A Controlled Language for the Specification of Contracts

Gordon Pace Michael Rosner

University of Malta

Outline

- Motivation
- Contracts
  - Language
  - Logic
- Current state of progress
- Issues

Objectives of the Work

- Natural Language Processing
  - Identification of what we mean by controlled language
  - Determination of a *particular purpose* for controlled language
  - Improving performance in one or more different areas of NLP (usually analysis, generation, semantics).
- Formal Verification
  - Provable correctness of programs
  - Showing that program behaviour is in conformity with specification
- Specification as contract
  - Identification of a *formal* language in which such contracts can be expressed
  - Implementation of such a language to enable various inference mechanisms to be defined.
  - Enabling reasoning about the specification itself.

Why Contract Language Might Make Good Controlled Language

- Language
  - The language of contracts is *potentially* a sublanguage.
  - Charactersitic terminology and syntactic constructs.
- Semantics
  - Meaning primarily concerns regulation of behaviour
  - Concerns a specific set of concepts (permission, prohibition etc).
  - Many different kinds of "application" can be imagined with respect to contracts (e.g. verification, explanation)
- But unfortunately not all contracts are expressed in controlled language.

Why Contracts are Not Automatically Controlled Languages

Groucho Marx: Now pay particular attention to this first clause, because it's most important. There's the party of the first part shall be known in this contract as the party of the first part. How do you like that, that's pretty neat eh?

Chico Marx: No, that's no good.

Groucho Marx: What's the matter with it?

Chico Marx: I don't know, let's hear it again.

Groucho Marx: All right. It says the first part of the party of the first part shall be known in this contract as the first part of the party of the first part, shall be known in this contract - look, why should we quarrel about a thing like this, we'll take it right out, eh?

Chico Marx: Yes, it's too long anyhow. Now what have we got left?

Moral

the controlled language of contracts needs to be very carefully delimited and is not just NL applied to contracts

Why Contracts are Not Automatically Controlled Languages

Groucho Marx: Now pay particular attention to this first clause, because it's most important. There's the party of the first part shall be known in this contract as the party of the first part. How do you like that, that's pretty neat eh?
Chico Marx: No, that's no good.
Groucho Marx: What's the matter with it?
Chico Marx: I don't know, let's hear it again.
Groucho Marx: All right. It says the first part of the party of the first part shall be known in this contract as the first part of the party of the first part, shall be known in this contract - look, why should we quarrel about a thing like this, we'll take it right out, eh?
Chico Marx: Yes, it's too long anyhow. Now what have we got left?

Moral

the controlled language of contracts needs to be very carefully delimited and is not just NL applied to contracts

Why Contracts are Not Automatically Controlled Languages

Groucho Marx: Now pay particular attention to this first clause, because it's most important. There's the party of the first part shall be known in this contract as the party of the first part. How do you like that, that's pretty neat eh?
Chico Marx: No, that's no good.
Groucho Marx: What's the matter with it?
Chico Marx: I don't know, let's hear it again.
Groucho Marx: All right. It says the first part of the party of the first part shall be known in this contract as the first part of the party of the first part, shall be known in this contract - look, why should we quarrel about a thing like this, we'll take it right out, eh?
Chico Marx: Yes, it's too long anyhow. Now what have we got left?

Moral

the controlled language of contracts needs to be very carefully delimited and is not just NL applied to contracts

Why Contracts are Not Automatically Controlled Languages

Groucho Marx: Now pay particular attention to this first clause, because it's most important. There's the party of the first part shall be known in this contract as the party of the first part. How do you like that, that's pretty neat eh?
Chico Marx: No, that's no good.
Groucho Marx: What's the matter with it?
Chico Marx: I don't know, let's hear it again.
Groucho Marx: All right. It says the first part of the party of the first part shall be known in this contract as the first part of the party of the first part, shall be known in this contract - look, why should we quarrel about a thing like this, we'll take it right out, eh?
Chico Marx: Yes, it's too long anyhow. Now what have we got left?

Moral

the controlled language of contracts needs to be very carefully delimited and is not just NL applied to contracts

Why Contracts are Not Automatically Controlled Languages

Groucho Marx: Now pay particular attention to this first clause, because it's most important. There's the party of the first part shall be known in this contract as the party of the first part. How do you like that, that's pretty neat eh?
Chico Marx: No, that's no good.
Groucho Marx: What's the matter with it?
Chico Marx: I don't know, let's hear it again.
Groucho Marx: All right. It says the first part of the party of the first part shall be known in this contract as the first part of the party of the first part, shall be known in this contract - look, why should we quarrel about a thing like this, we'll take it right out, eh?
Chico Marx: Yes, it's too long anyhow. Now what have we got left?

Moral

the controlled language of contracts needs to be very carefully delimited and is not just NL applied to contracts

Delimiting Contracts

- Agreements between parties, regulating their actions or behaviour.
- Concerning the general scenario of programs and their behaviours.
- Examples
  1. *Upon accepting a job, the system guarantees that the results will be available within an hour unless cancelled in the meantime.*
  2. *Only the owner of a job has permission to cancel the job.*
  3. *The system is forbidden from producing a result if it has been cancelled by the owner.*

Typical Problems

- Upon accepting a job, the system guarantees that the results will be available within an hour unless . . . cancelled in the meantime.
  1. Attachment Ambiguity: which green phrase does the blue phrase modify.
  2. Syntactic complexity caused by ellipsis: the red dots indicate something has been left out.
  3. Reference Ambiguity: what has been cancelled, the job or the results?
  4. Semantic complexity: what exactly does the phrase "in the meantime" refer to?

The controlled language has to eliminate or at least minimise these problems

Typical Problems

- Upon accepting a job, the system guarantees that the results will be available within an hour unless . . . cancelled in the meantime.
  1. Attachment Ambiguity: which green phrase does the blue phrase modify.
  2. Syntactic complexity caused by ellipsis: the red dots indicate something has been left out.
  3. Reference Ambiguity: what has been cancelled, the job or the results?
  4. Semantic complexity: what exactly does the phrase "in the meantime" refer to?

The controlled language has to eliminate or at least minimise these problems

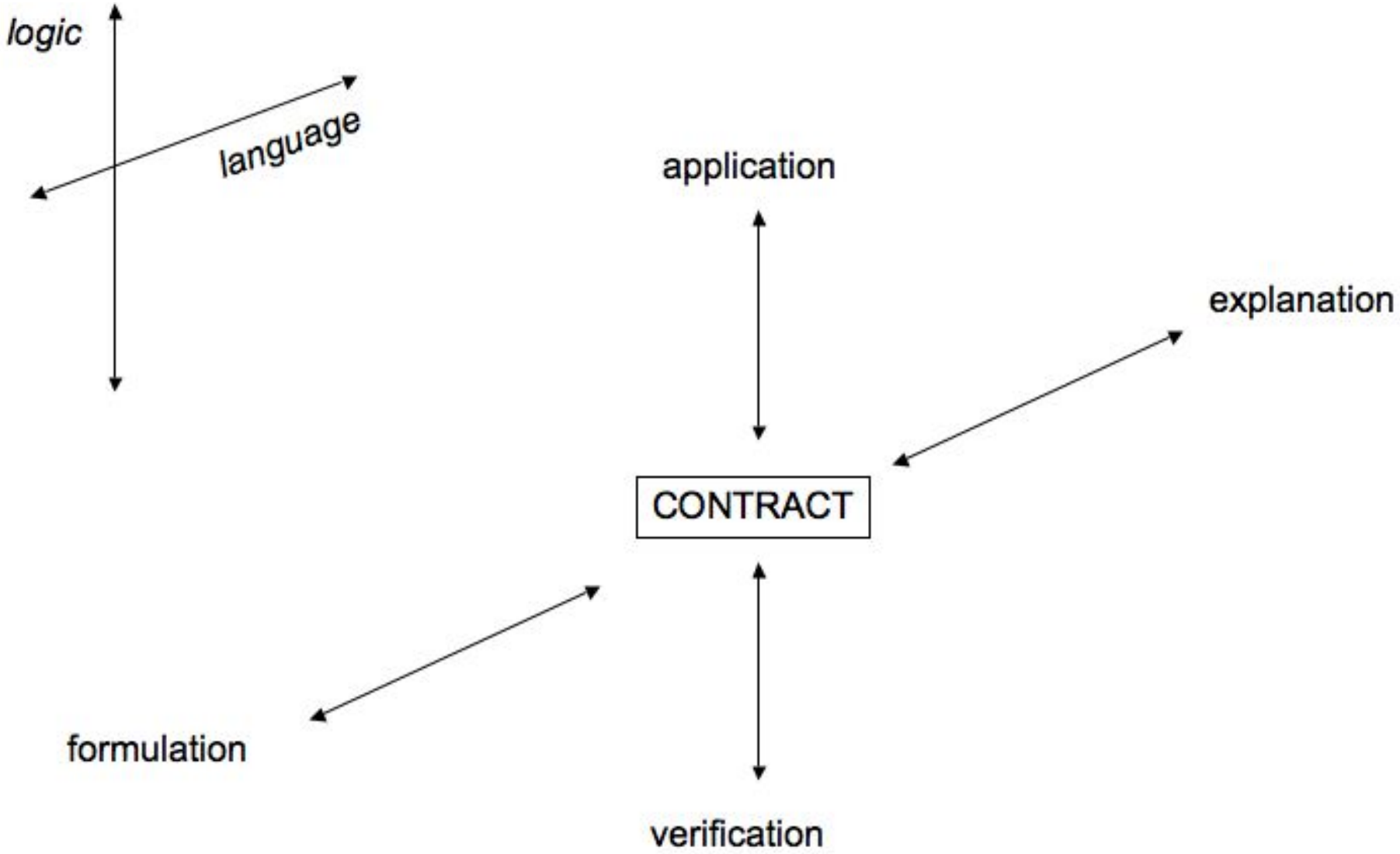
General Shape of Our Solution

- Proper Names
  - Predefined (e.g. SYSTEM)
  - User-defined using initial capital letter (e.g. Job101)
- Rationalised Syntax for Events
  - Inspired by RDF
  - Based on Agent Actor Object triples
- `if SYSTEM accepts Job, then during one hour it is obligatory that SYSTEM make available results of Job unless SOMEONE cancels Job.`

What We Can Do with Contracts

- NL-centric contract-processing tasks
  1. Formulation
  2. Explanation regarding status of particular behaviours and actions
- Reasoning-centric contract-processing tasks
  1. Verification of internal consistency
  2. Testing against actual behaviours
- If we are to provide machine assistance with these tasks, we'd better know what a contract really is.
- The big picture has to include formal models as well as NLP
- Contracts as first class objects
- Contract languages for the specification of such objects

The Big Picture



Deontic Logic for Formal Contracts?

- Ordinary logic is a language for expressing propositions and reasoning with them e.g. Mia robs Vincent.
- Deontic logic adds the ability to express, and reason with, deontic notions such as permission, obligation and prohibition.
- Typically these notions are expressed using special modal operators
  - Fp — it is forbidden that p
  - Op — it is obligatory that p
  - Pp — it is permitted that p

Why Deontic Logic is Suitable for Contract Specification

- Enables experimentation with the formulation of normative notions and their logical consistency.
- Allows us to distinguish ideal from actual behaviour: does actual behaviour x contradict obligation y?
- Explicitly handle such contradictions (e.g. reparations via — contrary-to-duty clauses).
- Issues
  - Ought-to-do (action-based) vs ought-to-be (state-based).
  - Paradoxes unless one is very careful and restricts the language.

Deontic Logic Paradoxes

For example, if obligations are monotonic, the following paradoxes arise:

Ross's Paradox: From "It is obligatory that the letter is mailed," one can conclude that "It is obligatory that the letter is mailed or the letter is burned."

The Good Samaritan Paradox: From "It ought to be the case that Jones helps Smith who has been robbed," it follows that "It ought to be the case that Smith has been robbed."

Combining Deontic and Temporal Notions

Example

The law of a country says that: 'You are obliged to hand in Form A on Monday and Form B on Tuesday, unless officials stop you from doing so.'

On Monday, John spent a day on the beach, thus not handing in Form A. On Tuesday at 00:00 he was arrested, and brought to justice on Wednesday.

The police argue: 'To satisfy his obligation the defendant had to hand in Form A on Monday, which he did not. Hence he should be found guilty.'

But John's lawyer argues back: 'But to satisfy the obligation the defendant had to hand in Form B on Tuesday, which he was stopped from doing by officials. He is hence innocent.'

Combining Deontic and Temporal Notions

Example

The law of a country says that: 'You are obliged to hand in Form A on Monday and Form B on Tuesday, unless officials stop you from doing so.'

On Monday, John spent a day on the beach, thus not handing in Form A. On Tuesday at 00:00 he was arrested, and brought to justice on Wednesday.

The police argue: 'To satisfy his obligation the defendant had to hand in Form A on Monday, which he did not. Hence he should be found guilty.'

But John's lawyer argues back: 'But to satisfy the obligation the defendant had to hand in Form B on Tuesday, which he was stopped from doing by officials. He is hence innocent.'

Who is right?

- This depends on the interpretation of the moment of violation of a sequence of obligations.

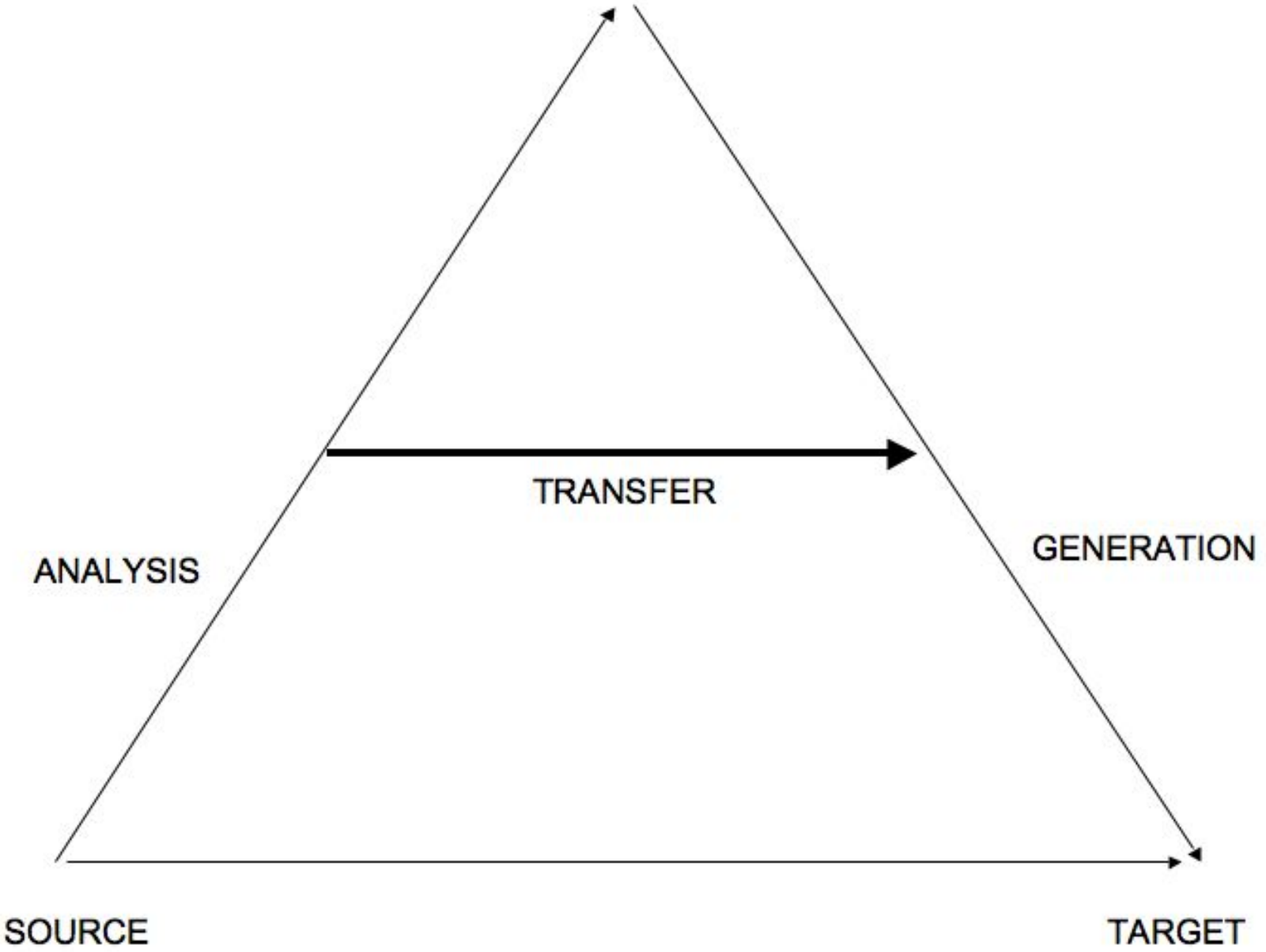
Jobs the Logic Has to Fulfil

- Enable reasoning about contracts as first-class objects.
- Enable verification of programs with respect to a contract.
  - Testing contracts
  - Model-checking programs (static)
  - Static and runtime-monitoring program behaviour
  - Interface restriction (e.g. in virtue of contractual obligations).
- Enable reasoning about contracts independently of models
  - Satisfiability of a contract.
  - Compatibility of two contracts.
- Allow the possibility of precisely specifying contract transformations.
  - Allow any obligations arising from contract A to be satisfied one day later but prohibiting the user from accessing certain services in the meantime.
  - Make all services under contract B 10% more expensive during prime time.
- Link to the CNL:
  - Natural language parsing.
  - Natural language generation.

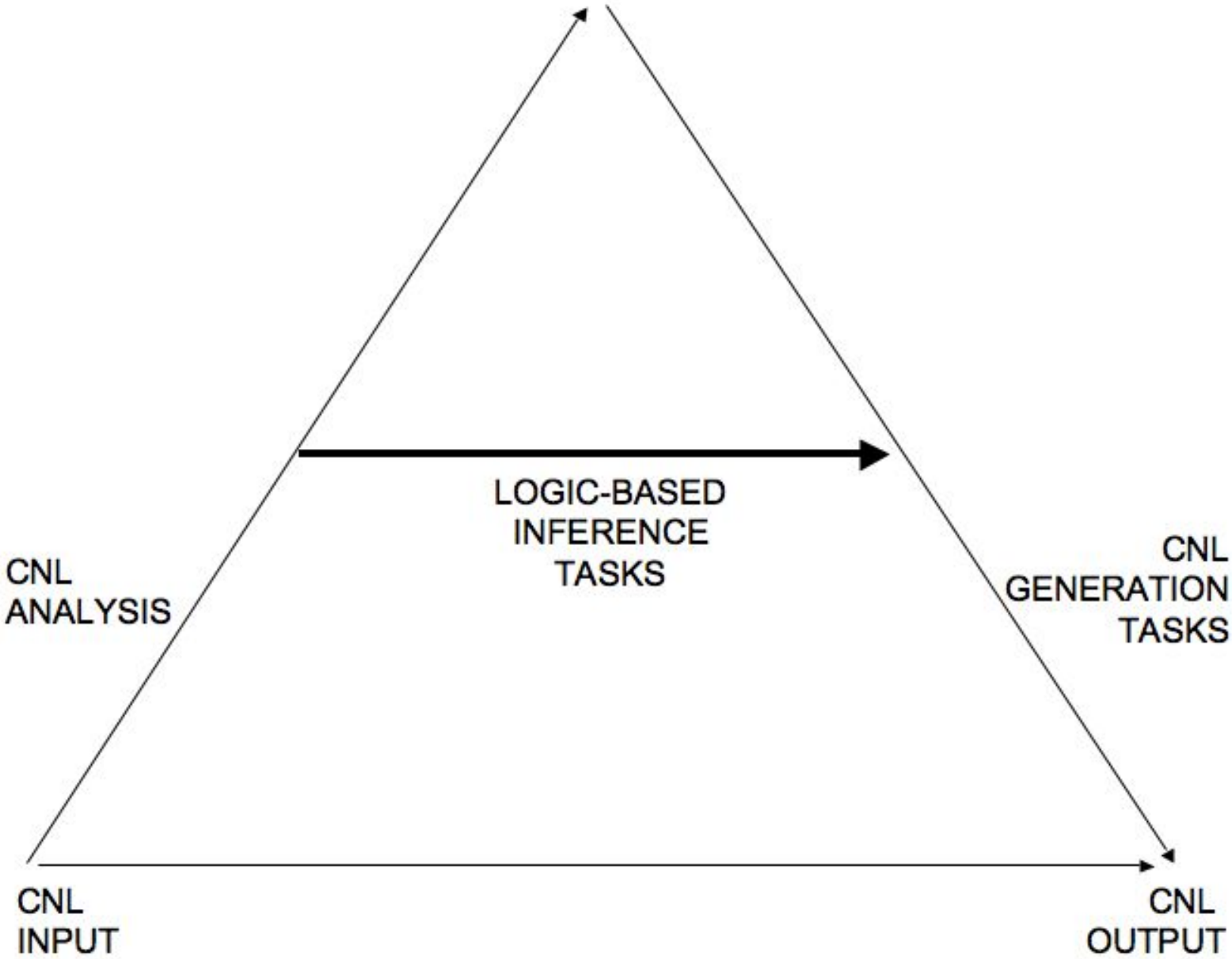
Choices, Choices, Choices

- The logic chosen clearly has to be sufficiently expressive to encompass the domain of the controlled language
- Do we go for a logic with few logical operators, or one with various operators closer matching the controlled language?
- Paradoxes will likely arise in the controlled language — but the logic must be expressive enough to enable reasoning about them, but with a semantics that avoids most of them.
- Off-the-shelf deontic logics usually try to either (i) limit expressivity to avoid the paradoxes; or (ii) circumvent the paradoxes by careful use of the operators. Neither is ideal in our case.

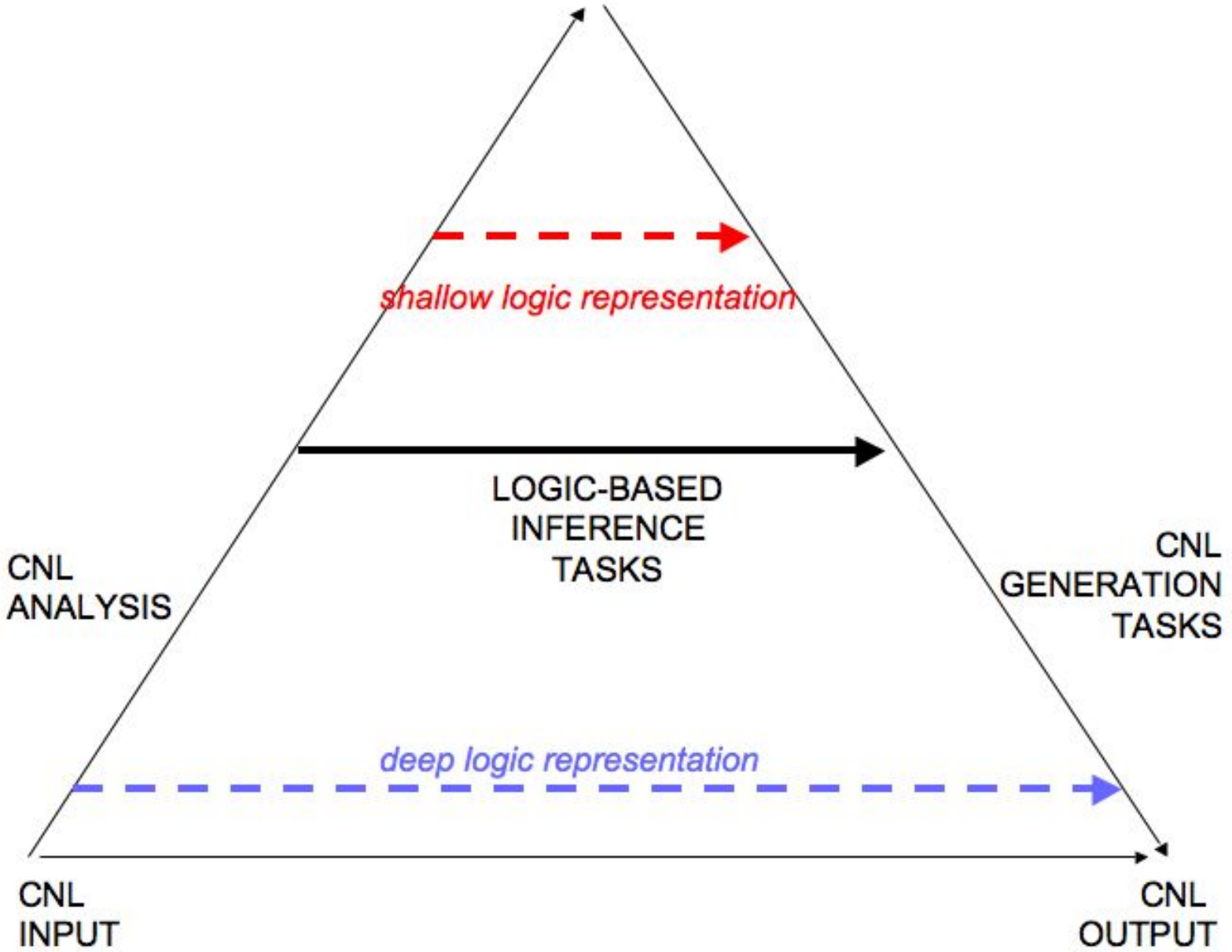
Vauquois Triangle for Machine Translation



Controlled Language Triangle



Deep and Shallow Logical Representations



The Problems of a Low Level Logic

Example: Expressing propositions

All propositional logic operators can be encoded in terms of the nand operator. Why not reduce all propositions to nands?

Pros

- Easy to formalise and show soundness of axiomatisation
- Easy to perform inference

One big con

- Reasonable natural language generation is practically impossible

The Problems of a Logic with Too Much Abstraction

Example: Having redundant operators

In propositional logic, one may keep various operators, including ones which can be defined in terms of each other. For instance, having negation, and disjunction, but also implication.

The big pro

- The closer the operands are to the controlled language, the easier it is to generate descriptions

Cons

- Axiomatisation of too many operators, resulting in potential unsound rules.
- Inference is difficult

The Solution Adopted

- Use a logic whose operators closely match operators in the CNL.
- Use syntactic sugar to avoid redundancy;
- To express operator scope retain certain syntactic features of CNL where possible by *deep embedding* into the logic.
- Higher level reasoning is done using *algebraic manipulation* at the blue level eg look for conflicts, and if any are discovered, explain in terms of the CNL
- Lower level reasoning can be done at the red level after reducing the contract into the underlying lower-level language eg just reporting whether the contract is satisfiable or not
- Essentially take the one-way road down from the blue to the red in a lazy manner.

Syntax of the Contract Language Logic

$$\textit{contract} \quad ::= \quad \top \mid \bot$$

Basic Contracts

The trivially accepted contract $\top$ and trivially refuted contract $\bot$.

Syntax of the Contract Language Logic

$$
\begin{array}{lcl}
\mathit{contract} & ::= & \top \mid \bot \\
& \mid & O(\mathit{agent} : \mathit{action\text{-}expression}) \\
& \mid & P(\mathit{agent} : \mathit{action\text{-}expression}) \\
& \mid & F(\mathit{agent} : \mathit{action\text{-}expression})
\end{array}
$$

Deontic Operators

Obligations, Permissions and Forbidden actions (prohibitions).

Syntax of the Contract Language Logic

$$
\begin{array}{rcl}
\textit{contract} & ::= & \top \mid \bot \\
& \mid & O(\textit{agent} : \textit{action-expression}) \\
& \mid & P(\textit{agent} : \textit{action-expression}) \\
& \mid & F(\textit{agent} : \textit{action-expression}) \\
& \mid & \textit{contract} + \textit{contract} \\
& \mid & \textit{contract} \,\&\, \textit{contract} \\
& \mid & \textit{contract} \lhd \textit{action-expression} \rhd \textit{contract}
\end{array}
$$

Choice, Conjunction, and Conditions

Standard regular expression-like contract combinators.

Syntax of the Contract Language Logic

$$
\begin{array}{rcl}
\mathit{contract} & ::= & \top \mid \bot \\
& \mid & O(\mathit{agent} : \mathit{action\text{-}expression}) \\
& \mid & P(\mathit{agent} : \mathit{action\text{-}expression}) \\
& \mid & F(\mathit{agent} : \mathit{action\text{-}expression}) \\
& \mid & \mathit{contract} + \mathit{contract} \\
& \mid & \mathit{contract} \,\&\, \mathit{contract} \\
& \mid & \mathit{contract} \lhd \mathit{action\text{-}expression} \rhd \mathit{contract} \\
& \mid & \mathit{contract} \blacktriangleleft \mathit{contract} \blacktriangleright \mathit{contract}
\end{array}
$$

Sequentiality

Follow up a contract by another, depending on whether it was satisfied (sequential composition) or broken (reparation).

Syntax of the Contract Language Logic

$$
\begin{array}{rcl}
\mathit{contract} & ::= & \top \mid \bot \\
& \mid & O(\mathit{agent} : \mathit{action\text{-}expression}) \\
& \mid & P(\mathit{agent} : \mathit{action\text{-}expression}) \\
& \mid & F(\mathit{agent} : \mathit{action\text{-}expression}) \\
& \mid & \mathit{contract} + \mathit{contract} \\
& \mid & \mathit{contract} \,\&\, \mathit{contract} \\
& \mid & \mathit{contract} \lhd \mathit{action\text{-}expression} \rhd \mathit{contract} \\
& \mid & \mathit{contract} \blacktriangleleft \mathit{contract} \blacktriangleright \mathit{contract} \\
& \mid & \mathit{contract}_{[\mathit{time},\mathit{time}]}
\end{array}
$$

Timing

Timed-regular expression style restriction of a contract to terminate satisfactorily or not within a time interval.

Syntax of the Contract Language Logic

$$
\begin{array}{rcl}
\mathit{contract} & ::= & \top \mid \bot \\
& \mid & O(\mathit{agent} : \mathit{action\text{-}expression}) \\
& \mid & P(\mathit{agent} : \mathit{action\text{-}expression}) \\
& \mid & F(\mathit{agent} : \mathit{action\text{-}expression}) \\
& \mid & \mathit{contract} + \mathit{contract} \\
& \mid & \mathit{contract} \,\&\, \mathit{contract} \\
& \mid & \mathit{contract} \lhd \mathit{action\text{-}expression} \rhd \mathit{contract} \\
& \mid & \mathit{contract} \blacktriangleleft \mathit{contract} \blacktriangleright \mathit{contract} \\
& \mid & \mathit{contract}_{[\mathit{time},\mathit{time}]}
\end{array}
$$

Syntactic Sugar

Various other operators closer to typical CNL usage can be defined in terms of these operators and fix-points:

- One branch conditional: $e \rightarrow c \equiv c \lhd e \rhd \top$
- Sequential composition: $c_1 ; c_2 \equiv c_2 \blacktriangleleft c_1 \blacktriangleright \bot$
- Always: $\Box(c) \equiv c \;\&\; \top_{[1,1]}; \Box(c)$
- Sometimes: $\Diamond(c) \equiv c + \top_{[1,1]}; \Diamond(c)$

Action Expressions

- Action expressions include temporal operators within just like at the level of contracts.
- These are required for expressivity:
  - John is prohibited from always pressing the button: $P(John : \Box(press))$
  - John is always prohibited from pressing the button: $\Box(P(John : press))$

Examples

- Upon accepting a job, the system guarantees that the results will be available within an hour unless cancelled in the meantime:
  $\Box(accept_j \rightarrow O(system : (result_j + cancel_j))_{[0,1hr]})$
- Only the owner of a job has permission to cancel the job:
  $\Box(P(owner_j : cancel_j)\ \&\ F(\overline{owner}_j : cancel_j))$
- The system is forbidden from producing a result if it has been cancelled by the owner:
  $\Box(cancel_j \rightarrow F(system : (\Diamond(result_j))))$

Semantics of the Contract Language Logic

- The contract language is given a trace semantics using three-valued logic to identify between:
  - Contracts which have been satisfied
  - Contracts which have been irremedially violated
  - Contracts which may still be satisfied or violated in the future
- The semantics keeps trace of deontic informtion — obligations, permissions and prohibitions active at a point in time.
- The semantics are used for various forms of analysis:
  - Check a model or behaviour against a contract
  - Identify potential conflicts in a contract eg you are obliged and forbidden to do the same action at the same point in time.
  - Compare contracts
  - Derive algebraic laws of contracts

Embedding the Logic in Haskell

The logic has been implemented as an *embedded language* in Haskell

- The domain-specific language combinators are implemented as objects in the host language
- Programs are just data objects in the host language which can be manipulated by programs.
- Gives a two-stage language approach.

Embedding the Logic in Haskell

The logic has been implemented as an *embedded language* in Haskell

- The domain-specific language combinators are implemented as objects in the host language
- Programs are just data objects in the host language which can be manipulated by programs.
- Gives a two-stage language approach.

Example

```
stroll :: Person -> Contract
stroll p = shop p <| enterShop |> success

shop :: Person -> Contract
shop p =
   permission(p,leave)
      « obligation(p,pay) »
         obligation(p,returnObject)
```

Embedding the Logic in Haskell

Contract generation Regular families of contracts can be generated automatically, enabling parametrisation on contracts.

Contract transformations Access to contract syntax enables syntactic transformation of contracts.

Contract interpretations The contracts are given different interpretations, including output to external analysis tools and natural language generation.

Contract analysis Semantic analysis can also be performed from within the host language itself, enabling richer transformations.

Example

```
retry :: Integer -> Contract -> Contract -> Contract
c `retry 1` e = success « c » e
c `retry n` e = success « c » (c `retryCTD (n-1)` e)
```

Embedding the Logic in Haskell

Contract generation Regular families of contracts can be generated automatically, enabling parametrisation on contracts.

Contract transformations Access to contract syntax enables syntactic transformation of contracts.

Contract interpretations The contracts are given different interpretations, including output to external analysis tools and natural language generation.

Contract analysis Semantic analysis can also be performed from within the host language itself, enabling richer transformations.

Example

```
retryObl :: Action -> Contract -> Contract
retryObl a (cs « c » cf) = retryObl a cs « retryObl a c » retryObl a cf
retryObl a (Obligation (p,b))
  | a == b = Success « Obligation (p,b) » Obligation (p,b)
  | otherwise = Obligation (p,b)
...
```

Embedding the Logic in Haskell

Contract generation Regular families of contracts can be generated automatically, enabling parametrisation on contracts.

Contract transformations Access to contract syntax enables syntactic transformation of contracts.

Contract interpretations The contracts are given different interpretations, including output to external analysis tools and natural language generation.

Contract analysis Semantic analysis can also be performed from within the host language itself, enabling richer transformations.

Example

```
generate :: Contract -> ControlledLanguage
generate (Timed (b,e) c) = BetweenCL (b,e) (generate c)
generate (Success « c » c') =
  ReparationCL (generate c) (generate c')
...
```

Embedding the Logic in Haskell

Contract generation Regular families of contracts can be generated automatically, enabling parametrisation on contracts.

Contract transformations Access to contract syntax enables syntactic transformation of contracts.

Contract interpretations The contracts are given different interpretations, including output to external analysis tools and natural language generation.

Contract analysis Semantic analysis can also be performed from within the host language itself, enabling richer transformations.

Example

```
hasConflict :: Contract -> Boolean
hasConflict ...
```

Implementing the CNL Grammar

- Current implementation is in PC-PATR
- Rules of the form

```
Rule
simple-event -> agent action object
<simple-event sem> = <action sem>
<simple-event sem> = <agent sem>
<simple-event sem> = <object sem>
```

- Rewrite part builds tree
- Equational part builds F-Structure

Where We Are

- Original: The system is forbidden from producing a result if it has been cancelled by the owner.
- Proposed Translation:
  $\Box(cancel_j \rightarrow F(system : (\Diamond(result_j))))$
- Controlled: if owner of Job cancels Job it is forbidden that SYSTEM produces result of Job
- Computed Translation:

Computed F-Structure Representation

```
if owner of Job cancels Job it is forbidden
that SYSTEM produces result of Job
[ cat:   s
  sem:     [ ante:    [ act:   cancel
                       agent:   [ obj:   J01 ]
                       obj:   J01
                       time:  t ]
             cons:    [ obj:     [ act:   produce
                                  agent: SYS0
                                  obj:     [ obj:   J01
                                             pred:  result ] ]
                       pred:  F
                       time:    [ obj:   t
                                  rel:   gt ] ]
             op:    impl ] ]
```

Conclusions and Future Work

- What we have
  - Logic + Implementation
  - Mechanism for parsing CNL to expressions that are pretty close to defined
- What we need (soon)
  - Close the gap
  - NLG from logic to CNL
- Future
  - Related domains - SOAs, simple legal contracts
  - Explanation
  - Discourse issues when considering complex contracts comprising several clauses.