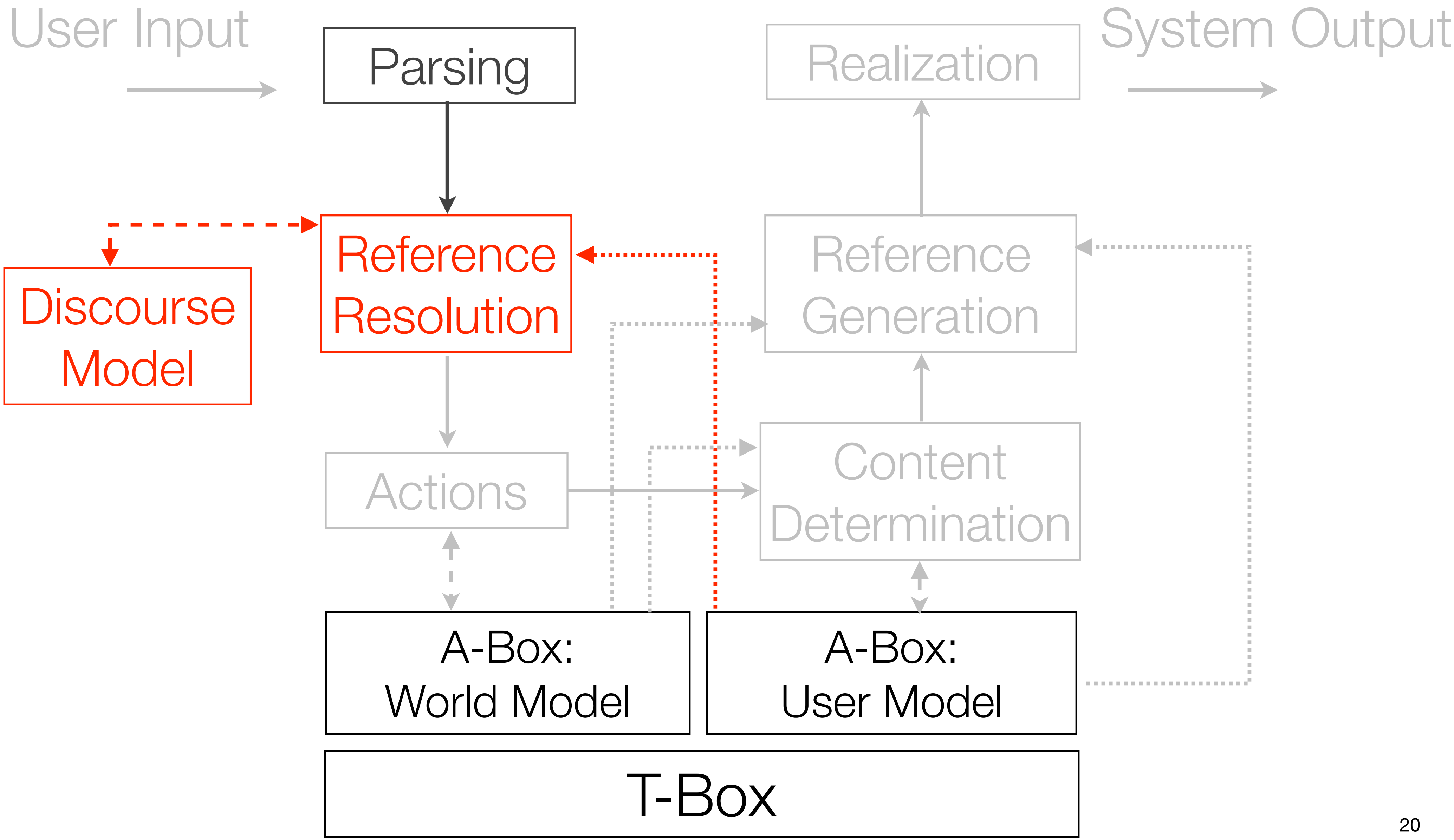


Syntax-Semantics Interface

- specify additional semantic constraints (depending on syntax) to get semantic dependency trees:



Reference Resolution



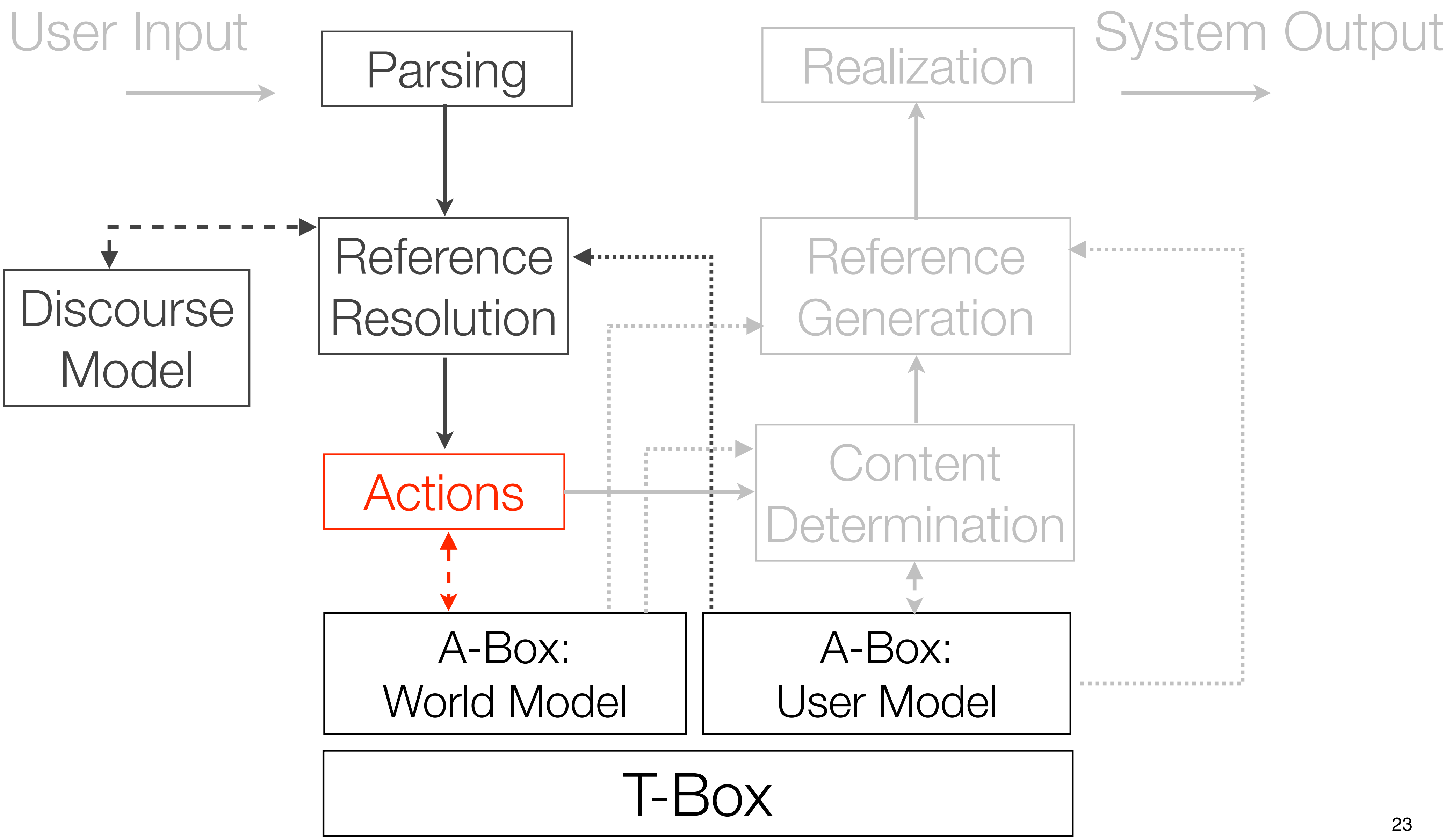
Reference Resolution

- relate the semantics of the user input to the knowledge base with help of RACER
- all inferences on base of the user‘s A-Box (to avoid unnecessary ambiguity)
- ask racer for individuals which are ‚*visible*‘ in the game
- definite NPs:
  the apple: $apple \sqcap visible$
  the apple with the worm: $apple \sqcap (\exists has\text{-}detail.worm) \sqcap visible$

Reference Resolution (cont.)

- indefinite NPs are treated like definite NPs, but here a *random* instance with the right properties is picked
- pronoun resolution: Find the most salient referent with a discourse model
  - consider agreement features
  - find the preferred referent according to a saliency list (entities of the last utterance are more salient than entities of the preceding utterances e.g.)
- avoid conflicts (reflexive pronouns e.g.) by restricting the input grammar

User Actions



User Actions

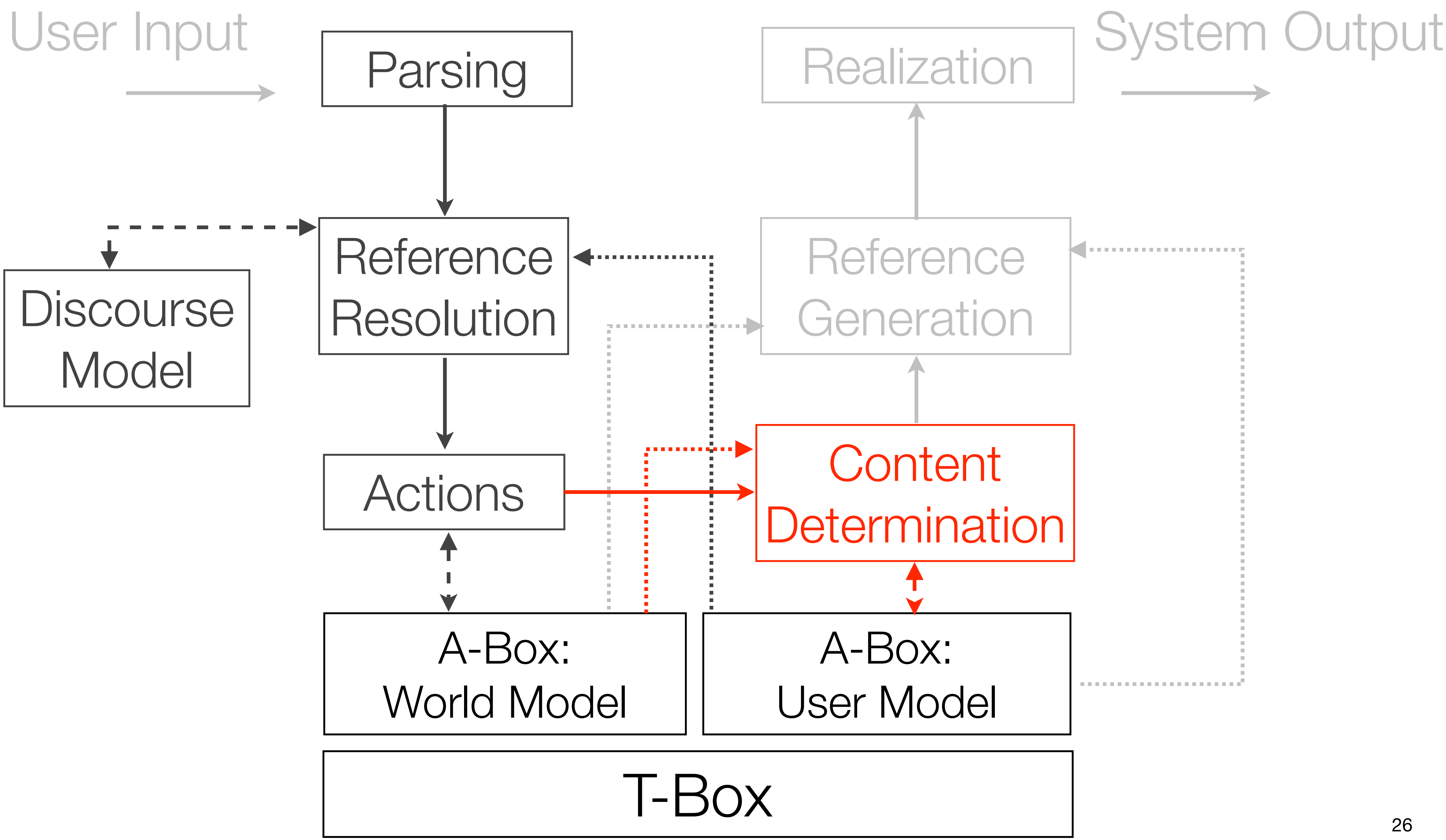
- user input may change the state of the world
- input is interpreted as action, like in STRIPS operators:

take(patient: x)	
preconditions	accessible(x), takable(x), not(inventory-object(x))
effects (world model)	***add***: related(x myself has-location) ***delete***: related(x individual-filler(x, has-location), has-location)
effects (user knowledge)	***add***: related(x myself has-location) ***delete***: related(x individual-filler(x, has-location), has-location)

User Actions (cont.)

- pre-defined action operators contain several unresolved „placeholders“ (as the model changes in the game):
  - ‚x‘ will be instantiated with some individual in the knowledge base if the preconditions are fulfilled
  - ‚individual-filler(x, R)‘ will be resolved to a current individual i for which R(x,i) holds
- if there are ambiguities (due to parsing or pronoun resolution), all readings are tried:
  - if only one reaches a consistent knowledge base, this one is taken
  - if there are still ambiguities, the user has to resolve them

Content Determination



Content Determination

- decide what to tell the user as a reaction to his / her input
- straight forward for actions like ‚*take*‘: inform the user about the new assertions („add“), assume the removal of information („delete“) can be inferred
- more complex actions for ‚describe‘ (no knowledge base change):
  - *describe(x)* triggers enumeration of x‘s properties
  - query RACER for the parent concepts of x and roles with x

Content Determination (cont.)

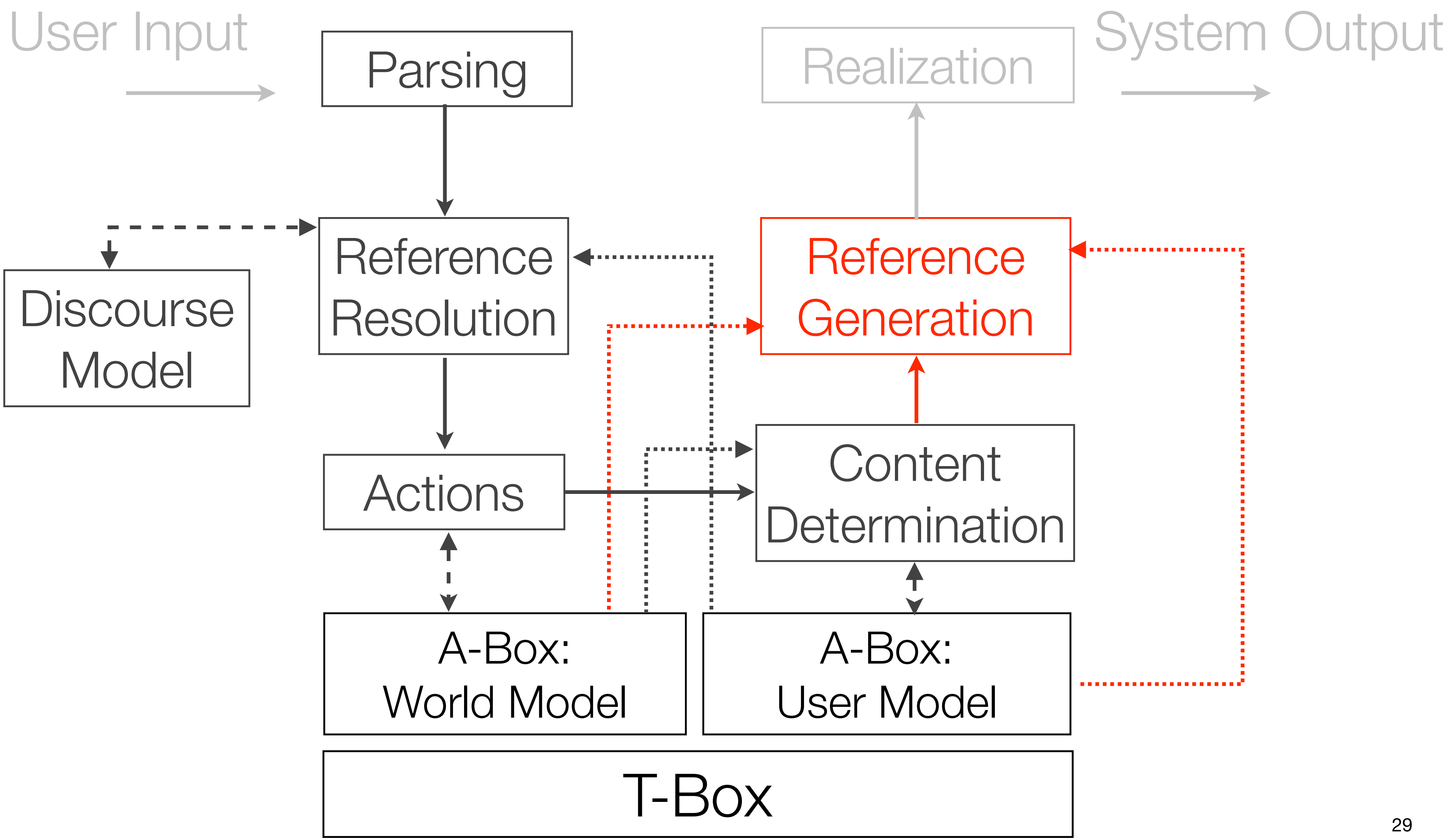
- in a description output, every concept or role of x gets one sentence

- store this in a list of sentences (each of them has a ‚goal' of the content, which is a variable to refer to the sentence's content):

describe the apple!

```
[content(goal: l1
         sem : [l1#apple(a2) green(a2)] )
 content(goal: l2
         sem : [l2#has-location(a2, ki)] )
 content(goal: l3
         sem : [l3#has-detail(a2, w1)] )
]
```

Generating Referring Expressions



Generating Referring Expressions

- individual variables (x, a2 w1,...) are meaningful to the system, but not to the user; NPs for the output have to be generated
- if an individual is not in the user's A-Box, simply generate an indefinite NP with the concept and perhaps the color:
  the bowl contains a red apple
- objects known to the user need unique definite descriptions:
  - query RACER for the individual (e.g. *a2*) and all individuals of the same concept (e.g. all apples)
  - add as many modifiers as necessary to describe the object uniquely (e.g. *the green apple*, if the other apples are red)

Generating Referring Expressions (cont.)

- add information necessary for the referring expressions to the content of a sentence:

(w1 is a worm, a2 is the apple which shall be described)

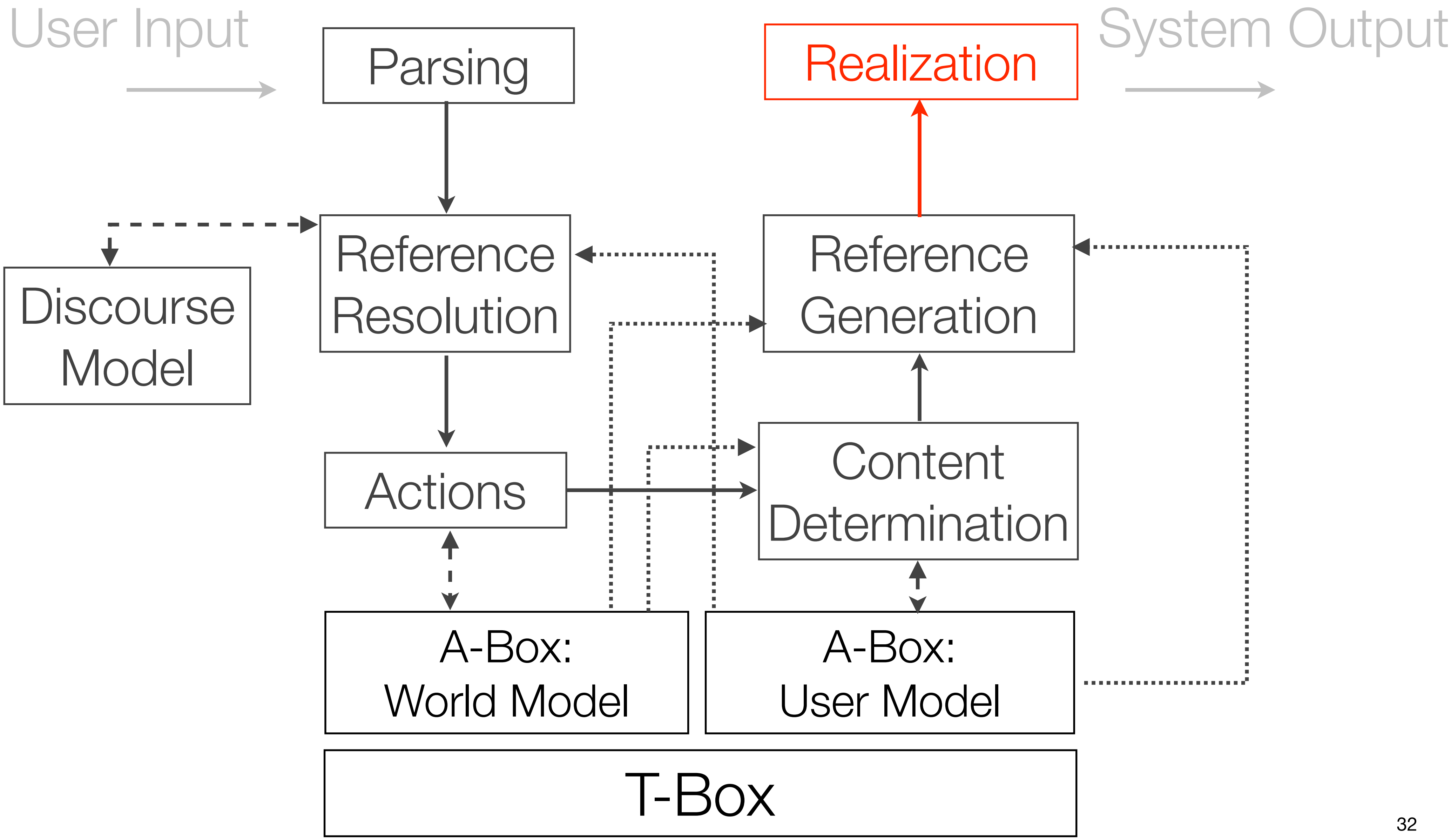
```
[content(goal: l3
         sem : [l3#has-detail (a2, w1)]  )  ]
```

```
[content(goal: l3
         sem : [l3#has-detail (a2, w1),
                indef(w1), worm(w1),
                def(a2), apple(a2), green(a2)]  ) ]
```

a worm

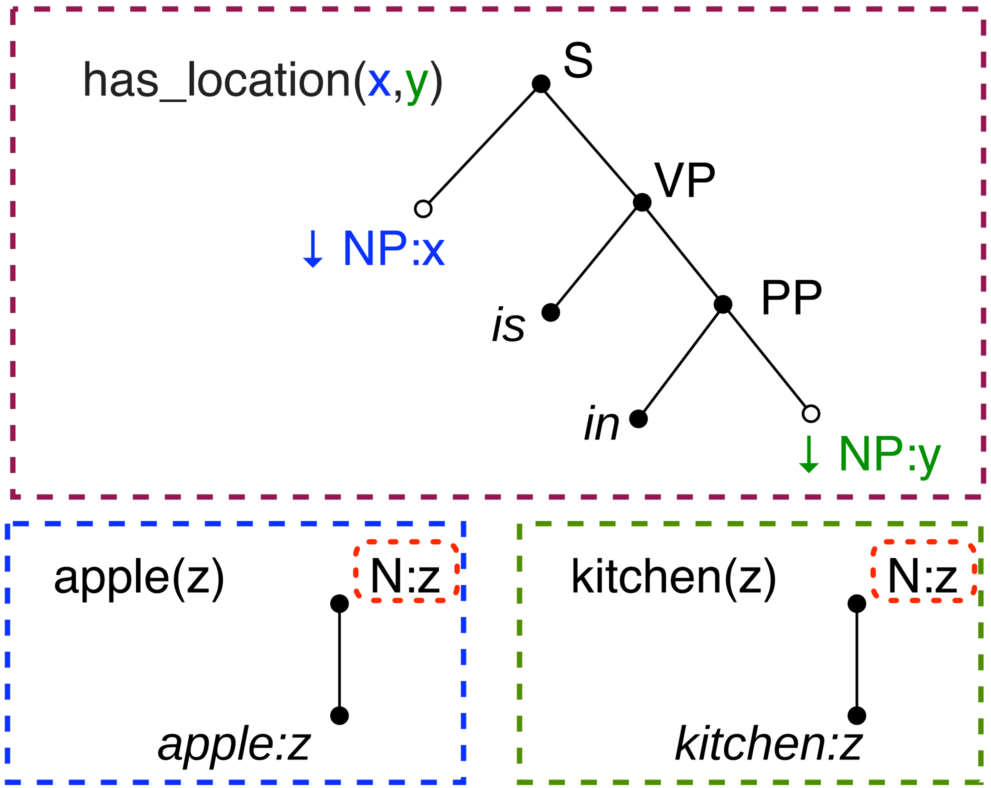
the green apple

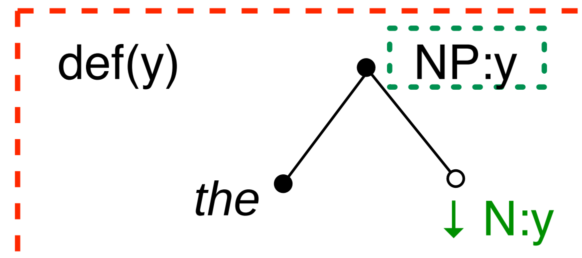
Surface Realization



Surface Realization

- A variant of Tree Adjoining Grammars as generation grammar; lexical entries associated with semantics:





Generator Input:

```
sem :
[l2#has-location(a2, ki),
def(a2), apple(a2),
def(ki), kitchen(ki)]
```

Performance

- (surprisingly) fast: Most user inputs are answered within 10 ms (upper bound: 500ms; only a few slower than 100ms)
- parsing and generation performs well for the given grammars
- RACER allows for fluent game playing; to fasten the game engine, A-BOX reasoning in RACER has been optimized further
- restrictions in DL (compared to FOL) don‘t harm the game (closed world, no actions in the knowledge base, user can only refer to what he knows)

Summary & Conclusion

- Description Logic as a decidable fragment of FOL for which fast reasoners (RACER) exit
- A computer game which employs CL-techniques and RACER in various stages of linguistic analysis and generation
- Limitations of DL don‘t harm the „closed world“ of the computer game, and RACER‘s efficiency makes the game playable
- There might be other NLP applications which can make use of the DL framework and its efficiency

References

- A. Koller, R. Debusmann, M. Gabsdil, and K. Striegnitz: Put my galakmid coin into the dispenser and kick it: Computational Linguistics and Theorem Proving in a Computer Game. Journal of Logic, Language, and Information, 2004.

- Ian Horrocks and Ulrike Sattler: Tutorial on description logics.
  Slides: http://www.cs.man.ac.uk/~horrocks/Slides/IJCAR-tutorial/Display/

- V. Haarslev and R. Möller. RACER System Description. In Proceedings of IJCAR-01, 2001.

- Alexander Koller and Kristina Striegnitz: Generation as dependency parsing. In Proceedings of ACL-02, Philadelphia, 2002.