IMPERIAL COLLEGE LONDON

INDIVIDUAL PROJECT REPORT

DEPARTMENT OF COMPUTING

Business Management Processes: Verifying their Compliance with Security and Business Rules

Author:
Joanna DIEP

Supervisor:
Professor. Michael HUTH
Second Marker:
Dr. Anandha GOPALAN

May 31, 2015

Abstract

TODO

Acknowledgements

I would like to whole heartedly thank my supervisor Professor. Michael Huth for his continuous and invaluable advice, feedback and support throughout the course of this project. As well as giving time for meetings to discuss ideas.
I would like to thank Dr. Anandha Gopalan, the second marker for their feedback and suggestions to greatly improve my report.
Finally, I would like to thank my family and friends for their love, support and who have had to put up with me throughout this project and throughout my time at Imperial College.

Contents

Chapter 1

Introduction

Chapter 2

Background

2.1 Workflows

Business management processes can be represented as a workflow of tasks, where each of these tasks produce a certain output for the next task to be realised. These tasks are assigned in the workflow as a sequence, where in order to perform the next task, the current task must be completed. However, in order to start a task, it must meet all the constraints within workflow. However, in a large business, making sure that all the constraints are met in a workflow can become a large problem as the more tasks are added to a workflow. Workflows are modeled as directed acyclic graphs [1], where there are no directed cycles. The vertices are tasks in the workflow and the directed edges connecting to each task vertex is the execution order.

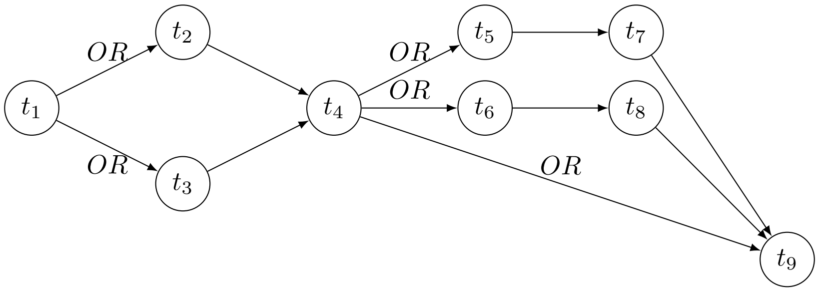
An example of a business workflow is given in Figure 2.1a with nine tasks that need to be allocated:

- A user in the business receives the order from a customer.
- They then pass it onto another user depending on the two possible sizes of the order and are given different prices accordingly.
- Someone then needs to approve and authorise the price for checkout.
- Then a discount may be provided depending on the current total cost of the order.
- Finally, the sale is approved and the new total is returned back to the customer.

However, the order of of execution in Figure 2.1a is affected by how to graph is forked. These forks are represented as constraints or rules within a business which may prevent tasks being executed such as a government restriction on business logic. The fork at t_1 is an OR-fork showing that either t_2 or t_3 can be executed. Depending on whether t_2 or t_3 can be executed, determines whether t_4 can then be executed afterwards and affects the whole execution of the workflow.

2.2 Tasks

In business management processes, users have to be allocated tasks for an execution to occur. These tasks are represented as vertices in the graph as t_n. The tasks in the example workflow given in Figure 2.1a are listed in Table 2.1b.
If users cannot be allocated to these tasks, then there is no way in which the task can be executed. Therefore it is not executed and the workflow becomes unsatisfiable.



(a) Ordered business workflow with nine tasks

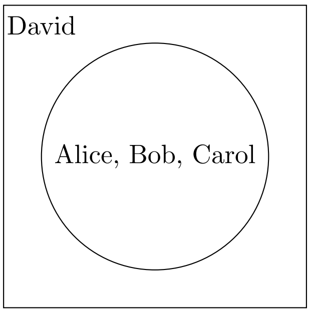
Task Number	Task
t_1	Receive order from customer
t_2	Give total of large sale
t_3	Give total of small sale
t_4	Approve and authorise checkout
t_5	Give 10% discount
t_6	Give 20% discount
t_7	Give new total of sale
t_8	Give new total of sale
t_9	Approve and return new total

(b) Table of tasks

Figure 2.1: Business management process workflow

2.3 Users

A user belongs to the set of users who can be allocated to tasks in order to execute them. However, there are possible allocation constraints which refrain particular users from executing these tasks. In the example given below, Alice, Bob and Carol are users specified in the domain of the workflow, but David is not.



2.4 Business and Security Rules

Business and security rules are used to prevent fraud and follow business compliance rules. For example, in some cases, different users are needed to execute a certain set of tasks to perhaps prevent fraud or erroneous activities in a workflow. In Figure 2.1 some constraints about which users in the business can execute these tasks are added below:

- Whomever is allocated task t_1 to receive the order from customers and pass them onto the relevant user, but cannot be allocated and execute t_2 and t_3, but can be allocated other tasks within the workflow besides t_2 and t_3.

- Whomever is allocated task t_2 to give a total of a large sale cannot be the same user that is allocated t_3 who gives a total of a small sale. Therefore whomever is allocated t_3 cannot be allocated to t_2.

With the additional constraints to the workflow it may be satisfiable given that there are enough users. But there may be other constraints that can make this workflow unsatisfiable. For example, if there are not enough users to be allocated to ensure that some of the tasks do not have the same user. So a valid workflow is a satisfied workflow if there can be users allocated tasks in the workflow that do not break the constraints within the given model.

2.5 Satisfiability Modulo Theories (SMT)

Satisfiability Modulo Theories (SMT) [2] check the satisfiability of logical formulas over given theories. It helps to determine whether there is a solution in a formula which expresses a constraint. It is one of the fundamental problems in the area of computer science to check boolean satisfiability over logical domains and the completeness and incompleteness of logical theories and complexity theory.
SMT is similar to Boolean or Propositional Satisfiability Problem (SAT) [3] , where the problem is to determine if there exists a determination that satisfies the boolean formula. But SAT ranges only over binary predicates which are predicated that only take in two arguments in their formula. Whereas SMT covers non-binary predicates with types and sorts. This project will focus on SMT rather than SAT as we can use non-binary predicates provided by SMT to define and solve some of these constraints.

2.6 Z3

Z3 [4] is a SMT solver developed by Microsoft Research. It is used to integrate several decision procedures and verify the satisfiability of logical formulas over given theories. The theories our case, is the workflow model. There are many features of Z3 which will come in useful including:

- Uninterpreted functions - where a theory has an empty set of sentences. An example of this can be an axiom, where the satisfiability of the axiom depends on whether the uninterpreted function can be evaluated to true.
- Linear arithmetic
- Bitvectors, arrays, datatypes
- Quantifiers
- Satisfiability core
- Returns a model

There were many other SMT solvers considered but did not include certain built in theories and features such as:

- Yices [5] - it has almost all features of Z3 but doesnt have quantifiers, which is needed to define general rules to satisfy a formula in the domain.
- CVC4 [6] - Similar features of Z3 including quantifiers not included in Yices. But typically with CVC4, it is not very scalable. It is intended to run with small finite models, but realistically, business processes can be huge within large organisations.
- MathSAT 5 [7]- It is lacking a lot of features in Z3, especially the quantifiers.

2.6.1 Basics

A simple example is illustrated in Figure 2.2 shows how some simple first order predicate logic in Figure 2.2a can be expressed in Z3 SMT solver in Figure 2.2b.

- To define constants in Z3, in this case x and y, we can declare them as constants using the keyword `(declare-const x Int)` where x is the name of the constant and the type of x is an Integer.

- Z3 uses assertions to add constraints to the solver as a keyword assert.
- `(check-sat)` (line 5) is a call to Z3 to check if the theory is satisfiable. It returns sat is the theory is satisfiable and unsat if the theory is unsatisfiable.
- `(get-model)` (line 6) is a call to Z3 to return an interpretation of the theory which makes all the formulas defined in the Z3 stack true. If the theory returns false, no model able to be retrieved.

In the example below, $x > 10$ is given as an assertion `(assert(> x 10))` and $y \times 10 \geq x$ is given the assertion `(assert (>= (* y 10) x))`. When this is run in Z3, a result is returned. `(check-sat)` returns as sat which means that this theory is satisfiable.
What we can see also, is that Z3 gives back an appropriate model as a result that satisfies these constraints with $x = 11$ and $y = 2$. This is true as if we put $x = 11$ and $y = 2$ back into the constraints, $11 > 10$ and $20 \geq 11$.

$$x > 10$$
$$y \times 10 \geq x$$

(a) Simple predicate logic using logic symbols

```
1 (declare-const x Int)
2 (declare-const y Int)
3 (assert (> x 10))
4 (assert (>= (* y 10) x))
5 (check-sat)
6 (get-model)
```

(b) Simple predicate logic in Z3

```
sat
(model
  (define-fun y () Int
    2)
  (define-fun x () Int
    11)
)
```

(c) Z3 Result

Figure 2.2: Simple predicate logic

2.6.2 Functions

Z3 also has uninterpreted functions where unlike most programming languages where functions have side effects, may never return a value or raise or throw exceptions, Z3 functions have no side effects since they are in classical first order logic and are total. Everything in Z3 is a function, including constants as they dont take in arguments.

- `(declare-fun f (Int) Int)` - We declare a function f which takes in and integer as its parameter and returns an integer

In Figure 2.3, a function `f` has been declared which takes an integer as input, and returns an integer. Since this is an uninterpreted function, Z3 does not know what this function does. But we can add some constraints, so when we apply the function to the integer, it ensures that the interpretation is consistent within the theory and constraints.
In Figure 2.3b, there are two assertions `(assert (= (f x) x))` which represents $f(x) = x$ and `(assert (> (f y) (f x))` as $f(f) > f(x)$. The result that Z3 returns in Figure refZ3 Result with functions has still kept the values of `x` and `y` as it is the same as Figure 2.2. But for function f, it takes in an integer as we have specified in our function declaration as `(x!1 Int)` which means that the first variable has a

type Int (integer). It returns an integer which is consistent and the type integer is interpreted.
Looking at the model returned in Figure 2.3c, we can see that `x` and texttty are both interpreted as a function as well as texttf. It interprets `f` to take in an integer, the `ite` stands for if-then-else. So we can read the definition of `f` as if x!1 is equal to 20, then return 20, else if x!1 is equal to 2, then return 2 else, return 21. Else, return 20. So for the case that `x` is put into the function `f`, then $x!1 = 20$, then the value of the function is 20, else if `y` is put into the function, then $x!1 = 2$, then the value of the function is 21. If any other input is put in, then it will return 20.
Z3 also includes built in arithmetic functions such as $=, -, +, \times, div, mod, \geq, \leq, >, <, not$ that support integer and real constants.

$$x > 10$$
$$y \times 10 \geq x$$
$$f(x) = x$$
$$f(y) > f(x)$$

(a) Predicate logic with functions

```
(declare-const x Int)
(declare-const y Int)
(assert (> x 10))
(assert (>= (* y 10) x))
(declare-fun f (Int) Int)
(assert (= (f x) x))
(assert (> (f y) (f x)))
(check-sat)
(get-model)
```

(b) Z3 with functions

```
sat
(model
  (define-fun y () Int
    2)
  (define-fun x () Int
    20)
  (define-fun f ((x!1 Int)) Int
    (ite (= x!1 20) 20
    (ite (= x!1 2) 21
      20)))
)
```

(c) Z3 Result with functions

Figure 2.3: Predicate Logic with Functions

2.6.3 Stack

Z3 has a stack implementation, where constraints and formulas can be pushed onto and popped off the stack using the commands `(push)` and `(pop)` which pushes and pops constraints off the stack respectively. These commands can be used to check the satisfiability of some rules or definitions. When the solver stack is pushed, the state of the solver is saved. When the stack is popped, any rules and assertions declared between that pop and the corresponding push on the stack is removed from the stack, and the interpretation is reverted back to its previous state before the push.
In Figure 2.5, the theory is satisfied before the push. However, when the stack is pushed and an assertion is added which violates the constraints already in the current frame which is $x < 2$, but $x > 2$ has been pushed onto the stack frame previously, the model becomes unsatisfied. Since the constraint $x < 2$ was between a push-pop frame, it can be popped off the stack and the model is returned back to its previous

state on the stack.

```
(declare-const x Int)
(declare-const y Int)
(declare-fun f (Int) Int)
(assert (> x 2))
(assert (< y 2))
(assert (= (f x) (f y)))
(check-sat)
(get-model)
(push)
(assert (< x 2))
(check-sat)
(pop)
(check-sat)
(get-model)
```

(a) Z3 with stack

```
sat
(model
  (define-fun y () Int
    0)
  (define-fun x () Int
    3)
  (define-fun f ((x!1 Int)) Int
    (ite (= x!1 3) 1
    (ite (= x!1 0) 1
      1)))
)
unsat
sat
(model
  (define-fun y () Int
    0)
  (define-fun x () Int
    3)
  (define-fun f ((x!1 Int)) Int
    (ite (= x!1 3) 1
    (ite (= x!1 0) 1
      1)))
)
```

(b) Z3 Result with stack

Figure 2.4: Predicate Logic with stack

2.6.4 Sorts

When a constant is defined, they are declared as a type which is a sort in Z3. For example, integers, reals and booleans are declared, they are a pre-defined sort in Z3.

- `(define-sort t1 Task)` - the command defines a new symbol with the type Task.

2.6.5 Quantifiers

One of the reasons why we chose Z3 as the back end constraint solver was because it is able to have quantifiable logic such as the universal quantifier which is interpreted as “for all” $\forall$. The universal quantifier asserts that all predicates within the scope of the quantifier must be true of every value of the predicate. In Z3, they are represented as:

- `(assert (forall ((x Int)) (x > 0)))` which in first order predicate logic is $\forall x : (x > 0)$. So for all integers, they must be greater than zero.

2.6.6 Satisfiability and Validity

Validity

A formula f is valid if f always evaluates to true for any assignment to an appropriate value.

Satisfiability

A formula f is satisfiable if there is some assignment to an appropriate value to the function where f evaluates to true.
As we mentioned previously, Z3 SMT Solver gives back the satisfiability of the interpretation. It has three states when the (check-sat) command is called:

- `sat` - satisfied model, a model can be returned. We give an example of a satisfied formula in Figure 2.5a.
- `unsat` - unsatisfied model - a model cannot be returned. We give an example of an unsatisfied formula in Figure 2.5b.
- `unknown` - when Z3 does not know whether a formula is satisfiable or not. We give an example of an unknown result in Figure 2.5c.

What is good about whether a formula is satisfiable is that it is about finding a solution under a set of constraints.

```
(declare-const x Int)
(assert (> x 10))
(assert (< x 100))
(check-sat)
(get-model)
```

(a) Z3 with satisfied core

```
(declare-const a Int)
(assert (> a 10))
(assert (< a 10))
(check-sat)
(get-model)
```

(b) Z3 with unsatisfied core

```
(declare-const a Int)
(assert (> a 10))
(assert (< a 10))
(check-sat)
(get-model)
```

(c) Z3 with unknown

Figure 2.5: Predicate Logic with stack

Chapter 3

Business Rules

Chapter 4

Implementation

Chapter 5

Evaluation

Chapter 6

Conclusion

Chapter 7

Appendix A: How to Use

Bibliography

[1] Mark Senn. *Acyclic Digraph*, (accessed February, 2015). `http://mathworld.wolfram.com/AcyclicDigraph.html`.

[2] Leonardo de Moura and Nikolaj Bjrner. Satisfiability modulo theories: An appetizer. pages 1–8, 2009. `http://research.microsoft.com/en-us/um/people/leonardo/sbmf09.pdf` (accessed Feburary, 2015).

[3] Carla P. Gomes, Henry Kautz, Ashish Sabharwal, and Bart Selmanr. Satisfiability solvers. *Handbook of Knowledge Representation*, pages 91–92, 2008. `http://www.cs.cornell.edu/gomes/papers/satsolvers-kr-handbook.pdf` (accessed Feburary, 2015).

[4] Microsoft Research. *Z3 Codeplex*, 2014. `http://z3.codeplex.com/`, (accessed February, 2015).

[5] SRI International's Computer Science Laboratory. *The Yices SMT Solver*, 2015. `http://yices.csl.sri.com/`, (accessed May, 2015).

[6] *CVC4 - The SMT Solver*, 2015. `http://cvc4.cs.nyu.edu/web/`, (accessed May, 2015).

[7] Alessandro Cimatti, Alberto Griggio, Bastiaan Schaafsma, and Roberto Sebastiani. The MathSAT5 SMT Solver. In Nir Piterman and Scott Smolka, editors, *Proceedings of TACAS*, volume 7795 of *LNCS*. Springer, 2013. `http://mathsat.fbk.eu/index.html`, (accessed May, 2015).