PKS: Knowledge-based planning with incomplete information and sensing

Ronald P. A. Petrick        Fahiem Bacchus
Department of Computer Science
University of Toronto
Toronto, Ontario, Canada
{rpetrick, fbacchus}@cs.utoronto.ca

ICAPS 2004 System Demonstration Session
Whistler, British Columbia, Canada
June 5, 2004
Overview of the PKS system

- Low-level C library (libPKS) with API
- GUI designed with GTK+ widget toolkit
- Command-line options for batch processing
- Linux/UNIX-based application
Medical domain example

PKS encoding of a classical conditional planning problem

XML-style language provides high-level structure specified in domain descriptor files

Domains encoded using a logic-like language
Plan generated by PKS (DFS)

Conditional plan (tree view)

Database information is accessible for any knowledge state of the plan

Example planning problem is trivial at the knowledge level; has "natural" structure
Plan linearizations (CLI)

A command-line interface (CLI) is also available for browsing low-level plan structures.

Example of plan linearizations considered by postdiction.
UNIX domain example

More complex problem requires use of functions and numerical expressions

E.g., find the number of files in a directory tree with the same name; determine largest file size (if known); track possible locations of the file

Domain specific update rules
Plan generated by PKS (BFS)

Plan: traverse the directory tree and reason about the possibility of the file being in each directory.
Differing goal states

Different goal states track the possible configurations of file locations and function mappings.

<table>
<thead>
<tr>
<th>Database entry</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>indir(icaps, root)</td>
<td>17</td>
</tr>
<tr>
<td>processed(papertex, icaps)</td>
<td>34</td>
</tr>
<tr>
<td>processed(papertex, root)</td>
<td>31</td>
</tr>
<tr>
<td>processed(papertex, tmp)</td>
<td>35</td>
</tr>
<tr>
<td>processed(papertex, planner)</td>
<td>36</td>
</tr>
<tr>
<td>pwd = planner</td>
<td>28</td>
</tr>
<tr>
<td>count = 0</td>
<td>14</td>
</tr>
<tr>
<td>size_unk = 4</td>
<td>42</td>
</tr>
<tr>
<td>processed(papertex, icaps)</td>
<td>34</td>
</tr>
<tr>
<td>processed(papertex, root)</td>
<td>31</td>
</tr>
<tr>
<td>processed(papertex, tmp)</td>
<td>35</td>
</tr>
<tr>
<td>processed(papertex, planner)</td>
<td>36</td>
</tr>
<tr>
<td>pwd = planner</td>
<td>28</td>
</tr>
<tr>
<td>count = 0</td>
<td>14</td>
</tr>
<tr>
<td>size_unk = 3</td>
<td>40</td>
</tr>
<tr>
<td>!indir(papertex, icaps)</td>
<td>29</td>
</tr>
</tbody>
</table>

File (size unknown) located in 4 directories
File (size unknown) located in 3 directories; not in directory icaps
Domain encodings can also use program-style constructs, limited forms of quantification.

```plaintext
$i = 10;
while($i) {
    add(Kf, p($i));
    $count += 1;
    $i = $i - 2;
}

$x = 0;
loopForallK(?x) {
    K(r(?x)) => add(Kf, s(?x));
    print("Binding: ", ?x, "\n");
    $x += 1;
}
print("Added "+$x + " ");
if (existsK(?x) forallK(?y)
    K(q(?x, ?y)) ^ print("Bindings: ", ?x, " ", ?y, "\n")
    print("> yes\n")
else
    print("> no\n");
done = 1;
</init>
```