Generic implementations of process calculi in Isabelle

Jesper Bengtson

31st October 2004

Abstract

Many process calculi have aspects in common, such as parallel composition, name passing and notions of bisimilarity. This abstract covers the beginning of an attempt to create a generic library for these calculi in the automatic theorem prover Isabelle. The goal of the project is to make the library powerful enough to verify both extensions of existing calculi as well as completely new ones.

Contents

1 Introduction **4**

2 Modelling CCS in Isabelle **4**
2.1 Defining the operational semantics 4
2.2 Weakening the transitions 4
2.3 Bisimulation and congruences 4
2.4 Future work 5

1 Introduction

There are several tools available to perform verification on process calculi: *The Concurrency Workbench* [2] for CCS and *The Mobility Workbench* [7] for the π-calculus to name a few. These are specialised tools which can perform preset tasks such as model- or bisimulation-checking of agents. They work well. They are fast and automatic as long as the agents they are working on have a finite state space. Using theorem provers, like Isabelle [5], it is possible to model agents with infinite state space. Moreover, it is possible to perform proofs on the actual calculi themselves. It must be stressed that these provers are not automatic – Isabelle needs to be "hand held" to a larger degree and the two methods would complement each other.
One of the main goals of the project is to make a generic library in Isabelle to reason with different varieties of process calculi – different calculi often have aspects in common, such as parallel composition, name passing and notions of bisimilarity. If a generic library can be created in Isabelle, new operators or even entirely new calculi could be added and verified reusing the old proofs.

2 Modelling CCS in Isabelle

The first step has been to model CCS in Isabelle. The operational semantics (except for relabelling) have been added as have strong and weak bisimilarity as well as equivalence of agents. Isabelle has also been used to prove that strong bisimilarity is a congruence.

2.1 Defining the operational semantics

The operational semantics of CCS has been implemented in Isabelle using an inductively defined set. The set consists of tuples of the form (P, α, P') where P is an agent, α is an action, and P' is an α-derivative of P. Every rule in the operational semantics is used as an inductive rule for the set with the base case of the semantics representing the base case of the set definition. The operational rules with their corresponding set rules can be found in table 1.

2.2 Weakening the transitions

The operational semantics described above is used when defining strong bisimilarity. In order to describe weak bisimilarity and equivalence the transitions must be weakened somewhat by allowing for hidden τ-transitions, also called silent actions. The transitions for weak bisimilarity and equivalence are $\stackrel{\hat{\alpha}}{\Longrightarrow}$ and $\stackrel{\alpha}{\Longrightarrow}$ respectively where $\stackrel{\alpha}{\Longrightarrow}$ specifies at least the τ-actions occurring in α and $\stackrel{\hat{\alpha}}{\Longrightarrow}$ specifies nothing about the τ-actions occurring in α [3]. The operational semantics for the two transitions can be found in table 2. They are modelled in Isabelle using inductively defined sets – just as the regular transition system.

2.3 Bisimulation and congruences

Two processes are said to be bisimilar if they can mimic each others actions. The bisimulation relations are co-inductive ones and are extensively covered in

Table 1: The operational semantics for CCS and the construction rules for the inductively defined set. The symmetric versions (of the rules marked with †) have been elided. These inductive definitions are not exactly the ones used in Isabelle. They have been altered somewhat to make them easier to read.

[**Act**]	$\frac{}{\alpha.P \xrightarrow{\alpha} P}$	$(\alpha.P, \alpha, P) \in actsSet$
[**Sum**]	$\frac{P \xrightarrow{\alpha} P'}{P+Q \xrightarrow{\alpha} P'}$ †	$(P, \alpha, P') \in actsSet \Longrightarrow (P + Q, \alpha, P') \in actsSet$
[**Com**]	$\frac{P \xrightarrow{\alpha} P'}{P\|Q \xrightarrow{\alpha} P'\|Q}$ †	$(P, \alpha, P') \in actsSet \Longrightarrow (P\|Q, \alpha, P'\|Q) \in actsSet$
[**Com₃**]	$\frac{P \xrightarrow{\alpha} P' \quad Q \xrightarrow{\bar{\alpha}} Q'}{P\|Q \xrightarrow{\tau} P'\|Q'}$	$[(P, \alpha, P') \in actsSet; (Q, \bar{\alpha}, Q') \in actsSet] \Longrightarrow$ $(P\|Q, \tau, P'\|Q') \in actsSet$
[**Res**]	$\frac{P \xrightarrow{\alpha} P'}{\nu(\beta)P \xrightarrow{\alpha} \nu(\beta)P'}$ $\alpha \notin \{\beta, \bar{\beta}\}$	$[(P, \alpha, P') \in actsSet; \alpha \notin \{\beta, \bar{\beta}\}] \Longrightarrow$ $(\nu(\beta)P, \alpha, \nu(\beta)P') \in actsSet$

Table 2: The operational rules for $\stackrel{\alpha}{\Longrightarrow}$ and $\stackrel{\hat{\alpha}}{\Longrightarrow}$.

$$\stackrel{\alpha}{\Longrightarrow} \equiv \frac{P \xrightarrow{\alpha} P'}{P \stackrel{\alpha}{\Longrightarrow} P'} \quad \frac{P \stackrel{\alpha}{\Longrightarrow} P' \quad P' \stackrel{\tau}{\Longrightarrow} P''}{P \stackrel{\alpha}{\Longrightarrow} P''} \quad \frac{P \stackrel{\tau}{\Longrightarrow} P' \quad P' \stackrel{\alpha}{\Longrightarrow} P''}{P \stackrel{\alpha}{\Longrightarrow} P''}$$

$$\stackrel{\hat{\alpha}}{\Longrightarrow} \equiv \frac{}{P \stackrel{\hat{\alpha}}{\Longrightarrow} P} \ \alpha = \tau \quad \frac{P \stackrel{\alpha}{\Longrightarrow} P'}{P \stackrel{\hat{\alpha}}{\Longrightarrow} P'}$$

[3]. Isabelle not only supports inductively defined sets but co-inductively defined ones as well so creating co-inductively defined sets containing the tuples of all bisimilar agents comes very natural. The way they are modelled can be found in Table 3.
The proofs that have been done so far are proofs that strong bisimulation is a congruence – that is an equivalence relation and substitutive under all combinators. We are also one transitivity proof short of proving that equivalence is a congruence. Since bisimulation is a co-inductive definition the proofs have been made by reasoning about the greatest fixed point – adding all possible derivatives of a combinator to the bisimulation relation should preserve the greatest fixed point. Weak bisimulation is not a congruence as the **Sum** rule breaks this premise.

2.4 Future work

The goal of this project is to make a generic library for reasoning about process calculi in Isabelle. What most notably is lacking at the moment is name passing – something which is very common in other calculi. The next step will be to extend the library to include the π-calculus [4] which is a calculus for mobile processes which supports the passing of names across links. This becomes problematic as notions of α-equivalence has to be taken into account. Some work

Table 3: The co-inductive definition of strong bisimulation $\sim$, weak bisimulation $\approx$ and equivalence $=$. Again, for readability reasons, this is not exact Isabelle notation.

$$\begin{array}{rcl} \sim & \equiv & [\forall \alpha Q'.Q \xrightarrow{\alpha} Q' \longrightarrow (\exists P'.P \xrightarrow{\alpha} P' P' \sim Q'; \\ & & \forall \alpha P' P \xrightarrow{\alpha} P' \longrightarrow (\exists Q'.Q \xrightarrow{\alpha} Q' P' \sim Q'] \Longrightarrow P \sim Q \end{array}$$

$$\begin{array}{rcl} \approx & \equiv & [\forall \alpha Q'.Q \xrightarrow{\alpha} Q' \longrightarrow (\exists P'.P \stackrel{\hat{\alpha}}{\Longrightarrow} P' P' \approx Q'; \\ & & \forall \alpha P' P \xrightarrow{\alpha} P' \longrightarrow (\exists Q'.Q \stackrel{\hat{\alpha}}{\Longrightarrow} Q' P' \approx Q'] \Longrightarrow P \approx Q \end{array}$$

$$\begin{array}{rcl} = & \equiv & [\forall \alpha Q'.Q \xrightarrow{\alpha} Q' \longrightarrow (\exists P'.P \stackrel{\alpha}{\Longrightarrow} P' P' \approx Q'; \\ & & \forall \alpha P' P \xrightarrow{\alpha} P' \longrightarrow (\exists Q'.Q \stackrel{\alpha}{\Longrightarrow} Q' P' \approx Q'] \Longrightarrow P = Q \end{array}$$

has already been done in this area by Röckl and Hirschkoff [6] and collaboration with them is underway. Care must also be taken not to make the library to specialised. A few attempts have already been made to add operators from CSP [1] to the existing implementation with good results.

References

[1] S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A theory of communicating sequential processes. *J. ACM*, 31(3):560–599, 1984.

[2] Rance Cleaveland, Joachim Parrow, and Bernhard Steffen. The concurrency workbench: a semantics-based tool for the verification of concurrent systems. *ACM Trans. Program. Lang. Syst.*, 15(1):36–72, 1993.

[3] R. Milner. *Communication and Concurrency*. Prentice-Hall, Inc., 1989.

[4] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, i/ii. *Inf. Comput.*, 100(1):1–77, 1992.

[5] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL: a proof assistant for higher-order logic*. Springer-Verlag, 2002.

[6] C. Röckl and D. Hirschkoff. A fully adequate shallow embedding of the π-calculus in Isabelle/HOL with mechanized syntax analysis. *Journal of Functional Programming*, 2003.

[7] Björn Victor and Faron Moller. The Mobility Workbench — a tool for the π-calculus. In David Dill, editor, *CAV'94: Computer Aided Verification*, volume 818 of *Lecture Notes in Computer Science*, pages 428–440. Springer-Verlag, 1994.