A Formalisation of the Soccer Substitution Rules

Yves Ledru

Université Joseph Fourier - Grenoble 1
Laboratoire Logiciels, Systèmes, Réseaux - IMAG
B.P. 72 - F-38402 - Saint Martin d'Hères Cedex - France
Yves.Ledru@imag.fr

Abstract. This paper presents a formal model of the substitution rules for soccer games as they existed at the 1994 World Cup. The model is expressed in VDM and can be animated with the VDMTools environment. The formalisation helps improve the precision of the original rules, stated in natural language. This animation shows that the rules make a useless distinction between goalkeeper and field player substitutions.

1 Introduction - Motivating example

During the 1994 Soccer World Cup, a problem arose about the interpretation of the substitution rules when Italy's goalkeeper was excluded during a match against Norway. At that time, the rules only allowed to substitute one goalkeeper and two field players. But the definition of "goalkeeper substitution" remained informal. The usual meaning of a goalkeeper substitution is when the actual goalkeeper leaves the field and is replaced by another goalkeeper. Unfortunately, there are times when a field player leaves the field and is replaced by a goalkeeper. Should such a substitution be counted as a goalkeeper or field player substitution?

What actually happened during the Italy-Norway match is the following:

1. The italian goalkeeper Pagliuca (number 1) is excluded.
2. Baggio (number 10) exits the field and is substituted by the substitute goalkeeper (number 12).
3. Two further substitutions of field players are performed.

If the first substitution is counted as a field player substitution, then the third one was not allowed!

This paper will present one possible formalisation of the substitution rules and show how the actual sequence of events of the 1994 Italy-Norway fits in these rules. The paper is intended to give an entertaining example of the potential usefulness of formalising the informal rules which govern human activities.

2 The informal soccer substitution rules

This paper models the referee's book for a soccer game. Its goal is to model the rules for the substitution of players during a game.

For a given team, the following rules apply during a match:

- A soccer team consists of up to eleven players and a set of substitutes.
- At most one of the players is the goal-keeper.
- The rules of soccer allow for the substitution of a player by one of the substitutes.
- Once a player has been replaced by another, he may no longer take part to the match.
- There is a maximum number of allowed substitutions (in 1994, one goal keeper and two field players).
- The referee may exclude a player (including the substitutes).
- The role of goalkeeper may be transfered from one player to another, provided the referee is notified about this transfer.

3 The VDM specification

Our model uses the VDM specification language. VDM is an ISO standard for software specification [3, 4, 2]. A VDM specification is composed of two main parts:

- A state is described by several variables. These variables may be constrained by invariant properties.
- Operations modify the state. These operations are specified by pre- and post-conditions.

Part of the language is executable and supported by a suite of industrial tools named VDMTools [1]. The specification presented here was initially prototyped with KIDS/VDM [5], then adapted to VDMTools.

3.1 Constants, types and state variables

Two constants are introduced to denote the maximum numbers of substitutions for goalkeepers (`gk-subs-max`) and field players (`fp-subs-max`). Type `player` is introduced as a renaming for natural numbers.

```
values gk_subs_max : nat = 1;
       fp_subs_max : nat = 2
types  player = nat
```

The state of the soccer team, as it appears in the referee's book, may be abstracted to five variables:

- the set of players on the field
- the set of potential substitutes
- the player who is the goalkeeper[1]

[1] The goalkeeper is usually a member of the players on the field, but not always, e.g. he can be excluded by the referee.

- the number of goalkeeper substitutions already performed
- the number of field player substitutions already performed

```
state  R_Book of
         on_field_players : set of player
         potential_substitutes : set of player
         goalkeeper : player
         nb_gk_subs : nat
         nb_fp_subs : nat
inv mk_R_Book(ofp,ps,gk,ngk,nfp) ==
         (card ofp) <= 11
         and  (ngk <= gk_subs_max) and (nfp <= fp_subs_max)
         and gk not in set ps
         and ofp inter ps = {}
init r == r = mk_R_Book({1,2,3,4,5,6,7,8,9,10,11},
                        {12,13,14,15,16}, 1, 0, 0)
end
```

The state invariant expresses that there are at most eleven players of the team on the field, and that the numbers of performed substitutions are less than or equal to the maxima allowed. It also states that the goalkeeper is not within the substitutes. Finally, the invariant states that a player can not simultaneously be on field and substitute. The last lines state the initial values, which are the usual ones in soccer matches.

3.2 Operations

There are three operations allowed on this state:

- the referee gives a red card to exclude one of the players,
- the goalkeeper role is transfered to another field player,
- a player is substituted by another player.

The `RED-CARD` operation takes the excluded player as argument. The player may be any of the team players, so the pre-condition states that he is member of one of both sets `on-field-players` and `potential-substitutes`. The post-condition states that he no longer appears in any of these sets and that everything else remains unchanged. Operations in VDM-SL include an implicit part where a pre-condition and a post-condition are stated, and an explicit part which is actually code to be executed by the operation. The VDM tools check at execution time that the execution of the operation conforms to the pre and post-conditions, as well as to the state invariant.

```
operations
RED_CARD :  player ==> ()
RED_CARD (p) ==
(
on_field_players := on_field_players \ {p};
potential_substitutes := potential_substitutes \ {p}
)
pre p in set on_field_players or p in set potential_substitutes
post on_field_players = on_field_players~ \ {p}
       and potential_substitutes = potential_substitutes~ \ {p}
      ;
```

The second operation `CHANGE-GOALKEEPER` expresses that one of the field players takes the role of goalkeeper. The pre-condition states that the player is on the field (not really mandatory, but often useful) and the post-condition that he is the new goalkeeper.

```
CHANGE_GOALKEEPER : player ==> ()
CHANGE_GOALKEEPER (p) ==
(
goalkeeper := p
)
pre p in set on_field_players
post goalkeeper = p
      ;
```

The last operation models the substitution of a player by another one. Depending on the role of the player who quits the field, the relevant variable (`nb-gk-subs` or `nb-fp-subs`) is updated. Actually, since our model does not allow the goalkeeper to be a substitute, the choice to update `nb-gk-subs` or `nb-fp-subs` may only depend on the role of the player that leaves the field. The pre-condition states that the player is on the field, that the substitute is a valid substitute, and that the maximum number of substitutions has not yet been reached. The post-condition states that the substitute is on the field and that `pl` no longer participates to the match. It also states that `subs` is the new goalkeeper if `pl` was goalkeeper. Finally, it updates the substitution counters.

```
SUBSTITUTION : player * player ==> ()
SUBSTITUTION (pl, subs) ==
(
on_field_players := on_field_players union {subs} \ {pl};
potential_substitutes := potential_substitutes \ {subs};
if pl = goalkeeper then
             (goalkeeper := subs;
              nb_gk_subs := nb_gk_subs +1)
        else (nb_fp_subs := nb_fp_subs +1)
)
pre pl in set on_field_players and subs in set potential_substitutes
       and (pl = goalkeeper => (nb_gk_subs+1 <= gk_subs_max))
       and (pl <> goalkeeper => (nb_fp_subs+1 <= fp_subs_max))
post on_field_players = on_field_players~ union {subs} \ {pl}
       and potential_substitutes = potential_substitutes~ \ {subs}
       and (pl = goalkeeper~ =>
             ((goalkeeper = subs)
                 and (nb_gk_subs = nb_gk_subs~ +1 )
                 and (nb_fp_subs = nb_fp_subs~)))
       and (pl <> goalkeeper~ =>
                ((goalkeeper = goalkeeper~)
                 and (nb_gk_subs = nb_gk_subs~)
                 and (nb_fp_subs = nb_fp_subs~ +1)))
;
```

4 Model execution and validation

The VDMTools environment proposes a validation approach based on animation and test of the specification. Animation is based on the execution of the explicit parts of operations, starting from the initial state. Validation can be carried out both informally and formally:

- An informal validation looks at the behaviour of the model and checks that it corresponds to the expected results. This activity may be supported by the definition of several test cases which correspond to expected or forbidden behaviours. For example, one can check that the model allows up to two substitutions of field players and rejects a third one.
- A formal validation mechanism is built in the tool: it checks that invariants and pre-conditions are verified in the initial state of an operation call, and that invariants and post-conditions are verified in their final state. This is mainly a consistency check: the explicit parts of the specification actually implement the constraints of the implicit parts.

VDMTools don't include a test generator: the animated sequences are thus designed by the analyst based on his understanding of the model and of the requirements.

4.1 Italy vs Norway revisited

We are now able to analyse the Italy-Norway game by executing the model with the VDM tools. It reveals that the following sequence of operations is invalid:

```
RED_CARD(1)
SUBSTITUTION(10,12)
SUBSTITUTION(2,13)
SUBSTITUTION(3,14)
Run-Time Error 58: The pre-condition evaluated to false
```

Actually, three field players have left the game. Moreover, Pagliuca (player 1) has remained goalkeeper for the whole match!

A valid sequence is:

```
RED_CARD(1)
CHANGE_GOALKEEPER(10)
SUBSTITUTION(10,12)
SUBSTITUTION(2,13)
SUBSTITUTION(3,14)
```

So, provided this formalisation captures the semantics of the soccer substitution rules, Roberto Baggio has exited the match as being the goalkeeper, and the remaining substitutions of Italy-Norway were valid!

Actually, the fact that it is possible to change the goalkeeper at any time allows to make three substitutions of field players, like in the following sequence:

```
SUBSTITUTION(2,13)
SUBSTITUTION(3,14)
CHANGE_GOALKEEPER(4)
SUBSTITUTION(4,15)
CHANGE_GOALKEEPER(1)
```

In this sequence, player 4 exits as being the goalkeeper, but as soon as the substitution has taken place, the original goalkeeper (player 1) is restored.

Such a formal model, and this counter-example, show that the distinction between goalkeeper and field player does not make sense for substitutions. Actually, the FIFA (international soccer federation) simplified its substitution rules short after the 1994 World Cup, allowing three substitutions of players during a match without distinction between field players and goalkeepers.

5 Conclusion

This paper has presented a formalisation of the rules of the soccer game related to the substitution of players. An example, taken from the 1994 World Cup shows that the original rules may lead to several diverging interpretations. It

also shows that rules may lead to unexpected interactions: here the rules related to exclusion of a player interfere with the rules related to player substitution.

Providing a formal model leads to a more rigorous application of the rules. In the Italy-Norway example, one would have expected that the goal keeper change which took place before the first substitution would have been notified to the referee. It also allows to experiment with the model, which leads here to the demonstration that the distinction between goalkeeper and field player substitutions does not make sense.

Finding ways to detect errors in the model is an interesting field of research. In this paper, the counter examples were discovered after a careful study of the model. But tools based on test generation and model-checking techniques can also be used during model validation.

This case study was primarily meant to be didactical and illustrative, but I hope that it shows the usefulness of formalising human rules and experimenting with executable models in order to find their weaknesses.

Acknowledgments Thanks to Marie-Laure Potet and the REMO2V reviewers for their comments on a earlier versions of this document.

Part of this work is supported by the EDEMOI project, sponsored by the ACI Sécurité Informatique of the French Ministry of Research.

References

1. CSK. VDMTools - The VDM-SL Language. Technical report, CSK, 2005. http://www.vdmtools.jp/files/langmansl_a4.pdf.
2. John Fitzgerald and Peter Gorm Larsen. *Modelling Systems – Practical Tools and Techniques in Software Development*. Cambridge University Press, The Edinburgh Building, Cambridge CB2 2RU, UK, 1998. ISBN 0-521-62348-0.
3. ISO. *Information Technology — Programming Languages, their environments and system software interfaces — Vienna Development Method-Specification Language Part 1: Base language*, 1996.
4. C. B. Jones. *Systematic Software Development Using VDM*. Prentice-Hall, London, 1986.
5. Y. Ledru. Using KIDS as a tool support for VDM. In *Proceedings of the 18th International Conference on Software Engineering*, pages 236–245. IEEE Computer Society Press, 1996.