Comparing Refinement Settings

Heiko Schmidt

Christian-Albrechts-Universität, Kiel, Germany

Copenhagen, 20 Sept. 2007

Motivation

Compare refinement settings, e.g. *Modal Transition Systems* (MTS) and *Disjunctive Modal Transition Systems* (DMTS)

- Can every set of TSs described by a DMTS also be described by an MTS?
- Can the refinement structure of MTS be embedded into DMTS?
- Which transformations between MTS and DMTS exist that preserve sets of implementations or refinement structure? What are their complexity?

Agenda

- compare refinement settings with respect to relative expressiveness
  - e.g., MTS, DMTS, mixed transition systems, modal automata, μ-automata, transition systems with ready, failure, ready trace, failure trace inclusion, ...
- discuss different comparison approaches via transformations
- present some comparison results

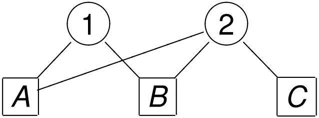
Refinement settings

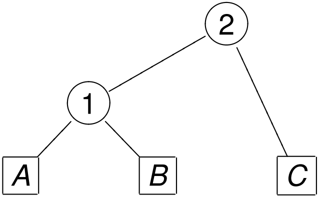
Refinement settings

- a set of models
  - e.g. MTS
- a refinement preorder
- a distinguished subset of *concrete* models, called *implementations*
  - usually the smallest elements of the refinement preorder
  - usually correspond to (deterministic) transition systems
  - refinement preorder coincides with bisimulation equivalence
  - e.g. those MTS with equal may- and must transition relations

Refinement settings are preorders

- preorder induces partial order on refinement equivalence classes
- partial order can be drawn as a Hasse diagram:

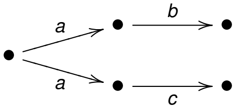




- models: $1, 2, 3, A, B, C$ (refinement equivalence classes)
- implementations: A, B, C (bisimulation equivalence classes)
- refinement preorder (refinements by transitivity not drawn)

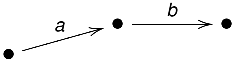
Implementations: TSs vs. deterministic TSs

- refinement settings, where the implementations are...
  - (possibly nondeterministic) transition systems



nondeterminism of implementations is *persistent* (not *resolvable* by refinement)

  - deterministic transition systems



nondeterminism is *resolvable* by refinement

Implementations are (possibly nondeterministic) transition systems

Examples:

- MTS, DMTS, mixed transition systems
- μ-automata, modal automata
- a variant called one-selecting modal transition systems (1MTS) with an exclusive (XOR) interpretation of hypertransitions

Implementations are deterministic transition systems

Examples:

- transition systems with ready simulation
- transition systems with readiness, failure, ready trace, failure trace inclusion
  - T_1 refines T_2 iff every ready/failure/... trace of T_1 is a ready/failure/... trace of T_2
- MTS, DMTS, mixed transition systems, μ-/modal automata

Comparison

Elementwise comparison

- S_i: refinement settings
- $\preceq$: less or equally expressive

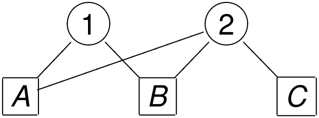
- elementwise comparison only makes sense, if the compared settings have the same models

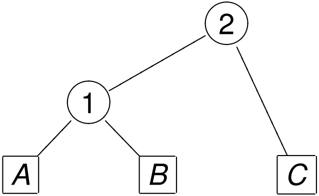
Definition

$S_1 \preceq S_2$ iff every refinement pair in S_1 is a refinement pair in S_2

$$\begin{aligned} S_1 \preceq S_2 \quad &\iff \quad \leq_{S_1} \subseteq \leq_{S_2} \\ &\iff \quad \forall M, M' : M \leq_{S_1} M' \Rightarrow M \leq_{S_2} M' \end{aligned}$$

Elementwise comparison





- $\preceq$? Yes!
- $\succeq$? No! 1 $\leq_{S_2}$ 2, but 1 $\not\leq_{S_1}$ 2
- right setting is more expressive

Implementation-based comparison

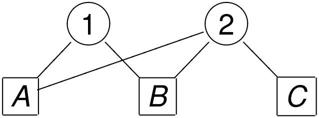
Definition

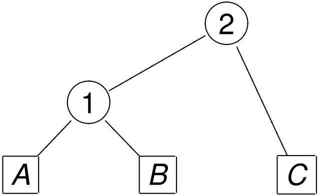
$S_1 \preceq S_2$ iff for every model M_1 in S_1 there is a model M_2 in S_2 such that the set of implementations refining M_1 equals the set of implementations refining M_2

$$\begin{aligned} S_1 \preceq S_2 \quad &\iff \quad \forall M_1 \in S_1 : \exists M_2 \in S_2 : impl(M_1) = impl(M_2) \\ &\iff \quad \exists f : S_1 \to S_2 : \\ &\qquad\quad \forall M_1 \in S_1 : impl(M_1) = impl(f(M_1)) \end{aligned}$$

- such a function f is called *implementation-based embedding*

Implementation-based comparison





- $\preceq$? Yes!
- $\succeq$? Yes!
- settings are equally expressive

Preorder-based comparison: Homomorphism

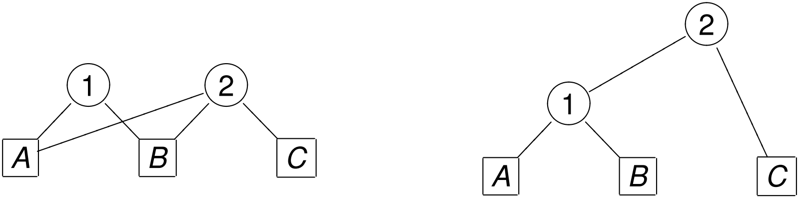
Definition

$S_1 \preceq S_2$ iff there is a *preorder-based homomorphism* $f : S_1 \to S_2$, i.e.,

1. f is monotonic, i.e., $\forall M_1, M_1' : M_1 \leq_{S_1} M_1' \Rightarrow f(M_1) \leq_{S_2} f(M_2)$
2. f keeps implementations fixed, i.e., for all implementations I_1, we have $I_1 \approx f(I_1)$

- $\approx$ is usually bisimulation equivalence, which coincides with refinement on implementations
- elementwise comparison is the special case $f = id$

Preorder-based comparison: Homomorphism



- $\preceq$? Yes! *id* is a preorder-based homomorphism
  (this comparison is a generalization of the elementwise comparison!)
- $\succeq$? Yes! $f : 1 \mapsto 2, 2 \mapsto 2$ is a preorder-based homomorphism
- settings are equally expressive

Preorder-based comparison: Embedding

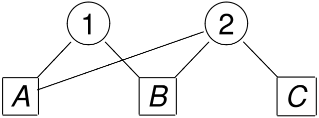
Definition

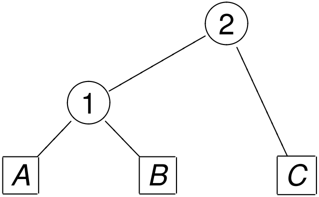
$S_1 \preceq S_2$ iff there is a *preorder-based embedding* $f : S_1 \to S_2$, i.e.,

1. $\forall M_1, M_1' : M_1 \leq_{S_1} M_1' \iff f(M_1) \leq_{S_2} f(M_2)$
2. f keeps implementations fixed, i.e., for all implementations I_1, we have $I_1 \approx f(I_1)$

- every preorder-based embedding is also an implementation-based embedding

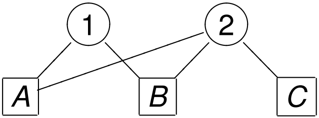
Preorder-based comparison: Embedding

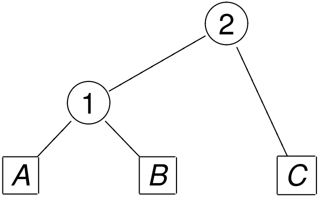




- $\preceq$? No! *id* does not work, because $1 \leq_{S_2} 2$. So try $f : 1 \mapsto 2, 2 \mapsto 2$? Does not work, because $C \leq_{S_2} 2$, but $C \not\leq_{S_1} 1$
- $\succeq$? No! The homomorphism $f : 1 \mapsto 2, 2 \mapsto 2$ from before is no embedding, because $C \leq_{S_1} 2$, but $C \not\leq_{S_2} 1$
- settings are incomparable

What a mess!





- Elementwise comparison: *right is more expressive*
- Implementation-based approach: *equally expressive*
- Preorder-based homomorphism approach: *right is more expressive*
- Preorder-based embedding approach: *incomparable*

Applicability and applications

- Elementwise comparison
  - Pro: clear and simple definition
  - Pro: checking refinement in a setting with less refinement pairs could be easier
  - Con: only settings based on the same models can be compared
  - Con: restriction to identity function, structures only isomorphic are not identified

Applicability and applications

- Implementation-based comparison
  - Pro: clear and simple definition
  - Pro: suitable for applications that are based only on implementations (e.g. generalized model checking)
  - Con: refinement structure not captured at all
  - Pro: a change in the refinement structure can be desirable: checking thorough refinement ($impl(M_1) \subseteq impl(M_2)$?): if checking approximated refinement fails in current setting $\longrightarrow$ apply a refinement-structure-changing transformation and re-check

Applicability and applications

- Preorder-based comparison (homomorphism/embedding)
  - Pro: takes refinement structures into account
    - important property of a refinement setting, e.g. for stepwise refinement or abstraction
    - algorithm/tool reuse (*complexity!*)
    - theoretical results carry over
  - Con: definition more complicate:
    - implementations need to remain fixed... why?
    - existence of a homomorphism only... significance?

Some results

DMTS/1MTS

DMTS/1MTS

- *disjunctive modal transition systems (DMTS)*
  - interpret hypertransitions disjunctively (OR)
- *1-selecting modal transition systems (1MTS)*
  - interpret hypertransitions exclusively (XOR)
- Do we increase expressiveness using the alternative refinement?
- No wrt. implementation-based comparison (equally expressive)
- Yes wrt. preorder-based comparison

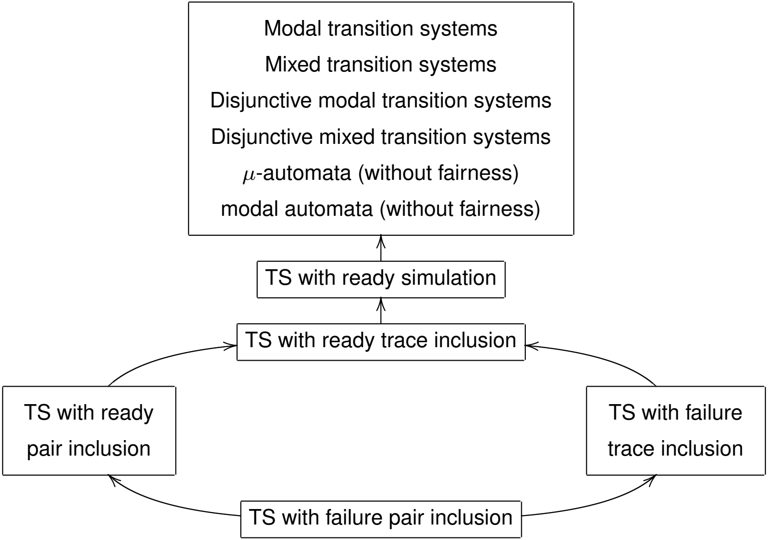
Some results

Comparison wrt. deterministic transition systems

Comparison wrt. deterministic transition systems

- various refinement settings
- implementations are deterministic transition systems
- implementation-based comparison

The preorder of refinement settings



Concluding remarks

Conclusion

- comparison via transformations is useful
  - for the theoretical understanding of refinement settings
  - for switching between settings to get the best of different settings: approximation to thorough refinement, algorithms, tools (*complexity!*)
  - to carry over theoretical results (e.g., non-existence)

Which comparison approach?

- elementwise comparison:
  - clear and simple
  - limited in application, because “transformation” must be *id*
- implementation-based comparison:
  - suitable for applications based only on implementations
  - suitable if it is desirable that the refinement structure changes (for a different approximation of thorough refinement)
- preorder-based comparison:
  - takes complete refinement preorder into account

Future Work

- lots of work to do:
  - different comparison approaches (primarily implementation- and preorder-based)
  - various refinement settings (weak refinement not considered so far)
  - implementations: deterministic or not
- understanding better the relevance of different comparison approaches
  - further applications for the different kinds of transformations?
  - is the requirement to keep implementations fixed always suitable?
  - any use for preorder-based homomorphisms?