Automated Negotiation from Declarative Contract Descriptions

Daniel M. Reeves

Michael P. Wellman

University of Michigan

Benjamin N. Grosof

MIT Sloan School of Mgmt

Hoi Y. Chan

IBM TJ Watson Research

2000 July 31

Designing a Negotiation Mechanism

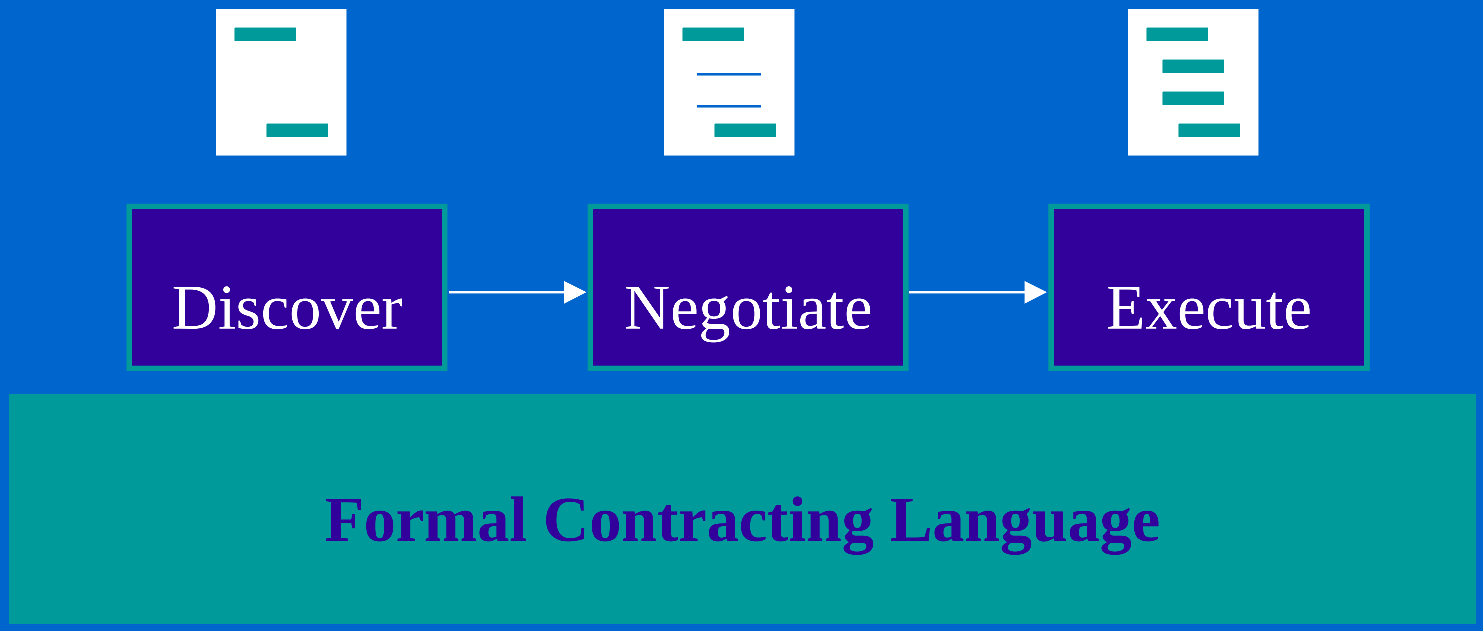
- Example: FCC spectrum auctions
- Alternative structures
  - Independent auctions for frequencies
  - Combinatorial mechanism
  - Simultaneous ascending auctions
- How (and why) to automate the construction of a negotiation mechanism…

Contracts

- Descriptions of goods and services
- Applicable terms and conditions
  - ancillary agreements detailing terms of a deal
  - customer service agreements, delivery schedules, conditions for returns, usage restrictions, other issues…
- *Partial Contracts* extend this
  - Intuitively: contracts with “blanks” to be filled in
  - More formally: defines space of possible negotiation outcomes

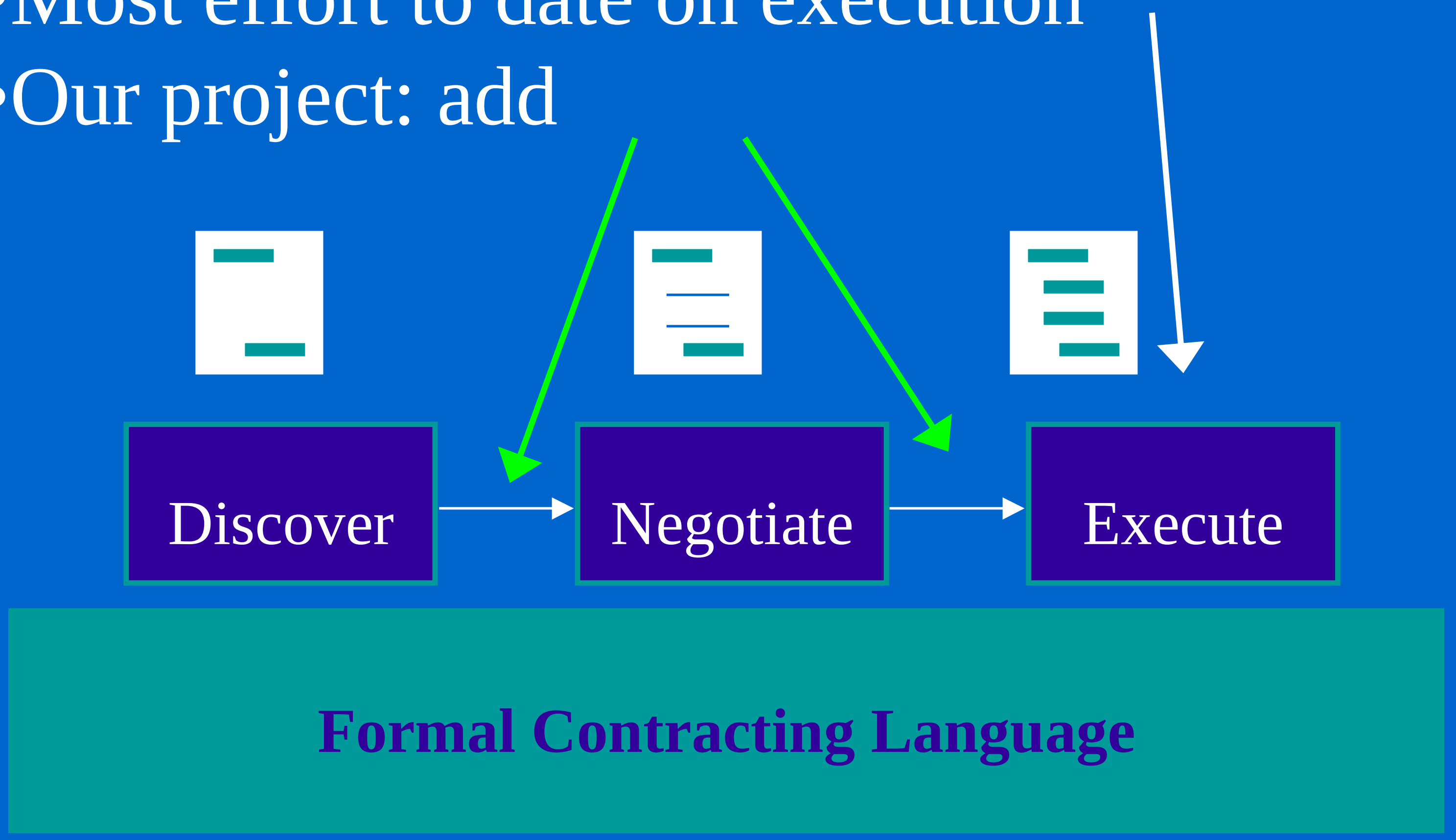
Contracting Infrastructure

- Contracting language supports all 3 stages of ecommerce
- Contract progressively more complete



Automated Contracting

- Most effort to date on execution
- Our project: add



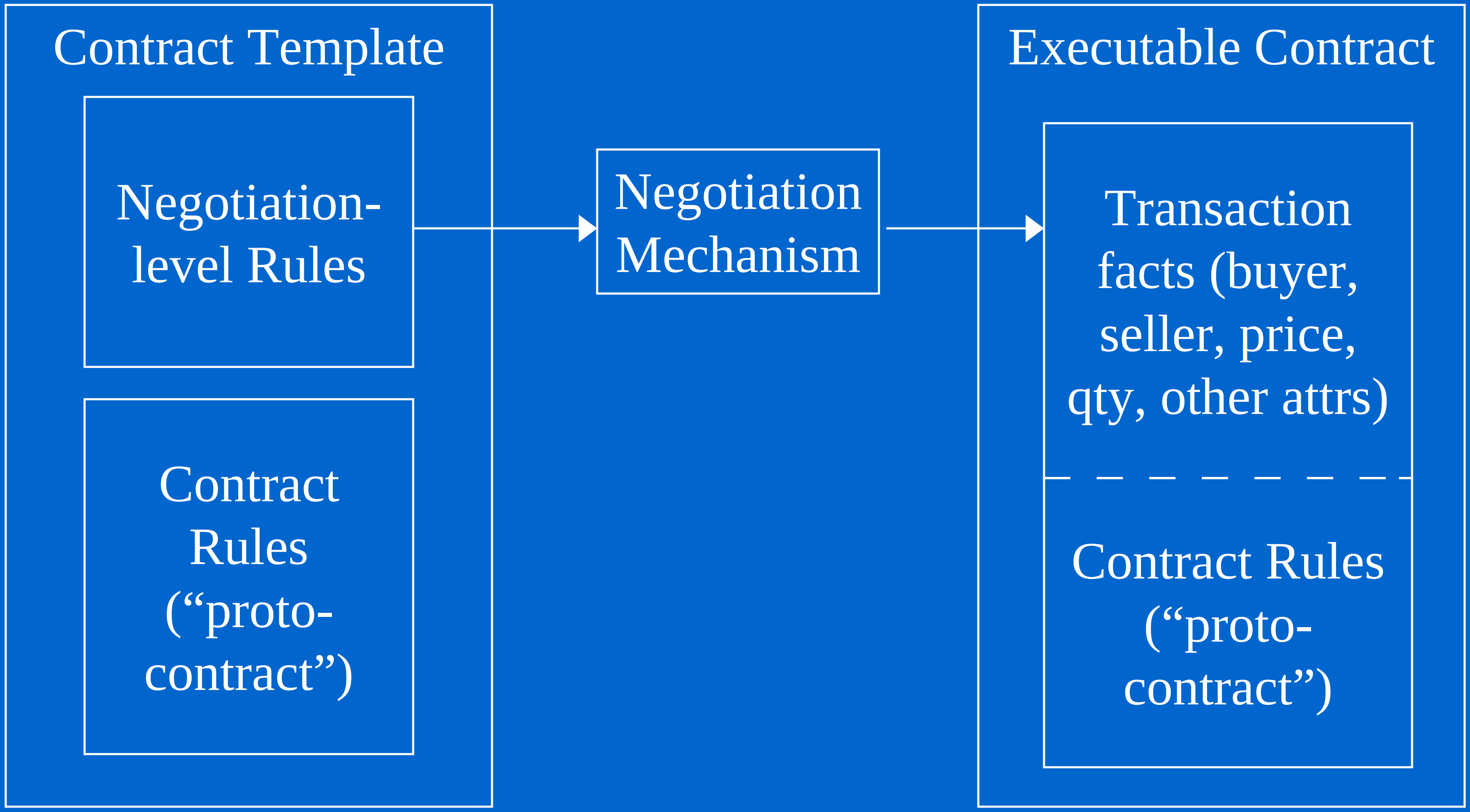
Formulating a Negotiation

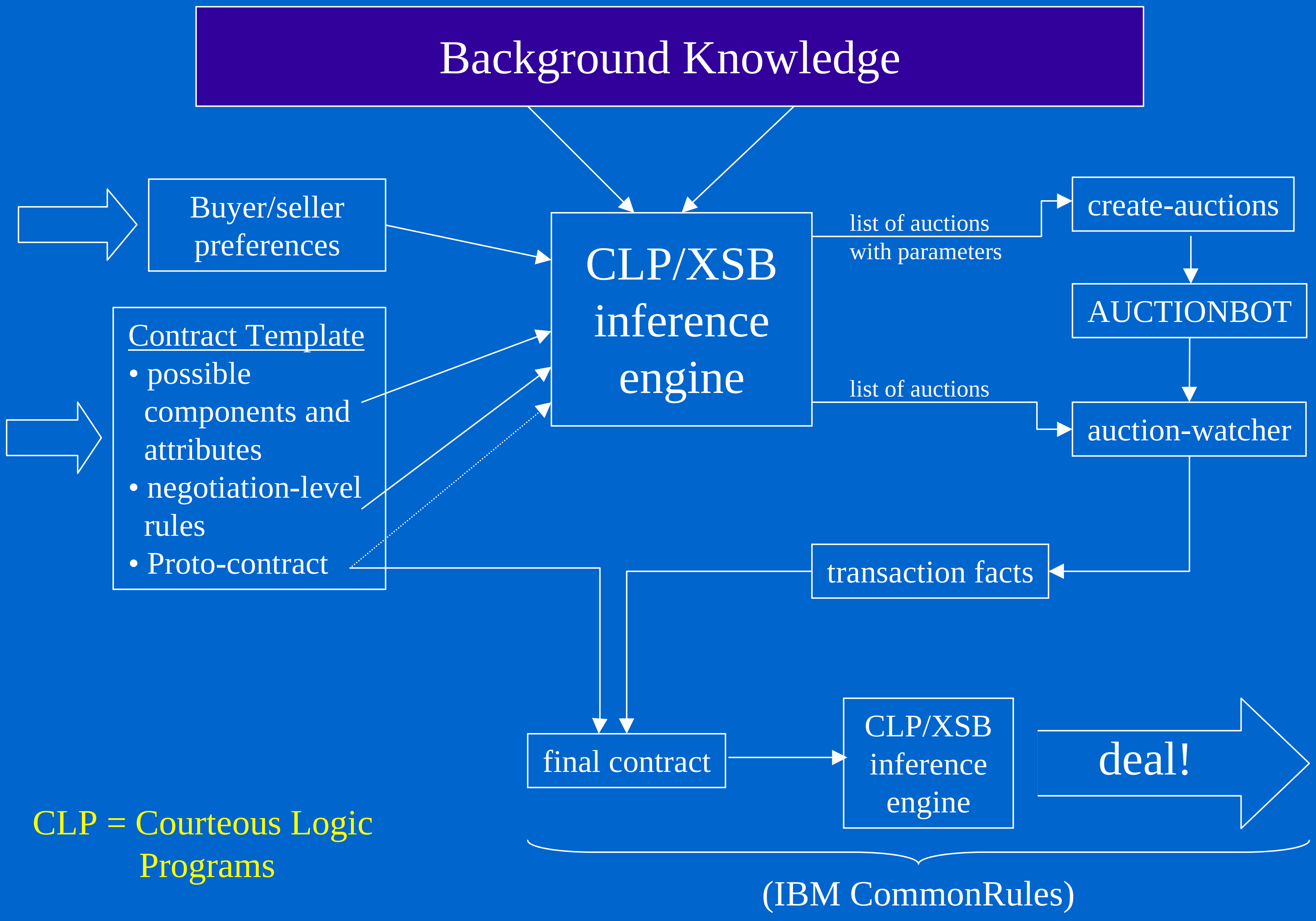
- *What* to negotiate
  - **Price** of an otherwise fully specified contract
  - **Everything** of a completely empty contract
  - *Something in between*…
- Negotiable parameters
  - Contracting issues to be determined in negotiation process
  - Flexibility/complexity tradeoffs

Formulating a Negotiation: Criteria

- Coherence/feasibility (e.g., size & color inseparable)
- Communication requirements
- Computational efficiency
- Allocation efficiency
- Examples:
  - ability to bundle (seller)
  - complementarities (buyer)
  - fewer/simpler markets (auctioneer)

Overall Process (Partial → Complete)


Contract Template
Negotiation-level Rules
Contract Rules (“proto-contract”)
Negotiation Mechanism
Executable Contract
Transaction facts (buyer, seller, price, qty, other attrs)
Contract Rules (“proto-contract”)



CLP = Courteous Logic Programs

Auction-Configuration Rulebase

- Partition negotiation into a set of components--separable bundle of goods
- Combine constraints between possible components, buyer/seller preferences
- Infer components to be negotiated
- Create arrays of 1-D auctions
- Priorities and mutual exclusion rules
  - E.g., only infer one value for each auction parameter

Auction-Space Rulebase: Domains, Defaults, and Constraints

- Domains for auction parameters
- Default values for all auction params (lowest priority rules)
- Conditional defaults (next lowest priority)
  - 1 seller implies multiple buyers and vice versa
- Hard Constraints

```
– <highest> auction(?ID, beatQuote, 0) <-
            auction(?ID, meetQuote, 1).
```

Improved Parameterization

- Current parameterization in AuctionBot created incrementally and slow to change due to backward-compatibility constraints
- Independent of actual AuctionBot parameters
- Provides more *flexible* and *extensible* structure than a flat and complicated parameter space

Higher-Level Knowledge

- Infer auction parameters from `negotiationType` facts
- `negotiationType` used in partial contract for meta-level knowledge about negotiation
- Example:

`negotiationType(continuous)` implies
`negotiationType(continuousQuotes)` implies
`auction(quoteMode, bid)`

Standard Auction Types

- Encodes well-known auction types
  - negotiationType(CDA) => negotiationType(double), etc
  - Uses exceptions and special cases
    - Example: Amazon-style auctions are like eBay *except* that there is no fixed final clear time.

Additional Benefits

- Advantage of rule-based approach: adding new structure to parameterization
- Example: inferring default parameter settings based on user profiles (business, consumer, skilled, novice, etc.)
- Succinct: AuctionBot requires 27 individual parameter settings, as opposed to a handful of rules for most auctions

AuctionBot Rulebase

- Maps Auction-Space parameters to AuctionBot parameters
- Example: “auction type” inferred from fundamental auction parameters

```
<chronmatch> auctionbot(type, 4)
  <- auction(matchingfunction, earliesttime).
<cda> auctionbot(type, 5)
  <- auction(matchingfunction, earliesttime)
     AND auction(intclearmode, 1).

overrides(cda, chronmatch).   /* special case */
```

Domain-specific Rules (Components, Attributes, Values)

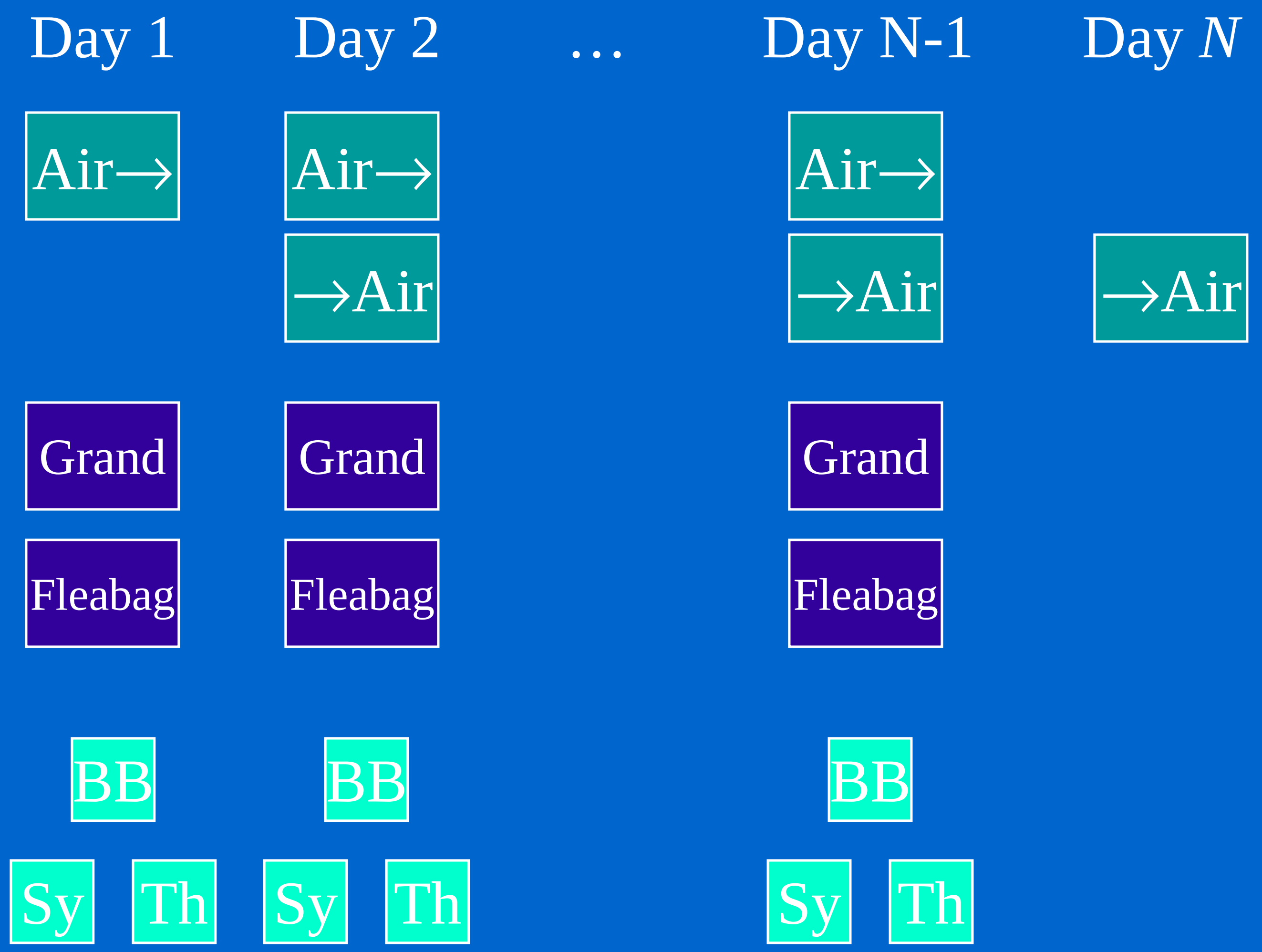
- Possible values
  - `value(quality, regular).`
  - `value(quality, deluxe).`
- Possible components and attributes
  - `component(widget).`
  - `attribute(widget, quality).`
- Possible values for widgets

```
value(?Component, quality, ?Q) <-
   component(?Component) AND
   value(quality, ?Q).
```

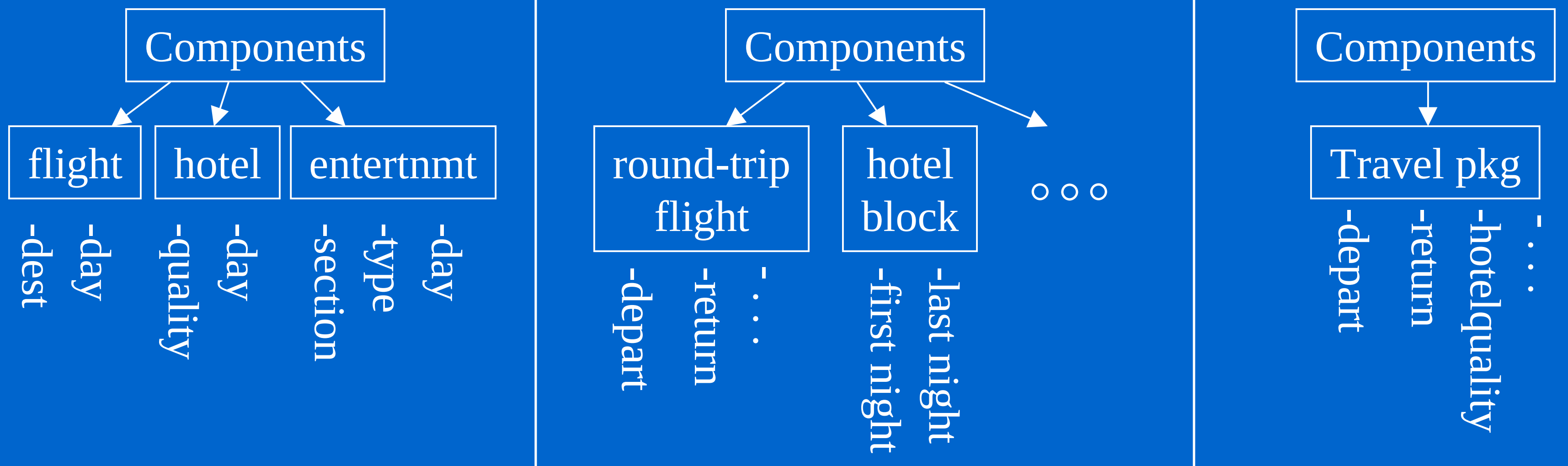
Trading Agent Example

- Generates all auctions for TAC
- 3 goods (flights, hotels, entertainment)
  - Each has attributes for day and for type
    - (2 flight types, in & out; 2 hotel types, good & bad; 3 entertainment types, baseball & symphony & theatre)
- Total auctions created per good:
  - [types]*[days]
- Negotiation-level rules included

Goods in TAC Domain



Alternative Negotiation Structures

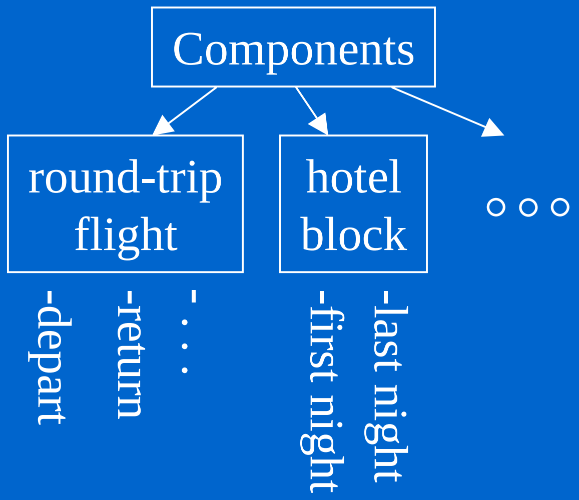


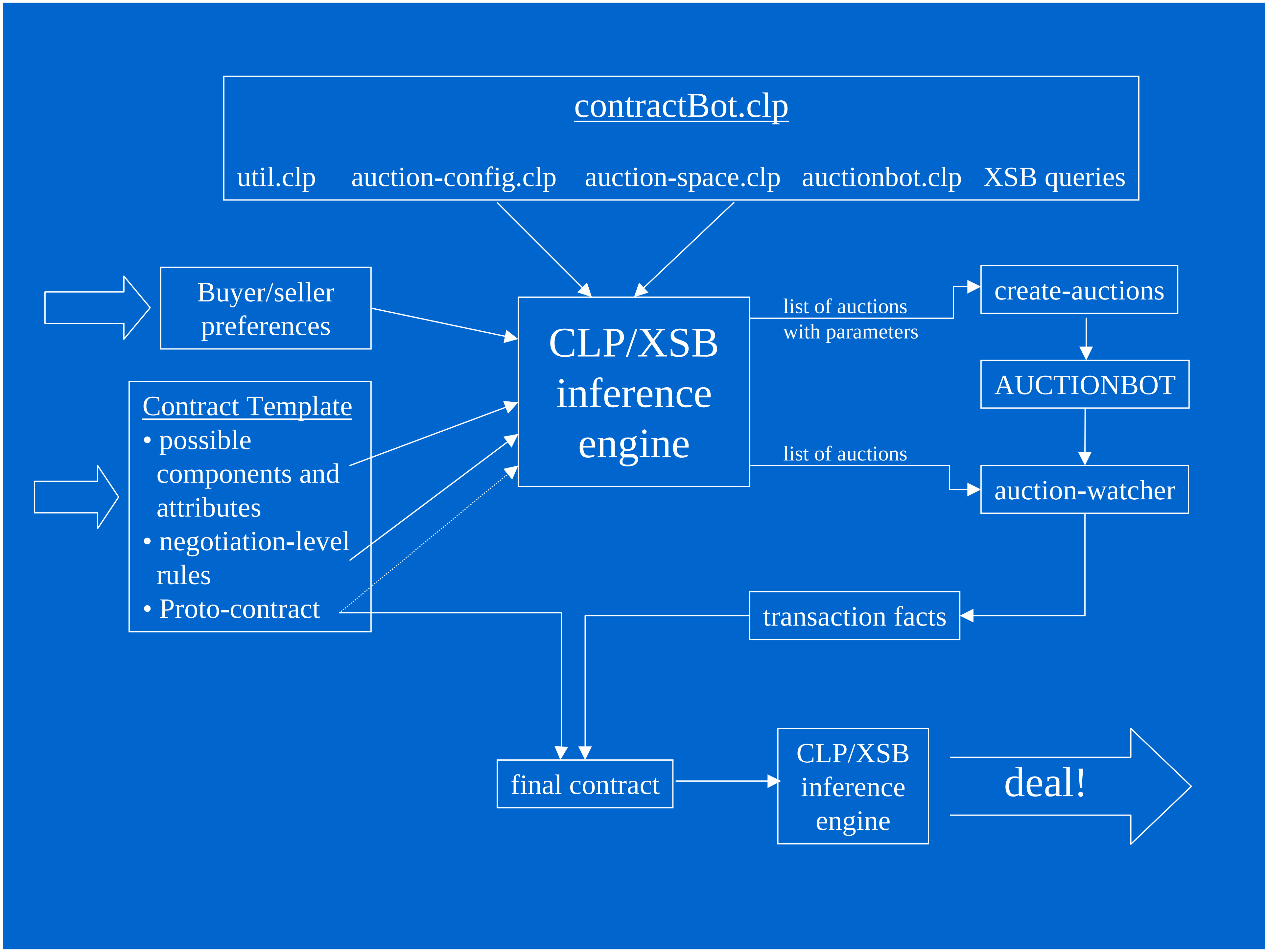
Agent Example

- Possible components: hotelblock, roundflight, flighthotel, entpackage, fullpackage, etc
  - (components may inherit features from each other)
- Constraints between components, e.g., buyers want hotelblocks xor individual rooms
- Buyer/seller preferences, e.g.:
  - `buyer(traveler2, hotelblock).`
  - `seller(airline1, roundflight).`

Simple Buyer/Seller Rules for Alternative Negotiation Structure

```
buyer(traveler1, roundflight).
buyer(traveler2, hotelblock).
buyer(traveler1, entpackage).
seller(airline1, roundflight).
seller(hotel1, hotelblock).
seller(agent3, entpackage).
```




contractBot.clp
util.clp
auction-config.clp
auction-space.clp
auctionbot.clp
XSB queries
Buyer/seller preferences
Contract Template
• possible components and attributes
• negotiation-level rules
• Proto-contract
CLP/XSB inference engine
list of auctions with parameters
list of auctions
create-auctions
AUCTIONBOT
auction-watcher
transaction facts
final contract
CLP/XSB inference engine
deal!

Summary

- Contracting language as infrastructure for automated contracting
- Contracting framework
  – Partial to complete contracts
- Rule-based auction generation/configuration
- Alternative negotiation structures for TAC
- ContractBot prototype

Future Work

- Support for richer negotiation mechanisms (e.g., combinatorial and multiattribute auctions)
- Extend ontology (e.g., orthogonality/separability)
- Analyze agent strategies for submitting rules influencing the choice of negotiation mechanism

BACKUP SLIDES

Prototype details

Auction Constraints (additional details)

- Rules about rule priorities
  - 4-5 levels of priority useful in this application for expressing defaults, exceptions, overrides
    - low, medium, high/very-high
    - also: “standard “ (no label)
- Mutual exclusion (similar to integrity constraint):
  - at most one value for each auction param

Creating a Batch of Auctions

```
require “auctionGenerator.pl”;  # simple Perl library

for($i = 1; $i <= $ARGV[1]; $i ++) {
  beginAuction();
  addRule(“negotiationType(cda).”);
  addRule(“negotiationType(revealAll).”);
    # could also have the rules in a file and use:
    #   addFile(“filename.clp”);
  addParam(“auctionname”, “auction$i”);  # uses override priority
  endAuction();
}
```

BACKUP SLIDES

Detailed ontology and configuration criteria

Configuration Criteria

- Feasibility/coherence
    - Will it result in valid/sensible contracts?
- Expected performance
    - Will it lead to desirable outcomes?
        - Pareto efficiency
        - Other measures of social utility
- Complexity
    - How costly, for both operators and participants?

Configuration Criteria: Complexity

- Agent complexity
  - Incentive compatibility
  - Bid format, iterations
- Computational complexity of mechanism
  - E.g., time complexity in number of agents/attributes
- Communication costs
- “Cognitive” complexity

Configuring the Mechanism (exploiting information from the contract template)

- Attribute hierarchy
- Orthogonality (w.r.t. siblings in hierarchy)
  - additive utility
  - vastly reduces search (eg, 10^4 vs. 2*10^2)
- Separability
  - suggests combinatorial mechanisms
  - can be reasoned about (example)

Hints from Contract Template, continued

- Privacy (example)
- “Negotiability” of attributes
  - E.g., seller/buyer chooses
- Constraints
  - Declarative language well-suited
  - E.g., ~hotel <- ~flight

Questions and Future Work [NWU]

- Parameterize the space of negotiation mechanisms
- Other hints from the partial-contract language for configuring the negotiation
  - Reducing search costs
  - Meta-level hints/specifications
  - Other information influencing design choices

BACKUP SLIDES

Negotiation-level predicates and examples of making aspects of an executable contract negotiable

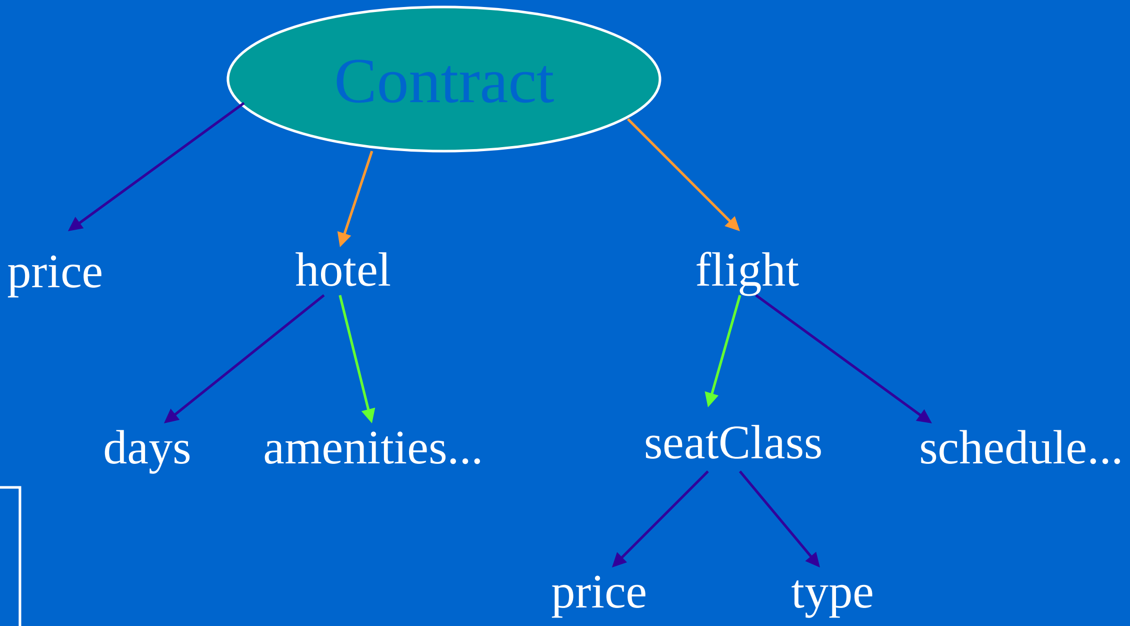
Negotiation-level Predicates:

attribute, separableComponent, orthogonalComponent

attribute(?Parent, ?Child).

separableComponent(hotel)…

orthogonalComponent(seatClass)...



price, quantity: *distinguished*

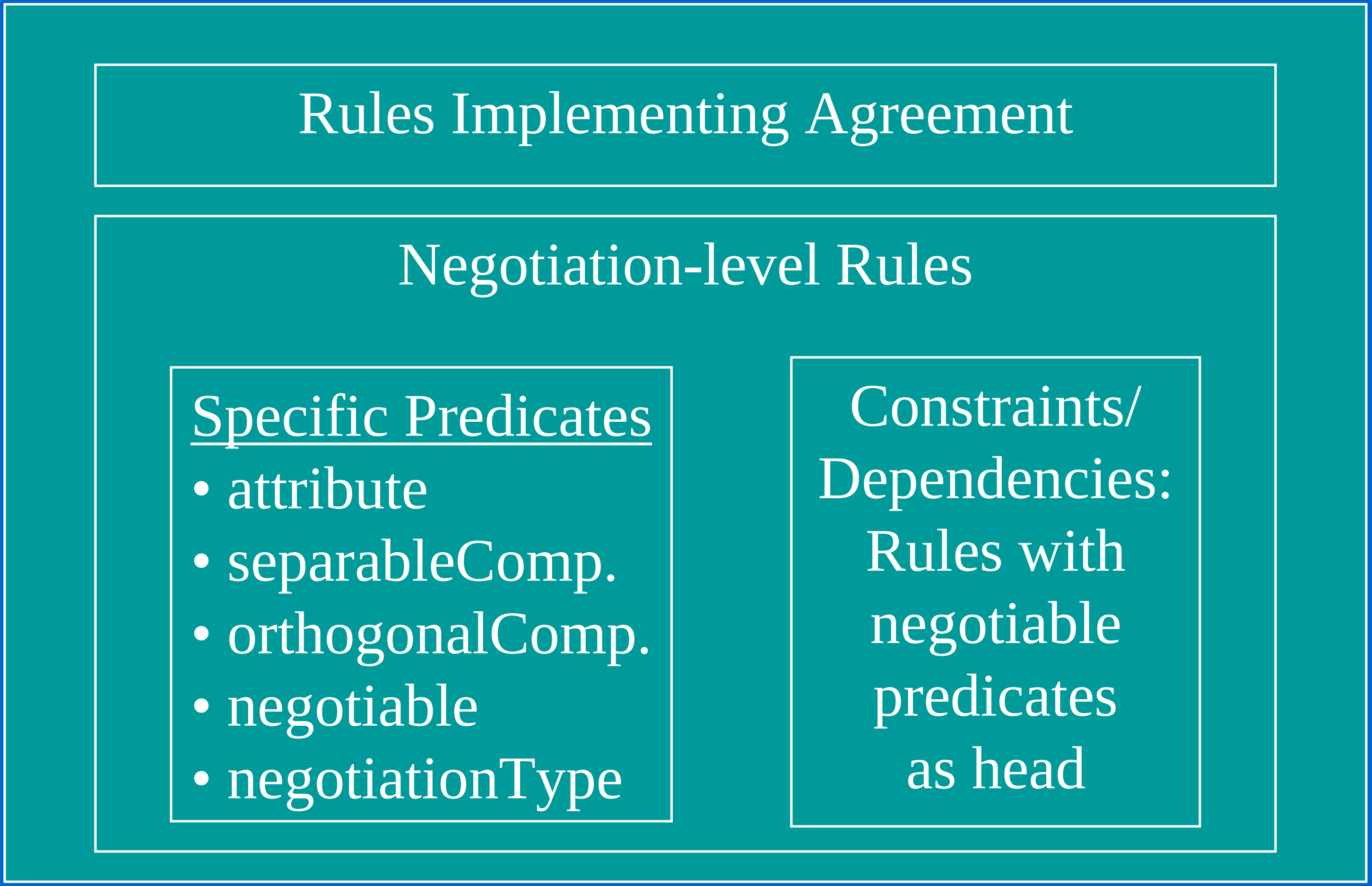
Negotiation-level Predicates: **negotiable**

```
negotiable(?PredicateName).

flight(?Airline, ?FromCity, ?ToCity, ?Stopovers) ←
    airline(?Airline) AND stopovers(?Stopovers) AND
    possibleRoute(?Airline, ?FromCity, ?ToCity).

negotiable(’airline).
negotiable(’stopovers).
```

Negotiation-level Predicates: **negotiationType**

```
negotiationType(?PredicateName, ?TypeOfNegotiation).

  negotiable(’hotelCost).
  negotiationType(’hotelCost, sellerChooses).
```

Composition of Contract Template (a.k.a., Partial Contract)



Adding Negotiation Constructs to Existing Contracts

- Negotiating the form of a rule

ruleHead ← ruleBody AND isRuleIncluded(yes).

negotiable(’isRuleIncluded).

$$\begin{tikzcd}
& \boxed{\text{Negotiation Mechanism}} \arrow[dl] \arrow[dr] & \\
\text{isRuleIncluded(yes).} & & \text{isRuleIncluded(no).}
\end{tikzcd}$$

Adding Negotiation Constructs to Existing Contracts (continued)

- Making constants negotiable

```
foo(constant1, constant2) ← conditions.

            becomes

foo(?Var1, ?Var2) ← conditions AND
   var1(?Var1) AND var2(?Var2).
negotiable('var1).
negotiable('var2).
```

BACKUP SLIDES

CLP details and general "rules motivation"

Why Rules?

- Contract terms involve conditional relationships
  - Terms and conditions, e.g., rules for price discounting
  - Service provisions, e.g., rules for refunds
  - Surrounding business processes, e.g., rules for lead time to place an order
- Shared semantics
- Existing executable contracts can be easily parameterized without a meta-language

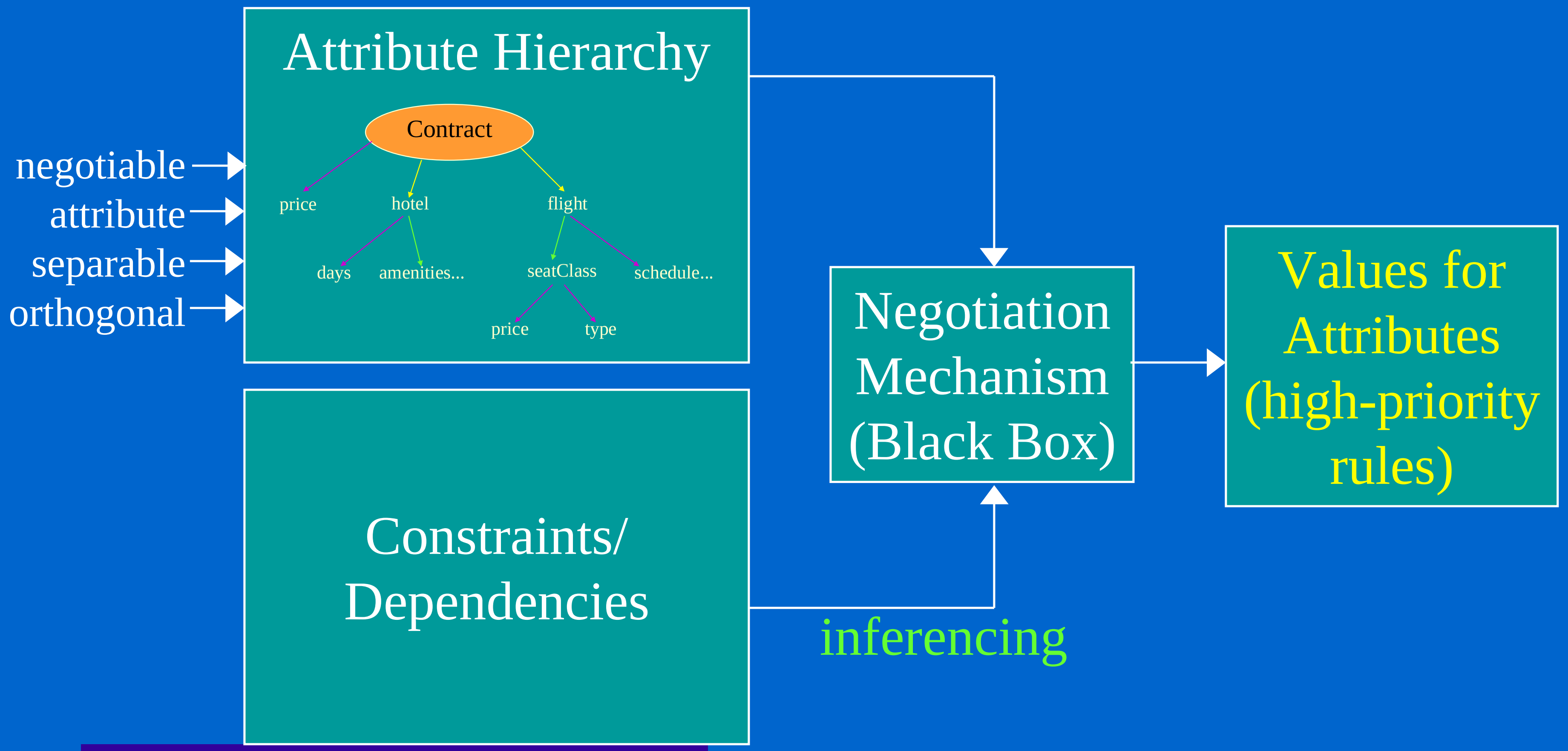
Courteous LPs: Advantages

- Facilitate updating and merging.
- Expressive: classical negation, partially-ordered prioritization, reasoning to infer prioritization.
- Set of conclusions guaranteed consistent, unique.
  - **Mutual exclusion is enforced**. E.g., never conclude both p & ¬p.
- Efficient: low computational overhead beyond ordinary LPs.
  - Tractable given reasonable restrictions (Datalog, max vars/rule)
  - Extra cost is equivalent to increasing v to (v+1) in ordinary LPs.

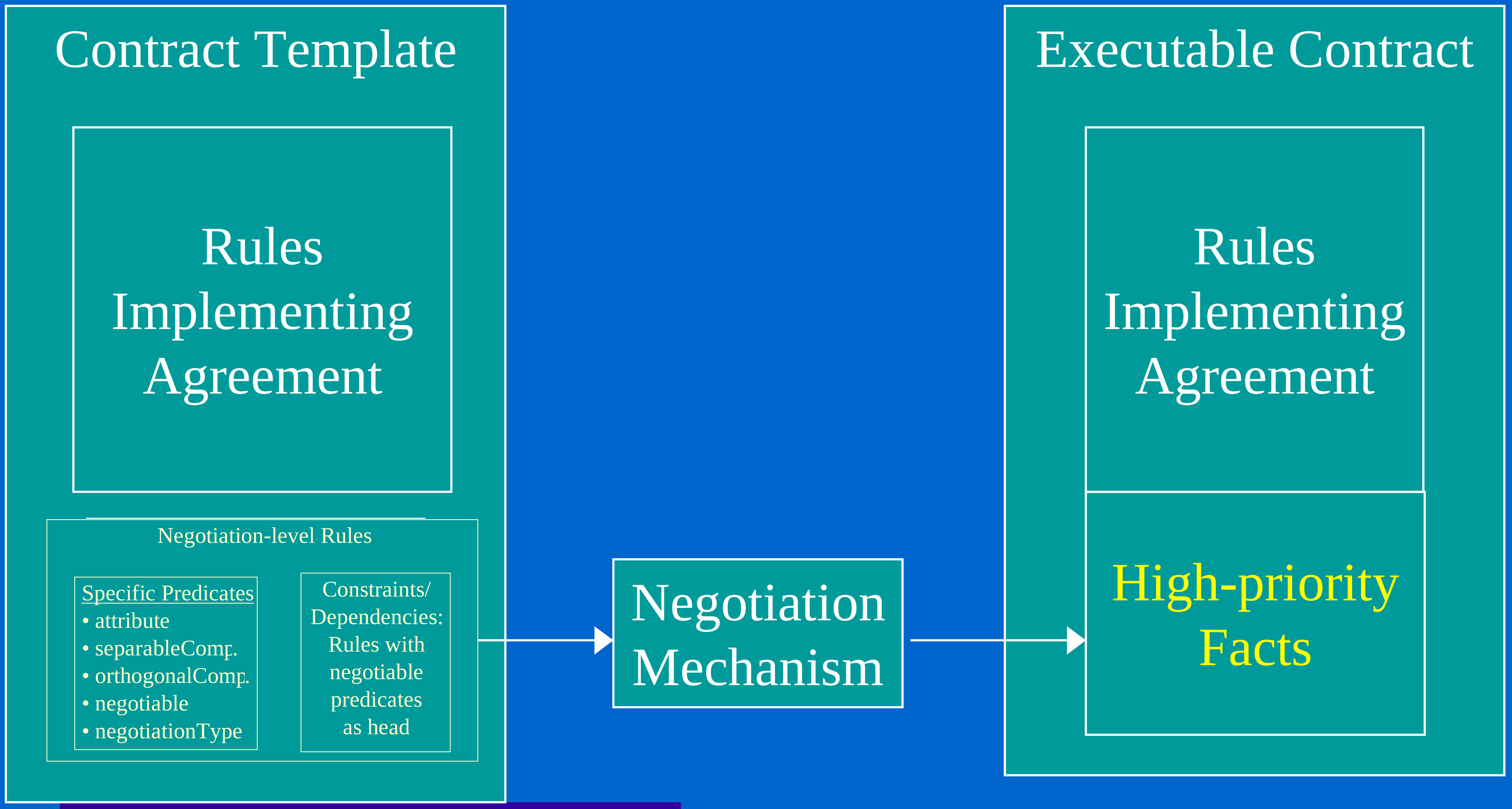
BACKUP SLIDES

Detailed process (partial to complete)

Negotiation Process



Overall Process (Partial → Complete)



BACKUP SLIDES

Miscellaneous

Multidimensional Auctions

- *How* to negotiate
  - Auctions: mediated, well-defined, market-based
  - Multidimensional: resolve multiple issues
- Types
  - Multiple single-dimensional
  - Combinatorial
  - Multiattribute

Size of Possible-Outcome Space

- Too big!
- Contract language can help
  - Declarative language well-suited for expressing constraints
  - Orthogonalities (eg, 10^4 vs. 2*10^2)
  - Negotiation types (eg, sellerChooses)

Definition of Negotiation

- Negotiation = establishing a contract
- Example: Auction
  - Description of good with blanks for price/quantity
- Example: Negotiating ecommerce transactions
  - Results in an executable piece of code that executes the transaction

BACKUP SLIDES

Intro/Motivation, General Contracting

Definition of a Contract

- Contract = Set of attributes and values
- Partial Contract: values unspecified
- Includes seemingly structural aspects
  - Example: whether to include a rule in a declarative contract

Formal Contracting Language

- Partial vs. Complete Contracts (Contract Templates vs. Executable Contracts)
- Unspecified terms reduced to “negotiable parameters”
  - Attributes with specified domains
- Contract language expresses both partial and complete contracts
  - Additional ontology for parameterizing the negotiable aspects

Negotiation from Contract Templates

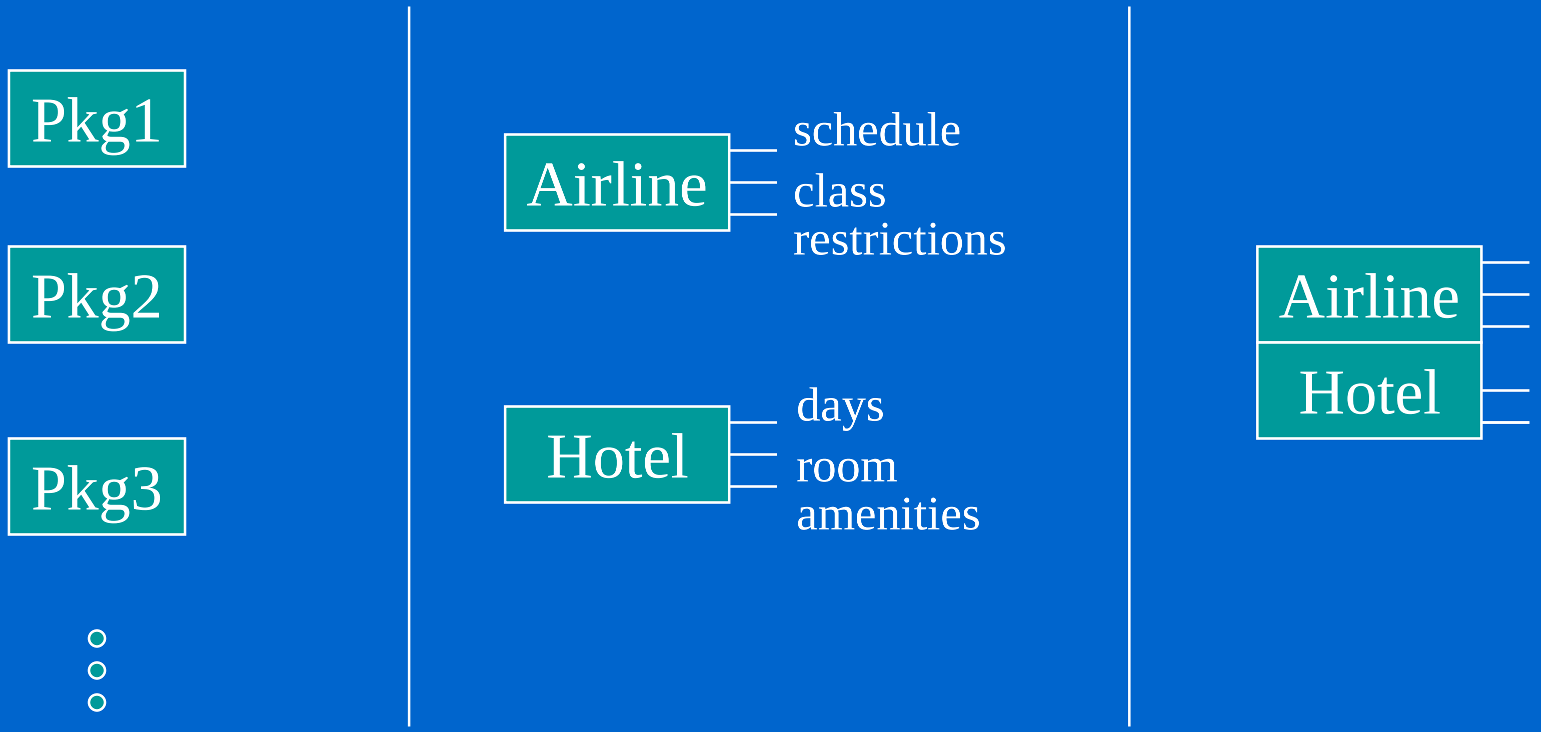
- Structural aspects can be parameterized
  - Rule-based language allows boolean attributes to effectively include or omit clauses/terms from the contract
- Negotiation reduces to assignment of values to attributes

BACKUP SLIDES

Old stuff

Alternative Negotiation Structures: Travel Packages Example

TODO: subset of possible TAC components



…and combinations thereof.

Summary [NWU]

- Contracting language as infrastructure for automated contracting
- Partial contract reduces negotiation to attribute assignment
- Alternative negotiation structures and criteria for mechanisms
- How the contracting language can guide the configuration of mechanisms and improve their efficiency

Discussion and Future Work

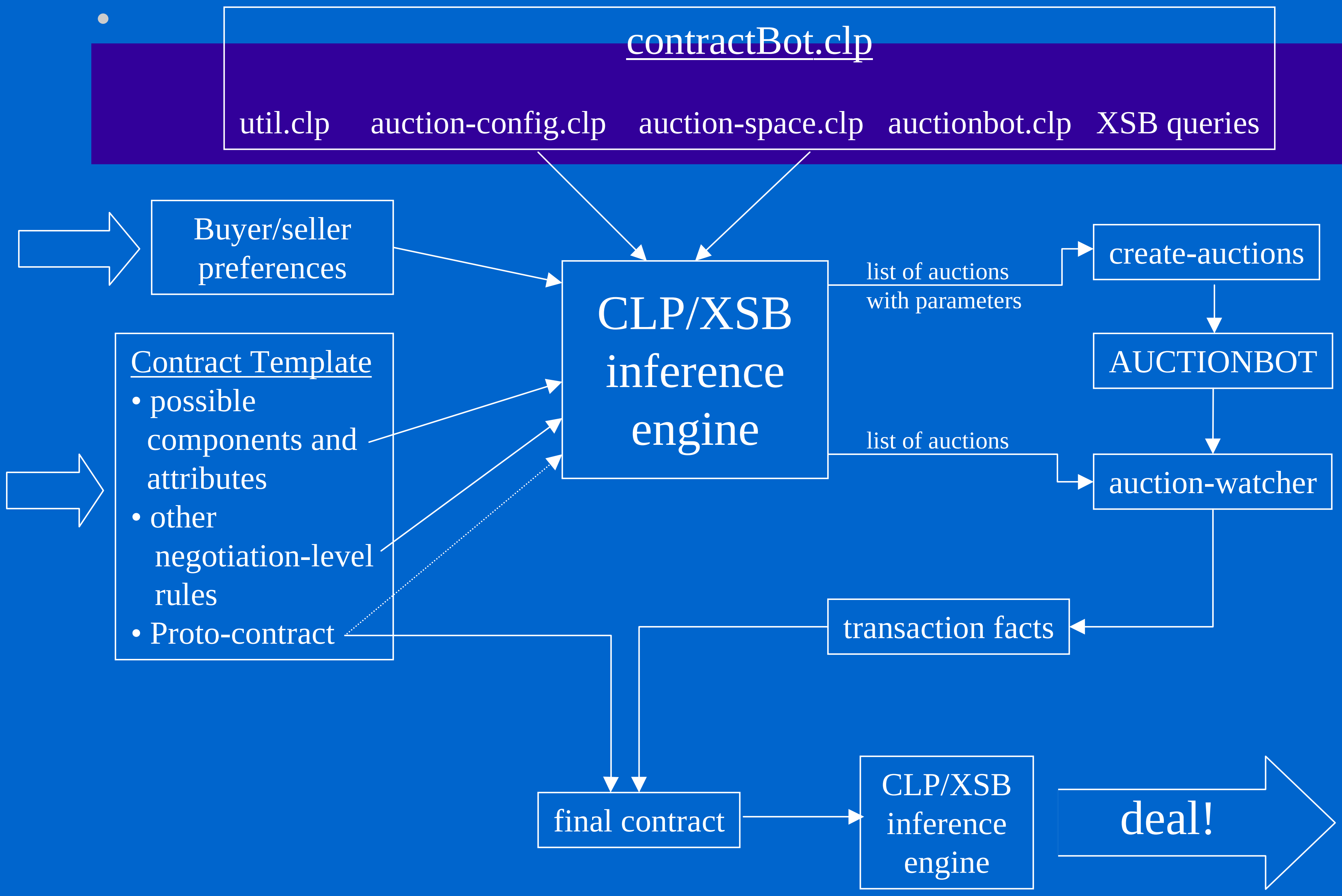
- Iterative negotiation
- Situated Courteous Logic Programs:
  - procedural attachments for actions, queries
- XML as common interlingua
- Multidimensional negotiation mechanisms

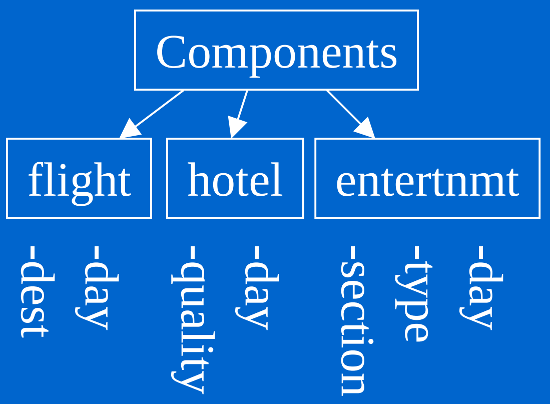
Summary [AIEC]

- CLP as basis for executable contracts
- CLP's prioritized conflict handling also facilitates modification during negotiation
- Introduced specific predicates for negotiation
  - Hierarchy of negotiables
  - Reason about what is negotiable and how
- Constraints and dependencies handled naturally
- Demonstrated how negotiation mechanism can transform a contract template to a fully executable contract
- Showed how to make an existing contract negotiable

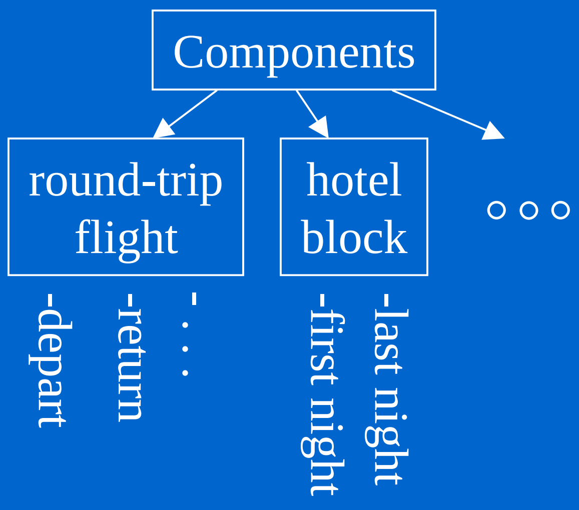
SCRATCH

Figures for paper and other miscellania

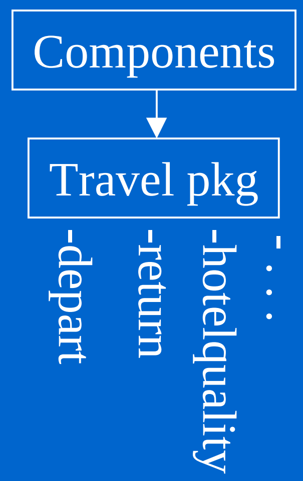

contractBot.clp
util.clp
auction-config.clp
auction-space.clp
auctionbot.clp
XSB queries
Buyer/seller preferences
Contract Template
• possible components and attributes
• other negotiation-level rules
• Proto-contract
CLP/XSB inference engine
list of auctions with parameters
list of auctions
create-auctions
AUCTIONBOT
auction-watcher
transaction facts
final contract
CLP/XSB inference engine
deal!


Components
flight
hotel
entertnmt
-dest
-day
-quality
-day
-section
-type
-day

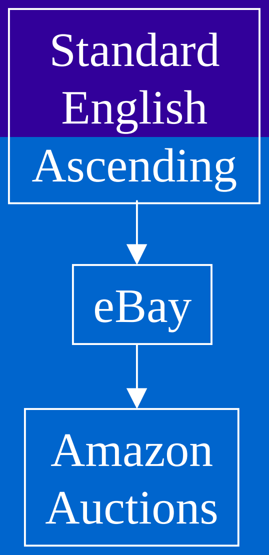
(a)


Components
round-trip flight
hotel block
-depart
-return
- . . .
-first night
-last night

(b)


Components
Travel pkg
-depart
-return
-hotelquality
- . . .

(c)


Standard English Ascending
eBay
Amazon Auctions

Rule-based Contracts for E-commerce

- Rules as way to specify business processes as part of contract terms.
- Facilitates specification
  - by multiple authors, cross-enterprise, cross-application
  - by non-technical authors
  - dynamically
- Existing executable contracts can be easily parameterized without a meta-language

Courteous Logic Programming

- Generalization of Logic Programming to include **prioritized conflict handling.**
- Rules may override other rules
  - special cases / exceptions / defaults
  - more recent updates
  - higher-authority (and/or more reliable) sources
  - closed world: lowest priority for catch cases

Example of Conflicting Rules

Vendor’s rules that prescribe how buyer must place or modify an order:

A) 14 days ahead if

- buyer is a preferred customer

B) 2 days ahead if

- the modification is to reduce the quantity, and
- the item is in backlog at the seller.

Resolved by **precedence** between the rules.

Often only *partial* order of precedence is justified.

Example of Conflicting Rules in CLP

```
<leadTimeRule1>
orderModificationNotice(?Buyer, ?Seller, ?Order, 14days) ←
    preferredCustomerOf(?Buyer, ?Seller).

<leadTimeRule2>
orderModificationNotice(?Buyer, ?Seller, ?Order, 2days) ←
    preferredCustomerOf(?Buyer, ?Seller) AND
    orderModificationType(?Order, reduce) AND
    orderItemIsInBacklog(?Order).

Overrides(leadTimeRule2, leadTimeRule1).
```

Configuring the Negotiation
Example: Orthogonality vs. Separability

	separable	NOT separable
orthogonal	Maps w/ car	Car color
NOT orthogonal	Flight (whether to get car depends on arrival city)	Car base price depends on time needed, etc

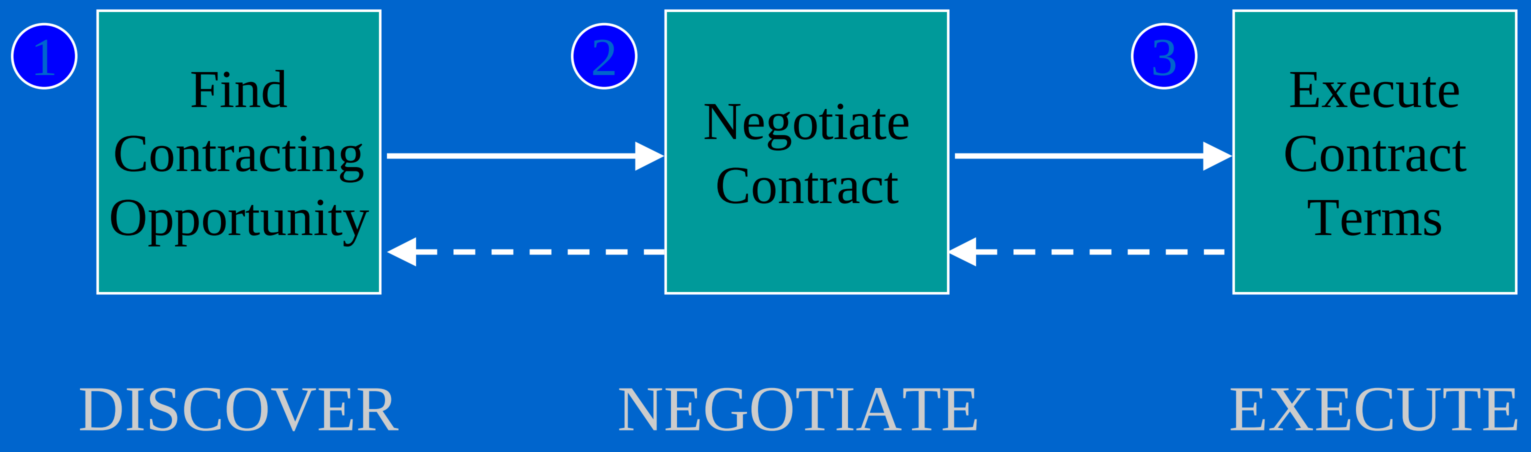
TODO (BillB)

- How does our approach scale?
  - Grosof complexity results
  - the straightforward way to express possible negotiation outcomes doesn’t scale… lots of things about our ontology address this
- How and what does the language make it easier to express?
- How have we exploited structure in a problem?
  - Component hierarchy, flight structure, etc

Other Uses for Prototype

- Generalization of Economy Generator
  - A few lines of CLP rules instead of 27 parameters
    - Rules of the form `auction(param,val)` also allowed, for specifying low-level parameters
  - Perl library for creating batches of auctions
- Support for reasoning about alternative negotiation structures

Contracting 1-2-3



Applies to any contracting, electronic or not.
May iterate or interleave these steps.
Boundaries not necessarily sharp.