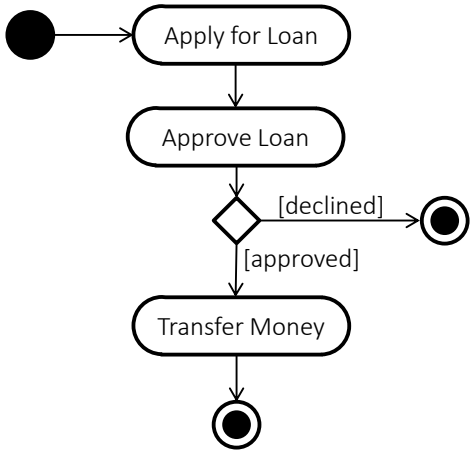
Introduction to Constraint DSLs

This document provides an introduction to three different constraint specification DSLs (Domain Specific Languages). To foster understanding of the purpose of these languages, we will now discuss an example related to business process management and the general framework that all three specific constraint DSLs share.

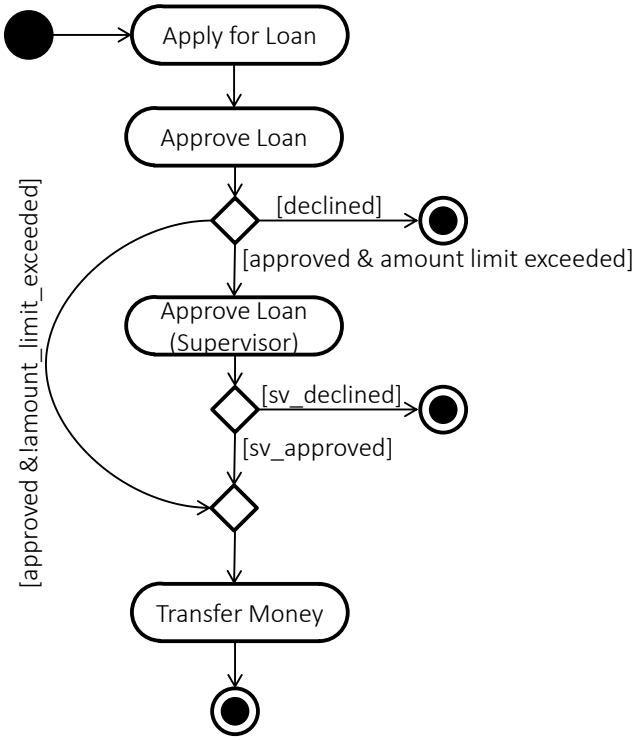
In many domains, such as health, insurance and banking, processes must comply with various rules stemming from sources such as regulations, laws and standards. By using a constraint DSL, such rules are formalized and can be automatically verified. An example of a rule could be a loan application that must be approved by two different bankers (one has the role of a supervisor) before the money is transferred if an amount threshold of 100.000 Euro is exceeded.

Let us consider the following two activity diagrams:

Process 1:



Process 2:



Process 1 is obviously incorrect because it does not take the necessary approval of the supervisor into account. In contrast, *Process 2* considers the rule. When these processes are executed, execution *traces* are created. A possible trace of Process 1 is [“Apply for Loan”.started, “loan amount = 110.000”, “Apply for Loan”.finished, “Approve Loan”.started, “approved”, “Approve Loan”.finished, “Transfer Money”.started, “Transfer Money”.finished]. At the beginning of the trace, the rule is *temporarily satisfied* but as soon as “approved” occurs, the constraint’s truth value changes to *temporarily violated* because an

approval action of the supervisor is pending due to “loan amount = 110.000”. Once “Transfer Money”.started occurs in this trace, the constraint becomes *permanently violated* because the necessary approval of the supervisor is missing. An example trace of *Process 2* could be [“Apply for Loan”.started, “loan amount = 110.000”, “Apply for Loan”.finished, “Approve Loan”.started, “approved”, “Approve Loan”.finished, “Approve Loan (Supervisor)”.started, “sv_approved”, “Approve Loan (Supervisor)”.finished, “Transfer Money”.started, “Transfer Money”.finished]. Again, the constraint starts as *temporarily satisfied* but becomes *temporarily violated* with the occurrence of “approved”. Once “sv_approved” happens, the rule is completely met and changes its truth value to *permanently satisfied*.

An example for the general framework of the three DSLs is:

```
Task approve_loan 550e8400-e29b-11d4-a716-446655440000
Task grant_loan 550e8400-e29b-11d4-a716-446655440001
Number loan_amount 550e8400-e29b-11d4-a716-446655440002
Role banker 550e8400-e29b-11d4-a716-446655440003
Role supervisor 550e8400-e29b-11d4-a716-446655440004

Constraint c1 for LoanApprovalCase {
      <DSL-specific Constraint>
      Mandatory: yes
      Documentation: “[…]”
}
```

The wildcard “<DSL-specific Constraint>” will be our focus. There are three different Constraint DSL approaches that can be embedded in this general constraint framework, namely *LTL-SNL*, *TQL* and *CoLa*.

For the sake of simplicity, we will in the following examples abstract from actions such as the start of a named task and named data attributes, and represent them by capital letters.

1. LTL-SNL

The first DSL is called *LTL-SNL* which is an abbreviation of *Linear Temporal Logic Structured Natural Language*. This language is composed of the following *operators*:

- ***finally*** p, which means that p must eventually hold,
- ***globally*** p, which means that p must always hold,
- ***next*** p, which means that p must hold in the directly following state,
- p_1 ***until*** p_2, which means that p_1 must hold until p_2 holds,
- p_1 ***weak-until*** p_2, which means that p_1 must always hold or p_1 must hold until p_2 holds,
- p_1 ***implies*** p_2, which is a logical implication,
- p_1 ***or*** p_2, which is a logical disjunction,
- p_1 ***and*** p_2, which is a logical conjunction,
- ***not*** p, which is a logical negation.

OPERATOR PRECEDENCE:

1. finally, globally, next, not
2. until
3. weak-until
4. and
5. or
6. implies

We will now use these operators in examples to become more familiar with them.

EXAMPLE 1.1 Let us assume a constraint ***finally Q*** and the following traces:

- Trace 1: [R, S, T, U, W, X, A, B, C, S]
- Trace 2: [R, S, T, U, Q, X, A, B, C, S]

In Trace 1, Q does not occur, therefore the constraint is violated. In particular, the constraint is *temporarily violated* because there is still a chance that Q will happen when new elements are added to the trace. In Trace 2, Q happens at position 5. The constraint is *temporarily violated* before Q occurs. With the occurrence of Q, the constraint's truth value changes to *permanently satisfied*.

EXAMPLE 1.2 Let us assume a constraint ***globally Q*** and the following traces:

- Trace 1: [R, S, T, U, Q, X, A, B, C, S]
- Trace 2: [Q, Q, Q, Q, Q, Q, Q, Q, Q]
- Trace 3: [Q, Q, Q, Q, Q, Q, Q, Q, Q, R]

Trace 1 does not contain Q at every position and it is impossible to satisfy the constraint even by adding new elements at the end of the trace. Therefore, the constraint is *permanently violated*. In contrast, Trace 2 fulfills the constraint but there is still a chance that another element than Q is added in the future. Thus, Trace 2 *temporarily satisfies* the constraint. In Trace 3, an element R is added to the trace which causes a change of the truth value from *temporarily satisfied* to *permanently violated*.

EXAMPLE 1.3 Let us assume a constraint ***globally(Q implies next R)*** and the following traces:

- Trace 1: [R, S, T, U, Q, X, A, B, C, S]
- Trace 2: [R, S, T, U, Q, R, A, B, C, S]
- Trace 3: [Q, R, S, T, C, Q]
- Trace 4: [R, S, T]

The constraints meaning is "Every time Q happens, it must be directly followed by R". Trace 1 contains Q at position 5 but this Q is not immediately followed by R. Therefore, the constraint becomes *permanently violated* at position 6. In contrast, Trace 2 fulfills the constraint because Q is directly followed by R but there is still a chance that it becomes violated in the future. Thus, Trace 2 *temporarily satisfies* the constraint. In Trace 3, the first Q is directly followed by an R but the Q at the end of the trace is not yet

followed by an R, therefore the constraint is *temporarily violated*. Trace 4 does not contain Q, therefore the constraint is *temporarily satisfied*.

EXAMPLE 1.4 Let us assume a constraint ***R until T*** and the following traces:

- Trace 1: [R, R, S, R, T]
- Trace 2: [R, R, T]

In Trace 1, there is not only R but also S occurring until T eventually occurs. With the occurrence of S, it becomes impossible to satisfy the constraint, therefore its truth value changes from *temporarily violated* to *permanently violated*. In Trace 2, the truth value is *permanently satisfied* at the end of the trace.

EXAMPLE 1.5 Let us assume a constraint ***R weak-until T*** and the following trace:

- Trace 3: [R, R, R]

This example is an extension of Example 1.4. Trace 3 contains only R which causes the constraint to be *temporarily satisfied*.

EXAMPLE 1.6 Let us assume a constraint ***finally(A and finally B)***. This constraint demands that eventually A happens and after that there must eventually happen B.

EXAMPLE 1.7 Let us assume a constraint ***finally A and finally B***. This constraint demands that eventually A and eventually B happen but the order is not important.

EXAMPLE 1.8 Let us assume a constraint ***finally A and globally not B***. This constraint demands that eventually A happens but B does not happen at all.

2. TQL

The *Temporal Query Language* (*TQL*) is built on top of EPL (Event Processing Language) but uses only a subset of it and extends it for runtime verification. It is necessary to specify an initial truth value for a constraint which is done by ***initial truth value: <ITV>*** where <ITV> is either "*temporarily satisfied"* or "*temporarily violated"*. Each constraint has between one and four *queries*:

- **temporarily satisfied query: <q_{TS}>**
- **temporarily violated query: <q_{TV}>**
- **permanently satisfied query: <q_{PS}>**
- **permanently violated query: <q_{PV}>**

A query q_i is composed of either a single event *e* or the following expressions:

- e_1 ***leads-to*** e_2, which means that e_1 is eventually followed by e_2,
- ***not*** e_1 ***until*** e_2, which means that e_1 does not occur until e_2 happens,
- ***every*** *e,* which means that every occurrence of *e* is observed,

- e_1 ***or*** e_2, which is a logical disjunction,
- e_1 ***and*** e_2, which is a logical conjunction,
- ***not*** e, which is a logical negation that starts as true and becomes negated by the occurrence of e.

OPERATOR PRECEDENCE:

1. every, not
2. until
3. and
4. or
5. leads-to

To become familiar with the language, we will now discuss two examples.

EXAMPLE 2.1 Let us assume a constraint

initial truth value: temporarily satisfied
temporarily violated query: Q
permanently satisfied query: Q leads-to R

and the following traces:

- Trace 1: [A, B, C, D]
- Trace 2: [A, Q, T, C, R, B, Q, T]

In Trace 1, Q does not occur, therefore the truth value remains *temporarily satisfied*. With the occurrence of Q in Trace 2, the truth value changes from *temporarily satisfied* to *temporarily violated*. With the occurrence of R, the truth value becomes *permanently satisfied*. Q occurs a second time in Trace 2 but the truth value does not change because of two reasons: First, the temporarily violated query fires just for the first occurrence of Q (for an opposite example where every Q causes a firing of the query see Example 2.2). Second, a constraint which has arrived at a permanent state can no longer change to another state.

EXAMPLE 2.2 Let us assume a constraint

initial truth value: temporarily satisfied
temporarily violated query: every Q
temporarily satisfied query: every(Q leads-to not T until R)

and the following traces:

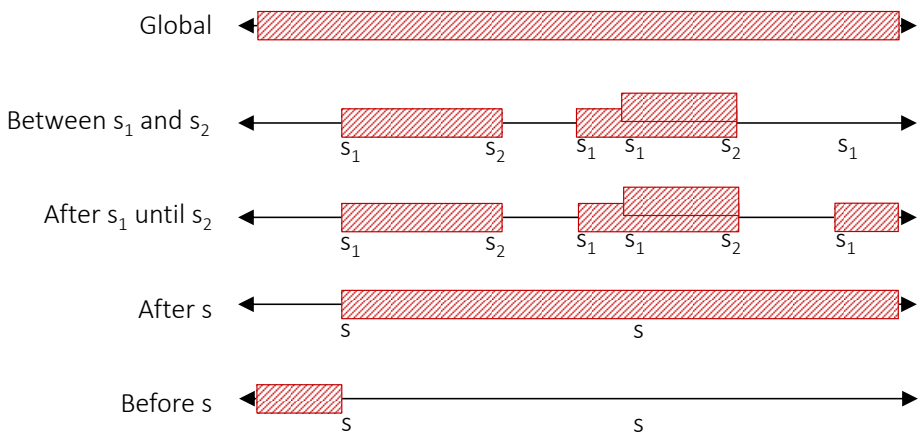
- Trace 1: [A, B, Q, C, D, R, D]
- Trace 2: [A, B, Q, C, T, R, D]
- Trace 3: [A, B, Q, C, D, R, Q]

In Trace 1, the occurrence of Q causes a truth value change to *temporarily violated*. This Q is followed by an R without any T in between. With the occurrence of this R, the truth value changes to *temporarily satisfied*. In Trace 2, there is a T in between Q and R, therefore the truth value remains *temporarily violated*. In Trace 3, there occurs a second Q which causes a change to *temporarily violated*.

3. CoLa

CoLa is an acronym for *Constraint Language*, a high-level constraint language. Constraints in CoLa can have one of the following *scopes*:

- Global Scope: The constraint must always hold
- Between Scope: The constraint must hold between a state s_1 and s_2
- After Until Scope: The constraint must hold after a state s_1 by no later than s_2
- After Scope: The constraint must hold after *s*
- Before Scope: The constraint must hold before *s*



By default, the Global Scope is chosen for a constraint. Otherwise, the scope is defined explicitly:
<scope> [<constraint>]

The following *operators* are available:

- *a* ***leads-to*** *b*, which means that *a* must be eventually followed by *b* in the given scope,
- *a* ***precedes*** *b*, which means that *b* is not allowed to happen unless *a* has already happened in the given scope,
- *a* ***never occurs***, which means that *a* must not happen at all in the given scope,
- *a* ***occurs***, which means that *a* must happen at least once in the given scope.

EXAMPLE 3.1 Let us assume a constraint ***between Y and K [T precedes C]*** and the following traces:

- Trace 1: [A, B, D, K, Y, C, T]
- Trace 2: [A, B, Y, A, Y, C, B, T, K]
- Trace 3: [A, B, Y, A, Y, T, B, C, K]
- Trace 4: [A, B, Y, A, Y, T, B, C, K, Y, C, K]

In Trace 1, the scope "*between Y and K*" does not exist; therefore the constraint is *temporarily satisfied*. In Trace 2, this scope exists and C occurs inside the scope without a T in front of it, therefore the constraint is *permanently violated*. In contrast, Trace 3 contains the scope and there is a T in front of it, therefore the constraint is *temporarily satisfied*. In Trace 4, the scope exists another time and inside of it is the T missing in front of C, which causes the constraint to be *permanently violated*.