CLAN: Contract Language ANalyser

Today's industry is pushing towards service-oriented architecture where the different functionality required by an organisation is distributed as services. These services need not necessary be on machines inside the organisation itself but could also be found outside of the organisation's borders. Since in such situations the organisation is executing code provided by third parties it requires some form of mechanism to protect itself. One such mechanism is the making use of Legal Contracts.

Furthermore, services are typically composed of a number of other services which in turn could again be composed of further services. Each of these would have their own contracts defining what they commit to perform and what they expect from the interaction. Such contracts may be quite complex, especially when considering the nesting of these contracts.

With this in mind, it would be desirable to be able to represent such contracts formally in order to avoid problems of ambiguity and diverse interpretations of such contracts. Furthermore, we could then apply a number of automatic analysis which will enable us to ensure that the contracts posses certain properties, for example that they are conflict free.

The Contract Language (CL) is aimed at specyfing such contracts in a formal manner. Once we have this formal representation using CL we can then apply a number of techniques in order to analyse the contracts. CLAN is an authomatic tool that performs such analysis. Using the trace semantics of CL, CLAN will generate an automaton representing the CL clauses which is then used for the analysis of the contract. The current version of CLAN is able to:

- Automatically analyse the contract for Conflicts where by conflict we mean that the contract is enforcing two opposing and contradicting actions and thus cannot be satisfied, for example being obliged to send a reply and forbidden to send a reply at the same time.
- Automatically generates a monitor for the contract.

CLAN can be extended to perform much more types of analysis and currently work is being done so as to be able to perform the following:

- Generate an SMV model from the contract.
- Analyse the contract for overlapping clauses.
- Analyse the contract for unreachable sub-clauses of the contract.
- Analyse the contract for superfluous clauses.

These extensions have the theoretical work already been done and simply need to be implemented. A more ambitious target would be to be able to model check such contracts. Once we automatically translate the contract into an SMV model we will be able to verify that a contract has certain temporal properties, however, it would be ideal to be able to model check properties written in CL. Furthermore, there is still work being done in improving CL at which point, this tool will evolve as well.

Papers

[1] **Conflict Analysis of Deontic Conflicts**, Stephen Fenech, M.Sc.(Computer Science) (PDF)

[2] **The CoCoME Specification in *CL***, Stephen Fenech, (supporting report PDF)

[3] **On the Specification of Full Contracts**, Stephen Fenech, Joseph Okika, Gordon Pace, Anders P. Ravn and Gerardo Schneider, in 6th International Workshop on Formal

Engineering approaches to Software Components and Architectures (FESCA'09), ENTCS, 2009. (PDF)

[4] **Automatic Conflict Detection on Contracts**, Stephen Fenech, Gordon Pace and Gerardo Schneider, in 6th International Colloquium on Theoretical Aspects of Computing (ICTAC'09), LNCS, 2009. (PDF)

[5] **CLAN: A Tool for Contract Analysis and Conflict Discovery**, Stephen Fenech, Gordon Pace and Gerardo Schneider, in 7th International Symposium on Automated Technology for Verification and Analysis (ATVA'09), LNCS, 2009.

The Toolkit

Introduction to CL

CL is a language based on Deontic and Dynamic logic, allowing us to represent obligations, prohibitions and permissions together with temporal properties. We can also describe exceptional situations where in literature are called Contrary to Duties (CTDs) and Contrary to Prohibitions (CTPs). The following is the CL syntax:

$$
\begin{aligned}
C &:= C_O | C_P | C_F | C \wedge C | [\beta]C | T | \bot \\
C_O &:= O_C(\alpha) | C_O \oplus C_O \\
C_P &:= P(\alpha) | C_P \oplus C_P \\
C_F &:= F_C(\delta) | C_F \vee [\alpha] C_F \\
\alpha &:= 0 | 1 | a | \alpha \& \alpha | \alpha ; \alpha | \alpha + \alpha \\
\beta &:= 0 | 1 | a | \beta \& \beta | \beta ; \beta | \beta + \beta | \beta^*
\end{aligned}
$$

A contract clause C can be either an Obligation (C_O), a Permission (C_P) a Prohibitions (C_F) or the conjunction of two clauses ($C \wedge C$). Furthermore, $[\beta]C$ is interpreted that if action β is observed, the clause C must hold in the following instant and T; is the trivial contract whereras $\bot$ is the impossible contract. An Obligation clause could be a CTD where $O_C(\alpha)$ is interpreted as one is obliged to perform action α and if this obligation is violated then clause C comes into effect. An obligation could be the exclusive disjunction of two obligation clauses. A Permission is similar to the obligation however does not have a reparation. Prohibition is similar to Obligation as well, however we do not have the exclusive disjunction between two prohibition clauses.

Atomic actions take the form of 0 (the impossible action), 1 (the satisfied action) and a, where a stands for any atomic action defined by the user. Compound actions could be created by using the concurrancy operator (&), the sequence operator(;) and the choice operator(+). There is also the Kleene star operator(*).

Introduction to the Toolkit

Download the tool:CLAN.zip

Installation

There is no actual installation to make use of this tool. The tool can be dowloaded from here. The tool is zipped and should be extracted into any desired folder. In the zip file you should find the tool that is bundled as an executable jar and a folder with a number of third party libraries which are used. One will require Java in order to use the tool. In order to execute the tool using the command prompt/terminal go to the folder where you have downloaded the jar and execute "java -jar clan.jar".

Usage

Synatx

Action Operators

!	not	*	Kleene star
&	concurrent	+	choice
.	sequence		

Deontic Operators

O	Obliged	F	Forbidden
P	Permission	_	Reparation

Boolean and Dynamic Operators

^	And	|	Disjunction
-	Exclusive Disjunction	[]	Necessary

The precedence is as follows:

For the action Operators !*&+.

For the Deontic Operators OFP_

For the boolean operators ^|-

Contact details

Stephen Fenech
Department of Computer Science,
Faculty of ICT,
University of Malta,
Msida MSD 06,
MALTA

Gordon J. Pace
Department of Computer Science,
Faculty of ICT,
University of Malta,
Msida MSD 06,
MALTA

Tel: (+356) 2340 2504
Fax: (+356) 2132 0539
E-mail:
Gordon.Pace@um.edu.mt

Gerardo Schneider
Department of Informatics,
University of Oslo,
P.O. Box 1080 Blindern,
NO-0316 Oslo,
NORWAY

Tel: (+47) 22 85 29 71
Fax: (+47) 22 85 24 01
E-mail:
gerardo@ifi.uio.no